МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ» (ДГТУ)

Колледж экономики, управления и права

Методические указания по организации практических занятий и самостоятельной работы студентов

по ПМ.03 «Разработка программных продуктов»

МДК.03.03 «Web-программирование»

PHP

Специальность

09.02.04 Информационные системы (по отраслям)

Ростов-на-Дону 2021

Методические указания по организации практических занятий и самостоятельной работы студентов по ПМ.03. Разработка и интеграция программного обеспечения МДК.03.03 Web-программирование

В данных методических указаниях представлены задания и пошаговые инструкции по разработке PHP-скриптов, а также даны задания для самостоятельной работы студентов и вопросы для самоконтроля.

Определяют этапы выполнения работы на практическом занятии, содержат рекомендации по выполнению индивидуальных заданий и образцы решения задач, а также список рекомендуемой литературы.

Разработаны с учетом ФГОС среднего профессионального образования специальности 09.02.04 Информационные системы (по отраслям), предназначены для студентов и преподавателей колледжа.

Автор-составитель: С.В.Шинакова

Одобрены решением учебно-методического совета колледжа и рекомендованы к практическому применению в образовательном процессе.

Протокол № 8 от «30» мая 2021 г

Председатель учебно-методического совета колледжа

_____ С.В.Шинакова

личная подпись

Ответственный за выпуск к.п.н., заместитель директора колледжа И.И.Джужук

СОДЕРЖАНИЕ

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ № 1 – 2

Основы синтаксиса php. Типы данных. Переменные

Цель занятия: формирование навыка в создании сценариев на обработку переменных различных типов.

Этапы выполнения работы:

1. Протестировать скрипты, сделать вывод.
2. Решить задачи, ответить на вопросы.
3. Подготовить работу к защите.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

PHP - это распространенный скриптовый язык с открытым исходным кодом. PHP сконструирован специально для ведения Web-разработок и его код может внедряться непосредственно в HTML.

Структура <?php … ?> - выделяет PHP код.

Если код не заключить в теги, то он будет передаваться без обработки PHP.

Разновидности конструкции тегов php:

1. <? … ?>
2. <% ... %>
3. <script language="php"> …</script>

PHP также допускает короткий открывающий тег <?, однако использовать их нежелательно, так как они доступны только если включены с помощью конфигурационной директивы php.ini short_open_tag, либо если PHP был сконфигурирован с опцией --enable-short-tags. Инструкции разделяются точкой с запятой, закрывающий тег (?>) так же подразумевает конец предложения, поэтому следующие две записи эквивалентны:

<?php Echo "Это текст"; ?>	<?php echo "Это текст" ?>

Echo – команда вывода на экран.

Комментарии:

1. // однострочный
2. # однострочный
3. /*…*/ - многострочные комментарии

Пример:

```
<?php
     Echo "<p>Hello</p>";    //комментарий
     Echo "<p>Hello</p>";    #комментарий
/*
     комментарий - многострочный
/*
?>
```

PHP поддерживает восемь простых типов.

Тип данных	Описание
boolean	Это простейший тип. boolean выражает истинность значения. Он может быть либо **TRUE**, либо **FALSE**
integer	это число из множества ℤ = {..., -2, -1, 0, 1, 2, ...}.
float	число с плавающей точкой, также известное как double
string	Строка - это набор символов, где символ - это то же самое, что и байт. Это значит, что PHP поддерживает ровно 256 различных символов, а также то, что в PHP нет встроенной поддержки Unicode.
array	Массив (тип array) может быть создан языковой конструкцией array().
object	Если object преобразуется в object, он не изменяется. Если значение другого типа преобразуется в object, создается новый экземпляр встроенного класса *stdClass*. Если значение было **NULL**, новый экземпляр будет пустым. Массивы преобразуются в object с именами полей, названными согласно ключам массива и соответствующими им значениям, за исключением числовых ключей, которые не будут доступны пока не проитерировать объект.
NULL	Специальное значение **NULL** представляет собой переменную без значения. **NULL** - это единственно возможное значение типа null. Переменная считается null, если: – ей была присвоена константа **NULL**. – ей еще не было присвоено никакого значения. – она была удалена с помощью unset().

Переменные в PHP представлены знаком доллара с последующим именем переменной. Имя переменной чувствительно к регистру.

Имена переменных соответствуют тем же правилам, что и остальные наименования в PHP. Правильное имя переменной должно начинаться с буквы или символа подчеркивания и состоять из букв, цифр и символов подчеркивания в любом количестве.

Замечание: *$this* - это особая переменная, которой нельзя ничего присваивать.

Функции для работы с переменными

- boolval — Get the boolean value of a variable
- debug_zval_dump — Выводит строковое представление внутреннего значения zend
- doubleval — Псевдоним floatval
- empty — Проверяет, пуста ли переменная
- floatval — Возвращает значение переменной в виде числа с плавающей точкой
- get_defined_vars — Возвращает массив всех определенных переменных
- get_resource_type — Возвращает тип ресурса
- gettype — Возвращает тип переменной
- import_request_variables — Импортирует переменные GET/POST/Cookie в глобальную область видимости

- intval — Возвращает целое значение переменной
- is_array — Определяет, является ли переменная массивом
- is_bool — Проверяет, является ли переменная булевой
- is_callable — Проверяет, может ли значение переменной быть вызвано в качестве функции
- is_double — Псевдоним is_float
- is_float — Проверяет, является ли переменная числом с плавающей точкой
- is_int — Проверяет, является ли переменная переменной целочисленного типа
- is_integer — Псевдоним is_int
- is_long — Псевдоним is_int
- is_null — Проверяет, является ли значение переменной равным NULL
- is_numeric — Проверяет, является ли переменная числом или строкой, содержащей число
- is_object — Проверяет, является ли переменная объектом
- is_real — Псевдоним is_float
- is_resource — Проверяет, является ли переменная ресурсом
- is_scalar — Проверяет, является ли переменная скалярным значением
- is_string — Проверяет, является ли переменная строкой
- isset — Определяет, была ли установлена переменная значением отличным от NULL.

Описание:

```
bool isset ( mixed $var [, mixed $... ] )
```

Если переменная была удалена с помощью unset(), то она больше не считается установленной. **isset()** вернет **FALSE**, если проверяемая переменная имеет значение **NULL**. Следует помнить, что **NULL**-байт (*"\0"*) не является эквивалентом константе PHP **NULL**.

Если были переданы несколько параметров, то **isset()** вернет **TRUE** только в том случае, если все параметры определены. Проверка происходит слева направо и заканчивается, как только будет встречена неопределенная переменная.

- print_r — Выводит удобочитаемую информацию о переменной
- serialize — Генерирует пригодное для хранения представление переменной
- settype — Преобразует переменную к новому типу
- strval — Возвращает строковое значение переменной
- unserialize — Создает PHP-значение из хранимого представления
- unset — Удаляет переменную
- var_dump — Выводит информацию о переменной
- var_export — Выводит в браузер или возвращает интерпретируемое строковое представление переменной

Сравнение типов $x и результатов функций PHP, связанных с типами

Выражение	gettype()	empty()	is_null()	isset()	boolean : *if($x)*
$x = "";	string	TRUE	FALSE	TRUE	FALSE
$x = null;	NULL	TRUE	TRUE	FALSE	FALSE
var $x;	NULL	TRUE	TRUE	FALSE	FALSE
$x не определена	NULL	TRUE	TRUE	FALSE	FALSE
$x = array();	array	TRUE	FALSE	TRUE	FALSE
$x = false;	boolean	TRUE	FALSE	TRUE	FALSE
$x = true;	boolean	FALSE	FALSE	TRUE	TRUE
$x = 1;	integer	FALSE	FALSE	TRUE	TRUE
$x = 42;	integer	FALSE	FALSE	TRUE	TRUE
$x = 0;	integer	TRUE	FALSE	TRUE	FALSE
$x = -1;	integer	FALSE	FALSE	TRUE	TRUE
$x = "1";	string	FALSE	FALSE	TRUE	TRUE
$x = "0";	string	TRUE	FALSE	TRUE	FALSE
$x = "-1";	string	FALSE	FALSE	TRUE	TRUE
$x = "php";	string	FALSE	FALSE	TRUE	TRUE
$x = "true";	string	FALSE	FALSE	TRUE	TRUE
$x = "false";	string	FALSE	FALSE	TRUE	TRUE

Предопределённые переменные

PHP предоставляет всем скриптам большое количество предопределённых переменных. Эти переменные содержат всё, от внешних данных до переменных среды окружения, от текста сообщений об ошибках до последних полученных заголовков.

- Суперглобальные переменные — Суперглобальные переменные - это встроенные переменные, которые всегда доступны во всех областях видимости
- $GLOBALS — Ссылки на все переменные глобальной области видимости
- $_SERVER — Информация о сервере и среде исполнения. Переменная *$_SERVER* - это массив, содержащий информацию, такую как заголовки, пути и местоположения скриптов. Записи в этом массиве создаются веб-сервером.

Пример #1 Пример использования *$_SERVER*

```
<?php
  echo $_SERVER['SERVER_NAME'];
?>
```

Результатом выполнения данного примера будет что-то подобное: www.example.com

- $_GET — GET-переменные HTTP
- $_POST — HTTP POST variables
- $_FILES — Переменные файлов, загруженных по HTTP
- $_REQUEST — Переменные HTTP-запроса
- $_SESSION — Переменные сессии
- $_ENV — Переменные окружения
- $_COOKIE — HTTP Куки
- $php_errormsg — Предыдущее сообщение об ошибке
- $HTTP_RAW_POST_DATA — Необработанные POST-данные
- $http_response_header — Заголовки ответов HTTP

- $argc — Количество аргументов переданных скрипту
- $argv — Массив переданных скрипту аргументов

Ссылки на переменные

По умолчанию, переменные всегда присваиваются по значению. То есть, когда вы присваиваете выражение переменной, все значение оригинального выражения копируется в эту переменную. Это означает, к примеру, что, после того как одной переменной присвоено значение другой, изменение одной из них не влияет на другую.

PHP также предлагает иной способ присвоения значений переменным: присвоение по ссылке. Это означает, что новая переменная просто ссылается (иначе говоря, "становится псевдонимом" или "указывает") на оригинальную переменную. Изменения в новой переменной отражаются на оригинале, и наоборот.

Важно отметить, что по ссылке могут быть присвоены только именованные переменные.

```
<?php
    $foo = 25;
    $bar = &$foo;      // Это верное присвоение.
    echo $bar;
  $bar = &(24 * 7);  // Неверно; ссылка на неименованное выражение.
?>
```

Для присвоения по ссылке, просто добавьте амперсанд (&) к началу имени присваиваемой (исходной) переменной. Например, следующий фрагмент кода дважды выводит '*Меня зовут Боб*':

```
<?php
        $foo = 'Боб';              // Присваивает переменной $foo значение 'Боб'
        $bar = &$foo;              // Ссылка на $foo через $bar.
        $bar = "Меня зовут $bar";  // Изменение $bar...
        echo $bar;
        echo $foo;                 // меняет и $foo.
?>
```

Хотя в PHP и нет необходимости инициализировать переменные, это считается очень хорошей практикой. Неинициализированные переменные принимают значение по умолчанию в зависимости от их типа, который определяется из контекста их первого использования: булевы принимают значение FALSE, целые и числа с плавающей точкой - ноль, строки (например, при использовании в echo) - пустую строку, а массивы становятся пустыми массивами.

Динамические переменные

Иногда бывает удобно иметь переменными имена переменных. То есть, имя переменной, которое может быть определено и изменено динамически. Обычная переменная определяется примерно таким выражением:

```
<?php
    $a = 'hello';
?>
```

Переменная переменной берет значение переменной и рассматривает его как имя переменной. В вышеприведенном примере *hello* может быть использовано как имя переменной при помощи двух знаков доллара. То есть:

```
<?php
```

```
    $hello =  'world';
     echo $$a;
?>
```

Теперь в дереве символов PHP определены и содержатся две переменные: *$a*, содержащая «hello», и *$hello*, содержащая "world". Таким образом, выражение

```
<?php
    echo "$a ${$a}";
?>
```

выведет то же, что и

```
<?php
    echo "$a $hello";
?>
```

то есть, они оба выведут: hello world.

ХОД РАБОТЫ 1 занятия

Выполните следующие задания.

1. Какие примеры названий переменных верные:
$a, #color, _X, $7users, $max_age, @print, $print, $A?
2. Что будет в результате выполнения скрипта:

```
<?php
      $a=34;
      $A=100;
     echo “a=”.$a;
     echo “A=”.$A;
     echo $a+$A;
?>
```

3. Что отобразится на экране:

```
<?php
      $y=5;
      echo $y;
      $float_=5.47;
      echo $float_; echo ”<br>”;
      $real=5.47e2;
      echo $real; echo ”<br>”;
      $double_=547e-3;
      echo $double_; echo “<br>”;
      $bool=true;
      echo “переменная равна TRUE:  ”.$bool; echo “<br>”;
      $bool2=False;
      echo “переменная равна False:  ”.$bool2; echo “<br>”;
?>
```

4. Что отобразится на экране:

```
<?php
      $oct=0547;
      echo $oct; echo <”br”>;
      $dec=547;
      echo $dec; echo <”br”>;
      $hex=0547;
```

```
    echo $hex; echo <”br”>;
?>
```

5. Что отобразится на экране:

```
<?php
    $stroka= “Вторая переменная пустая строка”;
    echo $stroka; echo < “br”>;
    $s= “”;
    echo $s; echo <“br”>;
?>
```

6. Определить значение каждой переменной скрипта:

```
<?php

$p = '';

if (isset($p))
       {
          echo "Эта переменная определена, поэтому напечатана.";
       }

      $a = "test";
      $b = "anothertest";

      var_dump(isset($a));
      var_dump(isset($a, $b));

      unset ($a);

      var_dump(isset($a));
      var_dump(isset($a, $b));

      $foo = NULL;
      var_dump(isset($foo));
?>
```

7. Определить значение каждой переменной скрипта:

```
<?php
    $a_bool = TRUE;
    $a_str  = "567";
    $a_str2 = '567';
    $an_int = 12;
     $an_float = 12;

    echo gettype($a_bool); echo “<br>”;
   echo gettype($a_str); echo “<br>”;
    echo gettype($a_str2); echo “<br>”;
    echo gettype($an_int); echo “<br>”;
    echo gettype($an_float);
?>
```

8. Проанализировать код и сделать выводы:

```
<?php
 $f = "5a";
    echo settype($f,  integer); echo “<br>”;
    echo settype($f,  string); echo “<br>”;
    echo settype($f,  float); echo “<br>”;

 $s = "A5";
    echo settype($s,  integer); echo “<br>”;
```

```
    echo settype($s,  boolean);   echo “<br>”;
    echo settype($s,  float); echo “<br>”;

 $b = true;

    echo settype($b,  string);   echo “<br>”;
    echo settype($b,  integer); echo “<br>”;
    echo settype($b,  float); echo “<br>”;

$d = 654;
    echo settype($s,  string); echo “<br>”;
    echo settype($s,  boolean); echo “<br>”;
    echo settype($s,  float); echo “<br>”;
?>
```

9. Проанализировать код и сделать выводы:

```
<?php
 $f = "5a";
    echo settype($f,  integer);
    echo settype($f,  string);
    echo settype($f,  float);

 $s = "A5";
    echo settype($s,  integer);
    echo settype($s,  boolean);
    echo settype($s,  float);

 $b = true;

    echo settype($b,  string);
    echo settype($b,  integer);
    echo settype($b,  float);

$d = 654;
    echo settype($s,  string);
    echo settype($s,  boolean);
    echo settype($s,  float);


?>
```

10. Проанализировать код и сделать выводы:

```
<?php
        $name = “id”;
        $id=55;
        echo $$name;
?>
```

ХОД РАБОТЫ 2 занятия

Написать скрипты для следующих задач.

1. Создайте PHP переменную $htmllab и поместите в нее значение «ru». Распечатайте результат.

2. Создайте константу NPP (news per page — количество новостей на страницу) и задайте ей значение 10.

3. Создайте две целочисленных переменных и распечатайте результат выполнения над ними математических операторов. Примечание: например, $num1=23 и $num2=67; результат оператора сложения echo $num1+$num2.

4. Создайте переменную $test= «345» и узнайте ее тип, при помощи функции gettype(); посмотрите в документации, какая функция устанавливает тип переменной.

5. При помощи разных способов приведения типов данных, приведите переменную $test из предыдущего задания в числовой тип.

6. Проверьте, что выведет выражение echo «103»+2.

7. Вычислить длину гипотенузы прямоугольного треугольника.

8. Используя функцию генерации случайного числа - rand, сгенерируйте целое число в диапазоне от 45 до 234.

9. Используя функцию генерации случайного числа, сгенерируйте дробное число в диапазоне от 45 до 234.

САМОСТОЯТЕЛЬНАЯ РАБОТА

1. Установить веб-сервер.

2. Изучить теоретический материал, протестировать все примеры, приведенные в нем, сделать скрины результатов скриптов. Подготовить отчет по практической работе. Титульный лист в Приложении А. На втором листе содержание работы. Начиная с третьего: 1) формулировка задания; 2) код сценария; 3) результат-скрин; 4) ответы на вопросы.

Дать ответы письменно на контрольные вопросы.

Последовательность действий при запуске скрипта:

1. Создать сценарий в текстовом редакторе сохранить с расширением .php.
2. Поместить файл в корневой каталог вашего web-сервера.
3. Набрать в адресной строке браузера: http://localhost/имя_файла_с_расширением

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Как обозначаются однострочные и многострочные комментарии?
2. Что такое переменная?
3. Как выделяется код PHP?
4. Для чего предназначена функция settype()?
5. Для чего предназначена функция gettype()?
6. Для чего предназначена функция is_float()?
7. Для чего предназначена функция is_string()?
8. Для чего предназначена функция unset()?
9. Для чего предназначена функция isset()?
10. Для чего предназначена функция is_resource()?
11. Для чего предназначена функция is_double ()?
12. Для чего предназначена функция empty ()?
13. Для чего предназначена функция $_SERVER ()?

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ № 3

Константы, операторы

Цель занятия: формирование навыка работы по созданию сценариев с применением констант и операторов.

Этапы выполнения работы:

1. Протестировать скрипты, сделать вывод.
2. Решить задачи, ответить на вопросы.
3. Подготовить работу к защите.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Константы

Константа - это неизменяемое значение. Исходя из их названия, нетрудно понять, что их значение не может изменяться в ходе выполнения скрипта (исключения представляют "волшебные" константы, которые на самом деле не являются константами в полном смысле этого слова). Имена констант чувствительны к регистру. По принятому соглашению, имена констант всегда пишутся в верхнем регистре.

Различия между константами и переменными:

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

– до PHP5.3 константы можно определить только с помощью функции define(), а не присваиванием значения;

– константы могут быть определены и доступны в любом месте без учета области видимости;

– константы не могут быть переопределены или аннулированы после первоначального объявления;

– константы могут иметь только скалярные значения, или скалярные и массивы в PHP 5.6 и новее. Вы можете использовать массивы в скалярных выражениях констант (например, *const FOO = array(1,2,3)[0];*), но результатом должно быть скалярное выражение.

Defined (“имя_константы”) - проверка существования константы.

Пример #1 Определение констант

```
<?php
   define("CONSTANT", "Здравствуй, мир.");
   echo CONSTANT; // выводит "Здравствуй, мир."
   echo Constant; // выводит "Constant" и предупреждение.
?>
```

Пример #2 Определение констант с помощью ключевого слова *const*

```
<?php
    // Работает, начиная с версии PHP 5.3.0
    const CONSTANT = 'Здравствуй, мир.';
    echo CONSTANT;

    // Работает, начиная с версии PHP 5.6.0
    const ANOTHER_CONST = CONSTANT.'; Прощай, мир.';
    echo ANOTHER_CONST;
?>
```

Пример #3 Правильные и неправильные имена констант

```
<?php
    // Правильные имена констант
    define("FOO",     "something");
    define("FOO2",    "something else");
    define("FOO_BAR", "something more");
    // Неправильные имена констант
    define("2FOO",    "something");
    // Это корректное объявление, но лучше его не использовать:
    // PHP однажды может зарегистрировать "волшебную" константу,
    // которая сломает ваш скрипт
    define("__FOO__", "something");
?>
```

Операторы

Операторы можно сгруппировать по количеству принимаемых ими значений.

Унарные операторы принимают только одно значение, например, ! (оператор логического отрицания) или ++ (инкремент).

Бинарные операторы принимают два значения; это, например, знакомые всем арифметические операторы + (плюс) и - (минус), большинство поддерживаемых в PHP операторов входят именно в эту категорию. Ну и, наконец, есть всего один тернарный оператор, *?:*, принимающий три значения, обычно его так и называют – «тернарный оператор» (хотя, возможно, более точным названием было бы «условный оператор»).

Арифметические операции		
Пример	**Название**	**Результат**
-$a	Отрицание	Смена знака *$a*.
$a + $b	Сложение	Сумма *$a* и *$b*.
$a - $b	Вычитание	Разность *$a* и *$b*.
$a * $b	Умножение	Произведение *$a* и *$b*.
$a / $b	Деление	Частное от деления *$a* на *$b*.
$a % $b	Деление по модулю	Целочисленный остаток от деления *$a* на *$b*.

```
<?php
   echo (5 % 3)."<br>";          // выводит 2
   echo (5 % -3)."<br>";         // выводит 2
   echo (-5 % 3)."<br>";         // выводит -2
   echo (-5 % -3)."<br>";        // выводит -2
?>
```

Базовый оператор присваивания обозначается как «=». На первый взгляд может показаться, что это оператор «равно». На самом деле это не так. В действительности, оператор присваивания означает, что левый операнд получает значение правого выражения, (т.е. устанавливается значением).

Результатом выполнения оператора присваивания является само присвоенное значение. Таким образом, результат выполнения «$a = 3» будет равен 3. Это позволяет делать следующее:

```
<?php
  echo $a = ($b = 4) + 5;
```

```
 // $a теперь равно 9, а $b было присвоено 4
?>
```

В дополнение к базовому оператору присваивания имеются «комбинированные операторы» для всех бинарных арифметических операций, операций объединения массивов и строковых операций, которые позволяют использовать некоторое значение в выражении, а затем установить его как результат данного выражения.

Например:

```
<?php
    $a = 3;
    $a += 5; // аналогично записи $a = $a + 5;
    $b = "Привет, ";
    echo $b .= "друг!"; // получим $b - "Привет, друг!"
?>
```

Логические операторы		
Пример	Название	Результат
$a and $b	И	TRUE если и *$a*, и *$b* TRUE
$a && $b		
$a or $b	Или	TRUE если или *$a*, или *$b* TRUE
$a || $b		
$a xor $b	Исключающее или	TRUE если *$a*, или *$b* TRUE, но не оба
!$a	Отрицание	TRUE если *$a* не TRUE

Пример #1 Объяснение логических операторов

```
<?php
    // Оператор "||" имеет больший приоритет, чем "or"

  // Результат выражения (false || true) будет присвоен
переменной $var1
  // Действие приоритета: $var1 = (false || true)
  $var1 = false || true;
  echo $var1; // => 1

  // Сначала переменной присваивается значение false, а затем
вычисляется второй операнд
  // Действие приоритета: ($var2 = false) or true
  $var2 = false or true;
  echo $var2;  // false не выводится

  // ($var3 = 0) or 3
  $var3 = 0 or 3;
  echo "<br>$var3";    // => 0
 ?>
```

В PHP есть два оператора для работы со строками (string). Первый - оператор конкатенации ('.'), который возвращает строку, представляющую собой соединение левого и правого аргумента. Второй - оператор присваивания с конкатенацией ('.='), который присоединяет правый аргумент к левому.

```
<?php
    $a = "Hello ";
    $b = $a."World!"; // $b теперь содержит строку "Hello World!"
    $a = "Hello ";
    $a .= "World!"; // $a теперь содержит строку "Hello World!"
?>
```

Операторы отношения

Пример	Название
$a==$b	проверка на равенство
$a!=$b	проверка на неравенство
$a<$b	проверка на меньшее значение
$a>$b	проверка на большее значение
$a<=$b	проверка на меньше или равно
$a>=$b	проверка на больше или равно
$a= = =$b	проверка на идентичность (требует от своих операндов не только одинаковых значений, но и совпадения типа данных)

В PHP есть возможность применить операторы отношения и к строкам.

Пример:

```
<?php
  $a=4;    //integer
  $b="44";    //string
  echo"Проверка на равенство: ";
  echo($a==$b); //выводит true
  echo"<br>";
  echo"Проверка на идентичность: ";
  echo($a===$b); //выводит FALSE
?>
```

Другие операторы

Оператор подавления ошибок @ применяется для отладки сценариев php. Если перед выражением написано @, то ошибки не будут выводиться в окне браузера. Оператор @ работает только с выражениями . Есть простое правило: если что-то возвращает значение, значит вы можете использовать перед ним оператор @. Например, вы можете использовать @ перед именем переменной, произвольной функцией или

вызовом include, константой и так далее. В то же время вы не можете использовать этот оператор перед определением функции или класса, условными конструкциями, такими как *if*, foreach и т.д.

Например:

```
<?php
  echo @$a=10/0;  // блокирует возникновение ошибки деления на 0
?>
```

Оператор увеличения и уменьшения:

1) **инкремент (++)** – увеличение на "1"
2) **декремент (--)** – уменьшение на "1"

Если оператор увеличения (++) находится слева от переменной, то выполняется сначала сложение, а потом увеличение оператора, в противном случае наоборот.

Операторы инкремента и декремента

Пример	Название	Действие
++$a	Префиксный инкремент	Увеличивает *$a* на единицу, затем возвращает значение *$a*.
$a++	Постфиксный инкремент	Возвращает значение *$a*, затем увеличивает *$a* на единицу.
--$a	Префиксный декремент	Уменьшает *$a* на единицу, затем возвращает значение *$a*.
$a--	Постфиксный декремент	Возвращает значение *$a*, затем уменьшает *$a* на единицу.

Пример:

```
$a=9;
$b=99;
$z=$a+2;        //11
$x=++$a;        //10
$y=$a++;        //10
$n=++$a+2;      //12
```

```
$a=9;
$b=99;
$y=$a++;        //9
```

ХОД РАБОТЫ

Выполнить следующие задания, создав скрипт.

1. Создать константу для администратора сайта и вывести ее значение.
2. Вычислить значение логического выражения при следующих значениях логических величин A, B и C: A= Истина, B= Ложь, C= Истина.

а) A или B; б) A и B; а) B или C.

3. Вычислить значение логического выражения при следующих значениях логических величин A, B и C: A= Истина, B= Ложь, C= Ложь.

а) A и (не B или C); б) не (A и B) или C; а) B или (C и не A).

4. Написать скрипт, сравнивая переменные различных типов.

Сравнение различных типов

Тип операнда 1	Тип операнда 2	Результат
null или string	string	NULL преобразуется в "", числовое или лексическое сравнение
bool или null	что угодно	Оба операнда преобразуются в bool, FALSE < TRUE
object	object	Встроенные классы могут определять свои собственные правила сравнения, объекты разных классов не сравниваются, объекты одного класса - сравниваются свойства тем же способом, что и в массивах (PHP 4), в PHP 5 есть свое собственное объяснение
string, resource или number	string, resource или number	Строки и ресурсы переводятся в числа, обычная математика
array	array	Массивы с меньшим числом элементов считаются меньше, если ключ из первого операнда не найден во втором операнде - массивы не могут сравниваться, иначе идет сравнение соответствующих значений (смотри пример ниже)
object	что угодно	object всегда больше
array	что угодно	array всегда больше

5. Протестировать и вывести результаты переменных.

```
<?php
    echo "<h3>Постфиксный инкремент</h3>";
    $a = 5;
    echo "Должно быть 5: " . $a++ . "<br>";
    echo "Должно быть 6: " . $a . "<br>";

    echo "<h3>Префиксный инкремент</h3>";
    $a = 5;
    echo "Должно быть 6: " . ++$a . "<br>";
    echo "Должно быть 6: " . $a . "<br>";

    echo "<h3>Постфиксный декремент</h3>";
    $a = 5;
    echo "Должно быть 5: " . $a-- . "<br>";
    echo "Должно быть 4: " . $a . "<br>";

    echo "<h3>Префиксный декремент</h3>";
    $a = 5;
    echo "Должно быть 4: " . --$a . "<br>";
    echo "Должно быть 4: " . $a . "<br>";
?>
```

САМОСТОЯТЕЛЬНАЯ РАБОТА

1. Изучить теоретический материал, протестировать все примеры, приведенные в нем, сделать скрины результатов скриптов. Подготовить отчет по практической работе. Начиная с третьего листа: 1) формулировка задания; 2) код сценария; 3) результат-скрин; 4) ответы на вопросы.

Дать ответы письменно на контрольные вопросы.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое простое условие?
2. Что такое составное условие?
3. Перечислить приоритет логических операций.

4. Что является результатом выполнения операции отношения?
5. В логическом выражении используются 3 величины логического типа. Сколько возможно вариантов сочетаний значений?

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ № 4

Управляющие конструкции

Цель занятия: формирование навыка работы по созданию сценариев с применением условных конструкций.

ТЕОРЕТИЧЕСКИЙ МАТЕРИАЛ

Любой сценарий PHP состоит из последовательности инструкций. Инструкцией может быть присваивание, вызов функции, повтор кода (цикл), сравнение, или даже инструкция, которая ничего не делает (пустой оператор). После инструкции обычно ставится точка с запятой. Кроме того, инструкции могут быть объединены в блоки заключением их в фигурные скобки. Блок инструкций также сам по себе является инструкцией.

Конструкция if

Конструкция if является одной из наиболее важных во многих языках программирования, в том числе и PHP. Она предоставляет возможность условного выполнения фрагментов кода. Структура if реализована в PHP по аналогии с языком C:

```
if (выражение)   инструкция;
```

Следующий пример выведет *a больше b*, если значение переменной *$a* больше, чем *$b*:

```
<?php
if ($a > $b)
  echo "a больше b";
?>
```

Часто необходимо, чтобы условно выполнялось более одной инструкции. Разумеется, для этого нет необходимости обворачивать каждую инструкцию в if. Вместо этого можно объединить несколько инструкций в блок. Например, следующий код выведет a больше b, если значение переменной $a больше, чем $b, и затем присвоит значение переменной $a переменной $b:

```
<?php
if ($a > $b)
{
  echo "a больше b";
  $b = $a;
}
?>
```

Инструкции if могут быть бесконечно вложены в другие инструкции if, что даёт большую гибкость в организации условного выполнения различных частей программы.

Оператор else

Часто необходимо выполнить одно выражение, если определенное условие верно, и другое выражение, если условие не верно. Именно для этого else и используется.

Else расширяет оператор if, чтобы выполнить выражение, в случае если условие в операторе if равно FALSE. К примеру, следующий код выведет *a больше чем b*, если $a больше, чем *$b*, и *a НЕ больше, чем b* в противном случае:

```
<?php
if ($a > $b)
{
  echo "a больше, чем b";
}
else
{
  echo "a НЕ больше, чем b";
}
?>
```

Конструкция elseif

Конструкция elseif, как ее имя и говорит есть сочетание if и else. Аналогично else, она расширяет оператор if для выполнения различных выражений в случае, когда условие начального оператора if эквивалентно FALSE. Однако, в отличии от else, выполнение альтернативного выражения произойдет только тогда, когда условие оператора elseif будет являться равным TRUE.

If(выражение) действие; //выражение=True
elseif (выражение 2) действие; //если выражение1=false и выражение2=true
else действие; //если выражение 1=F и выражение 2=F

К примеру, следующий код может выводить *a больше, чем b, a равно b* or a меньше, чем b:

```
<?php
if ($a > $b)
{
    echo "a больше, чем b";
} elseif ($a == $b) {
    echo "a равен b";
} else {
    echo "a меньше, чем b";
}
?>
```

Может быть несколько elseif в одном if выражении. Первое же выражение elseif (если будет хоть одно) равное TRUE будет выполнено. В PHP вы также можете написать 'else if' (в два слова), и тогда поведение будет идентичным 'elseif' (в одно слово). Синтаксически значение немного отличается (если Вы знакомы с языком C, это тоже самое поведение), но в конечном итоге оба выражения приведут к одному и тому же результату.

Выражение elseif выполнится, если предшествующее выражение if и предшествующие выражения elseif эквивалентны FALSE, а текущий elseif равен TRUE.

```
<?php

/* Некорректный способ: */
```

```
if($a > $b):
    echo $a." больше, чем ".$b;
else if($a == $b): // Не скомпилируется.
    echo "Строка выше вызывает фатальную ошибку.";
endif;


/* Корректный способ: */
if($a > $b):
    echo $a." больше, чем ".$b;
elseif($a == $b): // Заметьте, тут одно слово.
    echo $a." равно ".$b;
else:
    echo $a." не больше и не равно ".$b;
endif;

?>
```

Оператор switch

Оператор *switch* подобен серии операторов IF с одинаковым условием. Во многих случаях вам может понадобиться сравнивать одну и ту же переменную (или выражение) с множеством различных значений, и выполнять различные участки кода в зависимости от того, какое значение принимает эта переменная (или выражение). Это именно тот случай, для которого удобен оператор *switch*.

```
Switch (выражение)
 {
      case выражение: действие;
      break;
      …..
      case выражение: действие;
      break;
      default: действие;
}
```

Выражение может быть не только типа boolean но и integer, double, string.

Пример #1 Оператор *switch*

```
<?php
// первый способ
if ($i == 0)
{
    echo "i равно 0";
}
elseif ($i == 1)
{
    echo "i равно 1";
}
elseif ($i == 2)
 {
    echo "i равно 2";
 }
// второй способ
switch ($i)
```

```
 {
    case 0: echo "i равно 0"; break;
    case 1: echo "i равно 1"; break;
    case 2: echo "i равно 2"; break;
}
?>
```

Пример #2 Оператор *switch* допускает сравнение со строками

```
<?php
   $i=”груша”;
   switch ($i)
   {
    case "яблоко":  echo "i это яблоко"; break;
    case "шоколадка": echo "i это шоколадка"; break;
    case "пирог": echo "i это пирог"; break;
    default: echo “значение предмета не определено”;
   }
?>
```

Пример #3 Неочевидное поведение тернарного оператора

```
<?php
// на первый взгляд, следующий код должен вывести 'true'
echo (true?'true':false?'t':'f');

// однако, он выводит 't'
// это происходит потому, что тернарные выражения вычисляются сл
ева направо

// это намного более очевидная версия вышеприведенного кода
echo ((true ? 'true' : false) ? 't' : 'f');

// здесь вы можете видеть, что первое выражение вычисляется в 't
rue', которое
// в свою очередь вычисляется в (bool)true, таким образом возвра
щая истинную ветвь
// второго тернарного выражения.
?>
```

ХОД РАБОТЫ

1. Изучить теоретический материал, протестировать все примеры, приведенные в нем, сделать скрины результатов скриптов. Дать ответы письменно на контрольные вопросы.

2. Составить программу, которая позволит провести проверку двух чисел на равенство и выдаст большее из них.

```
<?php
If ($a>$b)
{
     echo"a>b";
}
Else If ($a==$b)
{
  echo "a=b";
    Else
```

```
    {
      echo "a<b";
    }
?>
```

3. Напишите скрипт, который будет, в зависимости от дня недели, выводить надпись. Например: сегодня среда. Примечание: используйте оператор switch.

4. Напишите скрипт, который выводить фразу об отсутствии товара в магазине или сведения о нем, если он есть в наличие. Примечание: используйте оператор switch.

САМОСТОЯТЕЛЬНАЯ РАБОТА

1. Составить программу, которая позволит провести проверку трех чисел на равенство и выдаст меньшее из них.

2. Составить сценарий, определяющий площадь выбранной фигуры: трапеция, параллелограмм и прямоугольный треугольник. Выполнить двумя способами: 1) if; 2) switch.

3. Подготовить отчет по практической работе. Титульный лист в Приложении А. На втором листе содержание работы. Начиная с третьего: 1) формулировка задания; 2) код сценария; 3) результат-скрин; 4) ответы на вопросы.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Способы записи условного оператора.
2. Назначение и варианты использования оператора выбора.
3. Синтаксис оператора выбора.

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ № 5

Управляющие конструкции

Цель занятия: формирование навыка работы по созданию сценариев с применением циклов.

ТЕОРЕТИЧЕСКИЙ МАТЕРИАЛ

Циклы while

Циклы while являются простейшим видом циклов в PHP. Они ведут себя так же, как и их коллеги из языка C. Синтаксис:

while (выражение) действие; //цикл выполняется пока выражение = true

Смысл выражения while очень прост. Оно указывает PHP выполнять вложенные выражения повторно до тех пор, пока выражение в самом while является TRUE. Значение выражения expr проверяется каждый раз перед началом цикла, поэтому даже если значение выражения изменится в процессе выполнения вложенных выражений в цикле, выполнение не прекратится до конца итерации (каждый раз, когда PHP выполняет выражения в цикле - это одна итерация). В том случае, если выражение while равно FALSE с самого начала, вложенные выражения ни разу не будут выполнены.

Также, как и с оператором if, вы можете группировать несколько выражений внутри одного цикла while, заключая эти выражения между фигурными скобками или используя альтернативный синтаксис:

Цикл do while

Цикл do-while очень похож на цикл while, с тем отличием, что истинность выражения проверяется в конце итерации, а не в начале. Главное отличие от обычного цикла while в том, что первая итерация цикла do-while гарантированно выполнится (истинность выражения проверяется в конце итерации), тогда как она может не выполниться в обычном цикле while (истинность выражения которого проверяется в начале выполнения каждой итерации, и если изначально имеет значение FALSE, то выполнение цикла будет прервано сразу).

Есть только один вариант синтаксиса цикла do…while:

do действие;
while (выражение);

```
<?php
$i = 0;
do {     echo $i;   }
 while ($i > 0);
?>
```

В примере цикл будет выполнен ровно один раз, так как после первой итерации, когда проверяется истинность выражения, она будет вычислена как FALSE ($i не больше 0) и выполнение цикла прекратится.

Опытные пользователи C могут быть знакомы с другим использованием цикла do…while, которое позволяет остановить выполнение хода программы в середине

блока, для этого нужно обернуть нужный блок кода вызовом do…while (0) и использовать break. Следующий фрагмент кода демонстрирует этот подход:

```
<?php
do
{
    if ($i < 5)
    {
        echo "i еще недостаточно велико";
        break;
    }
    $i *= $factor;
    if ($i < $minimum_limit)
    {
        break;
    }
   echo "значение i уже подходит";

    /* обработка i */

} while (0);
?>
```

Цикл for

Синтаксис цикла *for* следующий:

for(выражение1; выражение2; выражение3) действие;

Первое выражение всегда вычисляется (выполняется) только один раз в начале цикла.

В начале каждой итерации оценивается выражение*2*. Если оно принимает значение TRUE, то цикл продолжается, и вложенные операторы будут выполнены. Если оно принимает значение FALSE, выполнение цикла заканчивается.

В конце каждой итерации выражение*3* вычисляется (выполняется).

Каждое из выражений может быть пустым или содержать несколько выражений, разделенных запятыми.. Если выражение*2* отсутствует, это означает, что цикл будет выполняться бесконечно. PHP неявно воспринимает это значение как TRUE, также, как в языке C. Это может быть не столь бесполезно, так как часто необходимо прервать цикл, используя условный оператор *break* вместо использования выражения в цикле *for*, которое принимает истинное значение.

Оператор foreach

В 4 версии php появился оператор цикла foreach для работы с массивами.

ХОД РАБОТЫ

Задача 1. Составить программу, которая позволит провести проверку двух чисел на равенство и выдаст большее из них.

```
<?php
    If ($a>$b)
```

```
    {
        echo"a>b";
    }
    Else If ($a==$b)
    {
      echo "a=b";
       Else
       {
         echo "a<b";
       }
?>
```

Задача 2. Создать сценарий, выводящий числа от 1 до 10.

```
<?php
/* пример 1 */

$i = 1;
while ($i <= 10)
 {
    echo $i++;  /* выводится будет значение переменной
                   $i перед её увеличением
                   (post-increment) */
}

/* пример 2 */

$i = 1;

  do
    {
       echo $i;
       $i++;
    }
 while ($i == 10);
?>

/* пример 3 */

for ($i = 1; $i <= 10; $i++) {
    echo $i;
}

/* пример 4 */

for ($i = 1; ; $i++) {
    if ($i > 10) {
        break;
    }
    echo $i;
}
```

```
/* пример 5 */

$i = 1;
for (; ; ) {
    if ($i > 10) {
        break;
    }
    echo $i;
    $i++;
}

/* пример 6 */

for ($i = 1, $j = 0; $i <= 10; $j += $i, print $i, $i++);
?>
```

Конечно, третий пример кажется самым хорошим (или, возможно, шестой), но вы можете обнаружить, что возможность использовать пустые выражения в циклах *for* может стать удобной во многих случаях.

Задача 3. Таблица умножения

ЗАДАНИЕ 1

- Создайте две числовые переменные $cols и $rows
- Присвойте созданным переменным произвольные значения в диапазоне от 1 до 10

ЗАДАНИЕ 2

- Используя циклы отрисуйте таблицу умножения в виде HTML-таблицы на следующих условиях
- Число столбцов должно быть равно значению переменной $cols
- Число строк должно быть равно значению переменной $rows
- Ячейки на пересечении столбцов и строк должны содержать значения, являющиеся произведением порядковых номеров столбца и строки
- Рекомендуется использовать цикл for

ЗАДАНИЕ 3

- Значения в ячейках первой строки и первого столбца должны быть отрисованы полужирным шрифтом и выровнены по центру ячейки
- Фоновый цвет ячеек первой строки и первого столбца должен быть отличным от фонового цвета таблицы

Решение:

```
<?php
    $cols = 10;
    $rows = 10;
?>
    <table border="1">
<?php
    for ($tr=1; $tr<=$rows; $tr++)
    {
        echo "<tr>";
        for ($td=1; $td<=$cols; $td++)
         {
```

```
            if ($td==1 or $tr==1)
             {
echo "<th style='background-color:yellow'>", $tr * $td, "</th>";
             }
             else
             {
                echo "<td>", $tr * $td, "</td>";
             }
          }
       echo "</tr>";
    }
 ?>
 </table>
```

Безусловный оператор

Break прерывает выполнение текущей структуры *for, foreach, while, do…while* или *switch.*

Break принимает необязательный числовой аргумент, который сообщает ему выполнение какого количества вложенных структур необходимо прервать.

Continue используется внутри циклических структур для пропуска оставшейся части текущей итерации цикла и, при соблюдении условий, начала следующей итерации.

САМОСТОЯТЕЛЬНАЯ РАБОТА

1. Создать скрипт, который выводит числа от 24 до 73.
2. Найти сумму чисел от 23 до 60 включительно.
3. Напишите PHP цикл, который выводит ненумерованный список из 10 пунктов.
4. Создайте массив из 100 случайных чисел.
5. Вывести массив из предыдущего задания, при помощи цикла while.
6. Создайте массив из 10 строк и выведите их любым циклом внутри HTML-элемента div.
7. **Создайте цикл**, который выводит числа то 0 до 100 в HTML-элементах div; окраска HTML-элементов должна чередоваться («зебра»).
8. Создайте страницу, содержащую меню сайта. Выделите меню сайта в отдельный файл menu.inc.php и подключите его в основной странице.
9. *Создайте два файла shop.php и goods.php. Создайте массив с информацией о товарах. Второй файл должен содержать разметку для одного товара в магазине (например, как тут) . В цикле подключайте goods.php, одновременно заполняя его информацией из массива.
10. *Прочитайте как создаются темы (шаблоны оформления) для какой-нибудь системы управления содержимым (CMS). Например, WordPress.

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ № 6

Функции

Цель занятия: формирование навыка работы по созданию сценариев со встроенными функциями, а также созданию сценариев с пользовательской функцией.

ТЕОРЕТИЧЕСКИЙ МАТЕРИАЛ

Функции, определяемые пользователем

Приведем пример синтаксиса, используемого для описания функций:

Пример #1 Псевдокод для демонстрации использования функций

```
<?php
function foo($arg_1, $arg_2, /* ..., */ $arg_n)
{
    echo "Example function.\n";
    return $retval;
}
?>
```

Внутри функции можно использовать любой корректный PHP-код, в том числе другие функции и даже объявления классов.

Имена функций следуют тем же правилам, что и другие метки в PHP. Корректное имя функции начинается с буквы или знака подчеркивания, за которым следует любое количество букв, цифр или знаков подчеркивания.

В случае, когда функция определяется в зависимости от какого-либо условия, например, как это показано в двух приведенных ниже примерах, обработка описания функции должна предшествовать ее вызову.

Пример #2 Функции, зависящие от условий

```
<?php

$makefoo = true;

/* Мы не можем вызвать функцию foo() в этом месте,
   поскольку она еще не определена, но мы можем
   обратиться к bar() */

bar();

if ($makefoo) {
  function foo()
  {
    echo "Я не существую до тех пор, пока выполнение программы меня не достиг
нет.\n";
  }
}

/* Теперь мы благополучно можем вызывать foo(),
   поскольку $makefoo была интерпретирована как true */

if ($makefoo) foo();

function bar()
{
  echo "Я существую сразу с начала старта программы.\n";
}

?>
```

Пример #3 Вложенные функции

```php
<?php
function foo()
{
  function bar()
  {
    echo "Я не существую пока не будет вызвана foo().\n";
  }
}

/* Мы пока не можем обратиться к bar(),
   поскольку она еще не определена. */

foo();

/* Теперь мы можем вызвать функцию bar(),
   обработка foo() сделала ее доступной. */

bar();

?>
```

Обращение к функциям через переменные

PHP поддерживает концепцию переменных функций. Это означает, что если к имени переменной присоединены круглые скобки, PHP ищет функцию с тем же именем, что и результат вычисления переменной, и пытается ее выполнить. Эту возможность можно использовать для реализации обратных вызовов, таблиц функций и множества других вещей.

Переменные функции не будут работать с такими языковыми конструкциями как echo, print, unset(), isset(), empty(), include, require и другими подобными им операторами. Вам необходимо реализовывать свою функцию-обертку (wrapper) для того, чтобы приведенные выше конструкции могли работать с переменными функциями.

Пример #1 Работа с функциями посредством переменных

```php
<?php
  function foo() {
                    echo "In foo()<br />\n";
                  }

   function bar($arg = '')
                 {
                    echo "In bar(); argument was '$arg'.<br />\n";
                  }

// Функция-обертка для echo
          function echoit($string)
           {
              echo $string;
           }

          $func = 'foo';
          $func();        // Вызывает функцию foo()

          $func = 'bar';
          $func('test');  // Вызывает функцию bar()

          $func = 'echoit';
          $func('test');  // Вызывает функцию echoit()
?>
```

Вы также можете вызвать методы объекта, используя возможности PHP для работы с переменными функциями.

Пример #2 Обращение к методам класса посредством переменных

```php
<?php
class Foo
{
    function Variable()
    {
        $name = 'Bar';
        $this->$name(); // Вызываем метод Bar()
    }

    function Bar()
    {
        echo "This is Bar";
    }
}

$foo = new Foo();
$funcname = "Variable";
$foo->$funcname();  // Обращаемся к $foo->Variable()

?>
```

При вызове статических методов, вызов функции «сильнее», чем оператор доступа к статическому свойству.

Пример #3 Пример вызова переменного метода со статическим свойством

```php
<?php
class Foo
{
    static $variable = 'static property';
    static function Variable()
    {
        echo 'Method Variable called';
    }
}

echo Foo::$variable; // Это выведет 'static property'. Переменная $variable
//будет разрешена в нужной области видимости.
$variable = "Variable";
Foo::$variable();  // Это вызовет $foo-
>Variable(), прочитав $variable из этой области видимости.

?>
```

ХОД РАБОТЫ

Выполнить задания

Задание 1. Протестировать скрипты теоретической части.

Задание 2. Создать сценарий, позволяющий возводить число в степень.

Задание 3. Написать скрипт, вычисляющий факториал числа.

Задание 4. Написать сценарий с применением оператора include.

Задание 5. Написать сценарий с применением оператора require.

Задание 6. Написать сценарий, вычисляющий степень с помощью рекурсии.

САМОСТОЯТЕЛЬНАЯ РАБОТА

1. Создать функцию в PHP — getPlus10(), которая будет принимать число и распечатывать сумму этого числа и 10.
2. Изменить функцию из предыдущего задания: она должна возвращать сумму, а не сразу выводить.

3. Напишите функцию pythagoras(), которая принимает значения двух катетов прямоугольного треугольника и возвращает размер гипотенузы этого прямоугольного треугольника. Примечание: может пригодиться встроенная PHP-функция pow().

4. *Создать функцию, которая находит ипотечный платеж.

5. Создайте функцию col(), которая распечатает количество переданных аргументов. Например: col(12,6,123) должна распечатать число 3.

6. Создайте функцию, которая посчитает среднее значение всех целочисленных аргументов.

7. *Напишите функцию, которая принимает неограниченное количество числовых аргументов и строит столбчатую диаграмму. В каждом столбце указываются величины из аргументов. Примечание: это задание сформулировано по аналогии с заданием по созданию функций в JavaScript http://htmllab.ru/zadachi-po-javascript-function/

8. Напишите функцию op(), которая принимает три аргумента: $num1 и $num2 – числовые, $operator – символ, обозначающий операцию. Функция должна возвращатьрезультат выполнения оператора $operator над $num1 и $num.

9. *Создайте две функции add() и sub(), которые принимают пару аргументов и возвращают сумму и разницу соответственно. Создайте функцию op2(), которая принимает два числовых аргумента $num1 и $num2, и третий строковый вызываемый аргумент (callable).

10. Создайте функцию, которая при помощи статических переменных будет выполнять основную работу только один раз.

11. Создать сценарий, позволяющий возводить число в отрицательную степень.

12. Создать сценарий, позволяющий вычислять факториал рекурсией.

ВОПРОСЫ ДЛЯ КОНТРОЛЯ

1. В чем основное преимущество, получаемое при использовании функции?
2. Сколько значений может вернуть функция?
3. В чем разница между доступом к переменной по имени и по ссылке?
4. Что в PHP означает термин «область видимости»?
5. Что такое локальная переменная?
6. Что такое глобальная переменная?
7. Что такое статическая переменная?

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ № 7

Массивы

Цель занятия: формирование навыка работы по созданию сценариев по обработке массивов.

ТЕОРЕТИЧЕСКИЙ МАТЕРИАЛ

Массив (тип array) может быть создан языковой конструкцией array(). language construct. В качестве параметров она принимает любое количество разделенных запятыми пар key => value (ключ => значение).

```
array(
    key  => value,
    key2 => value2,
    key3 => value3,
    ...
)
```

Для массивов (array), присвоение значения именованному ключу происходит с помощью оператора "=>". Приоритет этого оператора такой же, как и у остальных операторов присваивания.

Запятая после последнего элемента массива необязательна и может быть опущена. Обычно это делается для однострочных массивов, т.е. *array(1, 2)* предпочтительней *array(1, 2,)*. Для многострочных массивов с другой стороны обычно используется завершающая запятая, так как позволяет легче добавлять новые элементы в конец массива.

Начиная с PHP 5.4 возможно использовать короткий синтаксис определения массивов, который заменяет языковую конструкцию *array()* на *[]*.

Пример #1 Простой массив

```
<?php
$array = array(
    "foo" => "bar",
    "bar" => "foo",
);

// Начиная с PHP 5.4
$array = [
    "foo" => "bar",
    "bar" => "foo",
];
?>
```

key может быть либо типа integer, либо типа string. value может быть любого типа.

Дополнительно с ключом key будут сделаны следующие преобразования:

– Строки, содержащие целое число будут преобразованы к типу integer. Например, ключ со значением *"8"* будет в действительности сохранен со значением *8*. С другой стороны, значение *"08"* не будет преобразовано, так как оно не является корректным десятичным целым.

– Числа с плавающей точкой (тип float) также будут преобразованы к типу integer, т.е. дробная часть будет отброшена. Например, ключ со значением *8.7* будет в действительности сохранен со значением *8*.

– Тип bool также преобразовываются к типу integer. Например, ключ со значением *true* будет сохранен со значением *1* и ключ со значением *false* будет сохранен со значением *0*.

– Тип null будет преобразован к пустой строке. Например, ключ со значением *null* будет в действительности сохранен со значением *""*.

– Массивы (тип array) и объекты (тип object) *не могут* использоваться в качестве ключей. При подобном использовании будет генерироваться предупреждение: *Недопустимый тип смещения (Illegal offset type)*.

Если несколько элементов в объявлении массива используют одинаковый ключ, то только последний будет использоваться, а все другие будут перезаписаны.

Пример #2 Пример преобразования типов и перезаписи элементов

```
<?php
$array = array(
    1    => "a",
    "1"  => "b",
    1.5  => "c",
    true => "d",
);
var_dump($array);
?>
```

Результат выполнения данного примера:

```
array(1) {
  [1]=>
  string(1) "d"
}
```

Так как все ключи в вышеприведенном примере преобразуются к *1*, значение будет перезаписано на каждый новый элемент и останется только последнее присвоенное значение *"d"*.

Массивы в PHP могут содержать ключи типов integer и string одновременно, так как PHP не делает различия между индексированными и ассоциативными массивами.

Пример #3 Смешанные ключи типов integer и string

```
<?php
$array = array(
    "foo" => "bar",
    "bar" => "foo",
    100   => -100,
    -100  => 100,
);
var_dump($array);
?>
```

Результат выполнения данного примера:

```
array(4) {
  ["foo"]=>
  string(3) "bar"
  ["bar"]=>
  string(3) "foo"
  [100]=>
  int(-100)
  [-100]=>
  int(100)
}
```

Параметр key является необязательным. Если он не указан, PHP будет использовать предыдущее наибольшее значение ключа типа integer, увеличенное на 1.

Пример #4 Индексированные массивы без ключа

```
<?php
$array = array("foo", "bar", "hello", "world");
var_dump($array);
?>
```

Результат выполнения данного примера:

```
array(4) {
  [0]=>
  string(3) "foo"
  [1]=>
  string(3) "bar"
  [2]=>
  string(5) "hello"
  [3]=>
  string(5) "world"
}
```

Возможно указать ключ только для некоторых элементов и пропустить для других:

Пример #5 Ключи для некоторых элементов

```
<?php
$array = array(
          "a",
          "b",
     6 => "c",
          "d",
);
var_dump($array);
?>
```

Результат выполнения данного примера:

```
array(4) {
  [0]=>
  string(1) "a"
  [1]=>
  string(1) "b"
  [6]=>
  string(1) "c"
  [7]=>
  string(1) "d"
}
```

Как вы видите последнее значение *"d"* было присвоено ключу *7*. Это произошло потому, что самое большое значение ключа целого типа перед этим было *6*.

array_change_key_case	Меняет регистр всех ключей в массиве
array_chunk	Разбивает массив на части
array_column	Return the values from a single column in the input array
array_combine	Создает новый массив, используя один массив в качестве ключей, а другой в качестве соответствующих значений
array_count_values	Подсчитывает количество всех значений массива
array_diff_assoc	Вычисляет расхождение массивов с дополнительной

	проверкой индекса
array_diff_key	Вычисляет расхождение массивов, сравнивая ключи
array_diff_uassoc	Вычисляет расхождение массивов с дополнительной проверкой индекса, осуществляемой при помощи callback-функции
array_diff_ukey	Вычисляет расхождение массивов, используя callback-функцию для сравнения ключей
array_diff	Вычислить расхождение массивов
array_fill_keys	Создает массив и заполняет его значениями, с определенными ключами
array_fill	Заполняет массив значениями
array_filter	Фильтрует элементы массива с помощью callback-функции
array_flip	Меняет местами ключи с их значениями в массиве
array_intersect_assoc	Вычисляет схождение массивов с дополнительной проверкой индекса
array_intersect_key	Вычислить пересечение массивов, сравнивая ключи
array_intersect_uassoc	Вычисляет схождение массивов с дополнительной проверкой индекса, осуществляемой при помощи callback-функции
array_intersect_ukey	Вычисляет схождение массивов, используя callback-функцию для сравнения ключей
array_intersect	Вычисляет схождение массивов
array_key_exists	Проверяет, присутствует ли в массиве указанный ключ или индекс
array_keys	Возвращает все или некоторое подмножество ключей массива
array_map	Применяет callback-функцию ко всем элементам указанных массивов
array_merge_recursive	Рекурсивное слияние двух или более массивов
array_merge	Сливает один или большее количество массивов
array_multisort	Сортирует несколько массивов или многомерные массивы
array_pad	Дополнить размер массива определенным значением до заданной величины
array_pop	Извлекает последний элемент массива
array_product	Вычислить произведение значений массива
array_push	Добавляет один или несколько элементов в конец массива
array_rand	Выбирает одно или несколько случайных значений из массива
array_reduce	Итеративно уменьшает массив к единственному значению, используя callback-функцию
array_replace_recursive	Рекурсивно заменяет элементы первого массива элементами переданных массивов
array_replace	Замена элементов массива элементами других переданных массивов
array_reverse	Возвращает массив с элементами в обратном порядке
array_search	Осуществляет поиск данного значения в массиве и возвращает соответствующий ключ в случае удачи
array_shift	Извлекает первый элемент массива

array_slice	Выбирает срез массива
array_splice	Удаляет часть массива и заменяет её чем-нибудь ещё
array_sum	Вычисляет сумму значений массива
array_udiff_assoc	Вычисляет расхождение в массивах с дополнительной проверкой индексов, используя для сравнения значений callback-функцию
array_udiff_uassoc	Вычисляет расхождение в массивах с дополнительной проверкой индексов, используя для сравнения значений и индексов callback-функцию
array_udiff	Вычисляет расхождение массивов, используя для сравнения callback-функцию
array_uintersect_assoc	Вычисляет пересечение массивов с дополнительной проверкой индексов, используя для сравнения значений callback-функцию
array_uintersect_uassoc	Вычисляет пересечение массивов с дополнительной проверкой индекса, используя для сравнения индексов и значений callback-функцию
array_uintersect	Вычисляет пересечение массивов, используя для сравнения значений callback-функцию
array_unique	Убирает повторяющиеся значения из массива
array_unshift	Добавляет один или несколько элементов в начало массива
array_values	Выбирает все значения массива
array_walk_recursive	Рекурсивно применяет пользовательскую функцию к каждому элементу массива
array_walk	Применяет заданную пользователем функцию к каждому элементу массива
array	Создает массив
arsort	Сортирует массив в обратном порядке, сохраняя ключи
asort	Сортирует массив, сохраняя ключи
compact	Создает массив, содержащий названия переменных и их значения
count	Подсчитывает количество элементов массива или что-то в объекте
current	Возвращает текущий элемент массива
each	Возвращает текущую пару ключ/значение из массива и смещает его указатель
end	Устанавливает внутренний указатель массива на его последний элемент
extract	Импортирует переменные из массива в текущую таблицу символов
in_array	Проверяет, присутствует ли в массиве значение
key_exists	Псевдоним array_key_exists
key	Выбирает ключ из массива
krsort	Сортирует массив по ключам в обратном порядке
ksort	Сортирует массив по ключам
list	Присваивает переменным из списка значения подобно массиву
natcasesort	Сортирует массив, используя алгоритм "natural order" без учета регистра символов
natsort	Сортирует массив, используя алгоритм "natural order"

next	Передвигает внутренний указатель массива на одну позицию вперёд
pos	Псевдоним current
prev	Передвигает внутренний указатель массива на одну позицию назад
range	Создает массив, содержащий диапазон элементов
reset	Устанавливает внутренний указатель массива на его первый элемент
rsort	Сортирует массив в обратном порядке
shuffle	Перемешивает массив
sizeof	Псевдоним count
sort	Сортирует массив
uasort	Сортирует массив, используя пользовательскую функцию для сравнения элементов с сохранением ключей
uksort	Сортирует массив по ключам, используя пользовательскую функцию для сравнения ключей
usort	Сортирует массив по значениям используя пользовательскую функцию для сравнения элементов

Операторы, работающие с массивами		
Пример	**Название**	**Результат**
$a + $b	Объединение	Объединение массива *$a* и массива *$b*.
$a == $b	Равно	**TRUE** в случае, если *$a* и *$b* содержат одни и те же пары ключ/значение.
$a === $b	Тождественно равно	**TRUE** в случае, если *$a* и *$b* содержат одни и те же паты ключ/значение в том же самом порядке и того же типа.
$a != $b	Не равно	**TRUE**, если массив *$a* не равен массиву *$b*.
$a <> $b	Не равно	**TRUE**, если массив *$a* не равен массиву *$b*.
$a !== $b	Тождественно не равно	**TRUE**, если массив *$a* не равен тождественно массиву *$b*.

Оператор + возвращает левый массив, к которому был присоединен правый массив. Для ключей, которые существуют в обоих массивах, будут использованы значения из левого массива, а соответствующие им элементы из правого массива будут проигнорированы.

```
<?php
$a = array("a" => "apple", "b" => "banana");
$b = array("a" => "pear", "b" => "strawberry", "c" => "cherry");

$c = $a + $b; // Объединение $a и $b
echo "Объединение of \$a and \$b: \n";
var_dump($c);

$c = $b + $a; // Объединение $b и $a
echo "Объединение of \$b and \$a: \n";
var_dump($c);
?>
```

При сравнении элементы массива считаются идентичными, если совпадает и ключ, и соответствующее ему значение.

Пример #1 Сравнение массивов

```
<?php
$a = array("apple", "banana");
$b = array(1 => "banana", "0" => "apple");

var_dump($a == $b); // bool(true)
var_dump($a === $b); // bool(false)
```

Перебор массивов как показано ниже - это обычное дело для многих пользователей.

```
<?php
/*
 * Это массив с некоторыми данными, которые мы хотим изменить
 * при работе цикла.
 */
$people = array(
    array('name' => 'Kalle', 'salt' => 856412),
    array('name' => 'Pierre', 'salt' => 215863)
);

for($i = 0; $i < count($people); ++$i) {
    $people[$i]['salt'] = mt_rand(000000, 999999);
}
?>
```

Вышеприведенный код может работать медленно, так как размер массива вычисляется в каждой итерации. Поскольку размер не меняется, цикл может быть легко оптимизирован с помощью промежуточной переменной, в которую будет записан размер массива, вместо повторяющихся вызовов функции count():

```
<?php
$people = array(
    array('name' => 'Kalle', 'salt' => 856412),
    array('name' => 'Pierre', 'salt' => 215863)
);

for($i = 0, $size = count($people); $i < $size; ++$i) {
    $people[$i]['salt'] = mt_rand(000000, 999999);
}
?>
```

Конструкция foreach

Конструкция *foreach* предоставляет простой способ перебора массивов. *Foreach* работает только с массивами и объектами, и будет генерировать ошибку при попытке использования с переменными других типов или неинициализированными переменными. Существует два вида синтаксиса:

```
foreach (array_expression as $value)
    statement
foreach (array_expression as $key => $value)
    statement
```

Пример 1: только значение

```
$a = array(1, 2, 3, 17);

foreach ($a as $v) {
    echo "Текущее значение переменной \$a: $v.\n";
}
```

Пример 2: значение (для иллюстрации массив выводится в виде значения с ключом)

```
$a = array(1, 2, 3, 17);

$i = 0; /* только для пояснения */

foreach ($a as $v) {
    echo "\$a[$i] => $v.\n";
    $i++;
}
```

Пример 3: ключ и значение

```
$a = array(
    "one" => 1,
    "two" => 2,
    "three" => 3,
    "seventeen" => 17
);

foreach ($a as $k => $v) {
    echo "\$a[$k] => $v.\n";
}
```

Пример 4: многомерные массивы

```
$a = array();
$a[0][0] = "a";
$a[0][1] = "b";
$a[1][0] = "y";
$a[1][1] = "z";

foreach ($a as $v1) {
    foreach ($v1 as $v2) {
        echo "$v2\n";
    }
}
```

Пример 5: динамические массивы

```
foreach (array(1, 2, 3, 4, 5) as $v) {
    echo "$v\n";
}
?>`
```

break прерывает выполнение текущей структуры for, foreach, while, do-while или switch.

break принимает необязательный числовой аргумент, который сообщает ему выполнение какого количества вложенных структур необходимо прервать.

```
<?php
$arr = array('один', 'два', 'три', 'четыре', 'стоп', 'пять');
while (list(, $val) = each($arr)) {
    if ($val == 'стоп') {
        break;    /* Тут можно было написать 'break 1;'. */
    }
    echo "$val<br />\n";
}

/* Использование дополнительного аргумента. */

$i = 0;
while (++$i) {
    switch ($i) {
    case 5:
        echo "Итерация 5<br />\n";
        break 1;  /* Выйти только из конструкции switch. */
    case 10:
        echo "Итерация 10; выходим<br />\n";
        break 2;  /* Выходим из конструкции switch и из цикла wh
ile. */
    default:
        break;
    }
}
?>
```

ХОД РАБОТЫ

Выполнить задания

1. Протестировать скрипты теоретической части.
2. Создать сценарий на сложение двух массивов с одинаковым числом элементов.
3. Создайте массив из 100 случайных чисел.
4. Вывести массив из предыдущего задания, при помощи цикла while, а потом при помощи foreach.
5. Создайте массив из 10 строк и выведите их любым циклом внутри HTML-элемента div.
6. * Создайте массив**,** каждый элемент которого тоже массив с ключами title, description, price. Выведите все элементы этого массива, так, чтобы заголовки были в HTML-элементе h2, описания в p, а цена в гиперсылке.
7. * При выводе элементов из предыдущего задания покрасьте фон элементов ниже определенной цены в отличный от других цвет.
8. *Создать цикл, которые выводит 20 фрагментов разметки Bootstrap — http://getbootstrap.com/components/#thumbnails-custom-content
9. *Создать массив с названиями и адресами ссылок. Вывести этот массив в виде выпадающего меню Bootstrap http://getbootstrap.com/components/#dropdowns
10. *Вывести календарь на текущий месяц. Разметку взять тут http://htmllab.ru/calendar-html/

11. *Взять текст с Яндекс.Рефератов. Задать ключевое слово и записать в массив все расположения этого ключевого слова.

12. *Разметить все найденные слова из предыдущего задания HTML-элементом mark.

САМОСТОЯТЕЛЬНАЯ РАБОТА

1. Создать сценарий на сложение двух массивов с разным числом элементов.
2. Создать сценарий на разворот массива.
3. Создать сценарий на удаление элементов массива: первого, последнего, с определенным номером.
4. Переверните массив $arr = array(“first”=>45, “second”=>76, “third”=>12) при помощи встроенной в PHP функции по работе с массивами.
5. Создайте массив из 50 случайных чисел от 0 до 100. Найти все числа меньшие 72 и поместить их в отдельный массив.
6. В предыдущем номере вывести полученный массив в порядке возрастания значений.
7. Есть массив $arr = array(“first”=>45, “second”=>76, “third”=>12). Используя встроенную в PHP функцию, получить массив, элементами которого являются **значениями** массива $arr.
8. Есть массив $arr = array(“first”=>45, “second”=>76, “third”=>12). Используя встроенную в PHP функцию, получить массив, элементами которого являются **ключами** массива $arr
9. Используя встроенные функции, удалите первый элемент массива $arr = [45, “тест”, 100] и добавьте в конец массива строку “тест2”. Примечание: добавление элемента в конец массива также нужно выполнить функцией.
10. Есть массив $arr = array(45, 76, 12, 12, 45, 12). Сколько элементов будет в массиве, который вернет функция **array_unique().**

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ № 8

Строки

Цель занятия: формирование навыка работы по созданию сценариев, обрабатывающего строки

ТЕОРЕТИЧЕСКИЙ МАТЕРИАЛ

Строка - это набор символов, где символ - это то же самое, что и байт. Это значит, что PHP поддерживает ровно 256 различных символов, а также то, что в PHP нет встроенной поддержки Unicode.

Простейший способ определить строку - это заключить ее в одинарные кавычки (символ ').

Чтобы использовать одинарную кавычку внутри строки, проэкранируйте ее обратной косой чертой (\). Если необходимо написать саму обратную косую черту, продублируйте ее (\\). Все остальные случаи применения обратной косой черты 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.

```
<?php
echo 'это простая строка';

echo 'Также вы можете вставлять в строки
символ новой строки вот так,
это нормально';

// Выводит: Однажды Арнольд сказал: "I'll be back"
echo 'Однажды Арнольд сказал: "I\'ll be back"';

// Выводит: Вы удалили C:\*.*?
echo 'Вы удалили C:\\*.*?';

// Выводит: Вы удалили C:\*.*?
echo 'Вы удалили C:\*.*?';

// Выводит: Это не будет развернуто: \n новая строка
echo 'Это не будет развернуто: \n новая строка';

// Выводит: Переменные $expand также $either не разворачиваются
echo 'Переменные $expand также $either не разворачиваются';
?>
```

Управляющие последовательности	
Последовательность	**Значение**
\n	новая строка (LF или 0x0A (10) в ASCII)
\r	возврат каретки (CR или 0x0D (13) в ASCII)

Управляющие последовательности	
Последовательность	**Значение**
\t	горизонтальная табуляция (HT или 0x09 (9) в ASCII)
\v	вертикальная табуляция (VT или 0x0B (11) в ASCII) (с версии PHP 5.2.5)
\e	escape-знак (ESC или 0x1B (27) в ASCII) (с версии PHP 5.4.4)
\f	подача страницы (FF или 0x0C (12) в ASCII) (с версии PHP 5.2.5)
\\	обратная косая черта
\$	знак доллара
\"	двойная кавычка

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

addcslashes	Экранирует строку слешами в стиле языка C
addslashes	Экранирует строку с помощью слешей
bin2hex	Преобразует бинарные данные в шестнадцатеричное представление
chop	Псевдоним rtrim
chr	Возвращает символ по его коду
chunk_split	Разбивает строку на фрагменты
convert_cyr_string	Преобразует строку из одной кириллической кодировки в другую
convert_uudecode	Декодирует строку из формата uuencode в обычный вид
convert_uuencode	Кодирует строку в формат uuencode
count_chars	Возвращает информацию о символах, входящих в строку
crc32	Вычисляет полином CRC32 для строки
crypt	Необратимое хэширование строки
echo	Выводит одну или более строк
explode	Разбивает строку с помощью разделителя
fprintf	Записывает отформатированную строку в поток
get_html_translation_table	Возвращает таблицу преобразований, используемую функциями htmlspecialchars и htmlentities
hebrev	Преобразует текст на иврите из логической кодировки в визуальную
hebrevc	Преобразует текст на иврите из логической кодировки в визуальную с преобразованием перевода строки
hex2bin	Преобразует шестнадцатеричные данные в двоичные
html_entity_decode	Преобразует все HTML-сущности в соответствующие

	символы
htmlentities	Преобразует все возможные символы в соответствующие HTML-сущности
htmlspecialchars_decode	Преобразует специальные HTML-сущности обратно в соответствующие символы
htmlspecialchars	Преобразует специальные символы в HTML-сущности
implode	Объединяет элементы массива в строку
join	Псевдоним implode
lcfirst	Преобразует первый символ строки в нижний регистр
levenshtein	Вычисляет расстояние Левенштейна между двумя строками
localeconv	Возвращает информацию о числовых форматах
ltrim	Удаляет пробелы (или другие символы) из начала строки
md5_file	Возвращает MD5-хэш файла
md5	Возвращает MD5-хэш строки
metaphone	Возвращает ключ metaphone для строки
money_format	Форматирует число как денежную величину
nl_langinfo	Возвращает информацию о языке и локали
nl2br	Вставляет HTML-код разрыва строки перед каждым переводом строки
number_format	Форматирует число с разделением групп
ord	Возвращает ASCII-код символа
parse_str	Разбирает строку в переменные
print	Выводит строку
printf	Выводит отформатированную строку
quoted_printable_decode	Преобразует строку, закодированную методом quoted-printable в 8-битовую строку
quoted_printable_encode	Кодирует 8-битную строку в с помощью метода quoted-printable
quotemeta	Экранирует специальные символы
rtrim	Удаляет пробелы (или другие символы) из конца строки
setlocale	Устанавливает настройки локали
sha1_file	Возвращает SHA1-хэш файла
sha1	Возвращает SHA1-хэш строки
similar_text	Вычисляет степень похожести двух строк
soundex	Возвращает ключ soundex для строки
sprintf	Возвращает отформатированную строку
sscanf	Разбирает строку в соответствии с заданным форматом
str_getcsv	Выполняет разбор CSV-строки в массив
str_ireplace	Регистронезависимый вариант функции str_replace
str_pad	Дополняет строку другой строкой до заданной длины
str_repeat	Возвращает повторяющуюся строку
str_replace	Заменяет все вхождения строки поиска на строку замены
str_rot13	Выполняет преобразование ROT13 над строкой
str_shuffle	Переставляет символы в строке случайным образом
str_split	Преобразует строку в массив
str_word_count	Возвращает информацию о словах, входящих в строку
strcasecmp	Бинарно-безопасное сравнение строк без учета регистра
strchr	Псевдоним strstr
strcmp	Бинарно-безопасное сравнение строк

strcoll	Сравнение строк с учетом текущей локали
strcspn	Возвращает длину участка в начале строки, не соответствующего маске
strip_tags	Удаляет HTML и PHP-теги из строки
stripcslashes	Удаляет экранирование символов, произведенное функцией addcslashes
stripos	Возвращает позицию первого вхождения подстроки без учета регистра
stripslashes	Удаляет экранирование символов
stristr	Регистронезависимый вариант функции strstr
strlen	Возвращает длину строки
strnatcasecmp	Сравнение строк без учета регистра с использованием алгоритма "natural order"
strnatcmp	Сравнение строк с использованием алгоритма "natural order"
strncasecmp	Бинарно-безопасное сравнение первых n символов строк без учета регистра
strncmp	Бинарно-безопасное сравнение первых n символов строк
strpbrk	Ищет в строке любой символ из заданного набора
strpos	Возвращает позицию первого вхождения подстроки
strrchr	Находит последнее вхождение символа в строке
strrev	Переворачивает строку задом наперед
strripos	Возвращает позицию последнего вхождения подстроки без учета регистра
strrpos	Возвращает позицию последнего вхождения подстроки в строке
strspn	Возвращает длину участка в начале строки, полностью соответствующего маске
strstr	Находит первое вхождение подстроки
strtok	Разбивает строку на токены
strtolower	Преобразует строку в нижний регистр
strtoupper	Преобразует строку в верхний регистр
strtr	Преобразует заданные символы или заменяет подстроки
substr_compare	Бинарно-безопасное сравнение 2 строк со смещением, с учетом или без учета регистра
substr_count	Возвращает число вхождений подстроки
substr_replace	Заменяет часть строки
substr	Возвращает подстроку
trim	Удаляет пробелы (или другие символы) из начала и конца строки
ucfirst	Преобразует первый символ строки в верхний регистр
ucwords	Преобразует в верхний регистр первый символ каждого слова в строке
vfprintf	Записывает отформатированную строку в поток
vprintf	Выводит отформатированную строку
vsprintf	Возвращает отформатированную строку
wordwrap	Переносит строку по указанному количеству символов

ХОД РАБОТЫ

Выполнить задания
Задание 1. Протестировать скрипты теоретической части.
Задание 2. Создать сценарий на вычисление длины строки.
Задание 3. Создать сценарий на нахождение подстроки в строке.
Задание 4. Создать сценарий на нахождение позиции последнего вхождения подстроки в строке.
Задание 5. Создать сценарий на преобразование строки в верхний регистр.

САМОСТОЯТЕЛЬНАЯ РАБОТА

1. Создать 5 сценариев на применение пяти любых команд из таблицы.
2. Создайте массив. Объедините все элементы массива в строку и распечатайте её.
3. Возьмите любой из рефератов на сервисе Яндекс.Весна. Выберите любое наиболее часто встречающееся слово. Используя функцию замены подстроки, поменяйте все вхождения отдельного слова, на это же слово, заключённое в HTML-элемент mark.
4. Используя функцию удаления HTML и PHP-тегов из строки, выведите на экран строку «<h1>Привет, мир!</h1>». Примечание: строка не должна выглядеть как заголовок первого уровня – все теги должны быть удалены.
5. Используя stripos(), найдите во фразе «Ехал Грека через реку» ближайшее вхождение «ре». Работа ведется с однобайтной кодировкой.
6. Найдите длину строки «Ехал Грека через реку» в однобайтной кодировке
7. Найдите длину строки «Ехал Грека через реку» в многобайтной кодировке. Примечание: вам пригодится mb_strlen().
8. Используя встроенную функцию PHP по работе со строками, найдите количество вхождений «ре» в «Ехал Грека через реку»

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ № 9

Работа с html-формами

Цель занятия: формирование навыка работы по созданию страницы с элементами формы при изучении языка веб-программирования PHP.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

HTML-форма - это средство языка HTML для ввода и передачи данных. Хотя они напрямую не относятся к PHP, для лучшего понимания материала этой главы рассмотрим основные принципы синтаксиса их создания. Любая форма начинается с тега ‹form› и заканчивается ‹/form›. Внутри открывающего тега могут указываться атрибуты, которые определяют параметры HTML-формы. Каждый из них имеет название и определенное значение. Нас будут интересовать атрибуты action и method. Внутри HTML-формы (между тегами ‹form› и ‹/form›) обычно помещают поля для ввода текста с клавиатуры, списки, кнопки и т.д. Их подробное описание можно найти в любой литературе, посвященной языку HTML.

```
‹html›
  ‹head›
        ‹title› HTML-форма ‹/title›
  ‹/head›
 ‹body›
   ‹form action="test.php"›
      ‹input type="Text" name="text"›
     ‹input type="Submit" value="Go!"›
   ‹/form›
 ‹/body›
‹/html›
```

В этом примере создается HTML-форма, которая содержит поле для ввода текста и кнопку для отправки данных. Если пользователь нажмет на кнопку из этой HTML-формы, то запустится файл с названием test.php, куда определенным способом передадутся данные из HTML-формы. Обратите внимание, что файл test.php и файл, содержащий HTML-форму, должны находиться в одной папке.

В HTML 4.01 определены следующие типы управляющих элементов:

Кнопки	задаются с помощью элементов BUTTON и INPUT. Различают:
кнопки отправки	при нажатии на них, они осуществляют отправку формы серверу;
кнопки сброса	при нажатии на них, управляющие элементы принимают первоначальные значения;
прочие кнопки	кнопки, для которых не указано действие, выполняемое по умолчанию при нажатии на них.
Зависимые переключатели (переключатели с зависимой фиксацией)	задаются элементом INPUT и представляют собой переключатели "вкл/выкл". Если несколько зависимых переключателей имеют одинаковые имена, то они являются взаимоисключающими. Это значит, что если одна из них ставится в положение "вкл", то все остальные автоматически - в положение "выкл". Именно это и является преимуществом их использования.

Независимые переключатели (переключатели с независимой фиксацией)	задаются элементом INPUT и представляют собой переключатели "вкл/выкл", но в отличие от зависимых, независимые переключатели могут принимать и изменять свое значение независимо от остальных переключателей. Даже если последние имеют такое же имя.
Меню	реализуется с помощью элементов SELECT, OPTGROUP и OPTION. Меню предоставляют пользователю список возможных вариантов выбора.
Ввод текста	реализуется элементами INPUT, если вводится одна строка, и элементами TEXTAREA - если несколько строк. В обоих случаях введенный текст становится текущим значением управляющего элемента.
Выбор файлов	позволяет вместе с формой отправлять выбранные файлы, реализуется HTML-элементом INPUT.
Скрытые управляющие элементы	создаются управляющим элементом INPUT.

ХОД РАБОТЫ

Задание 1

Создать веб-страницу с применением HTML-форм. Разместите на странице поле ввода, выпадающий список, список с видимыми 4 элементами, зависимые и независимые переключатели, поле для ввода большого текста, кнопку.

Задание 2

Создадим простую форму с одним checkbox:

```
<form action="checkbox-form.php" method="post">

    Do you need wheelchair access?

    <input type="checkbox" name="formWheelchair" value="Yes" />

    <input type="submit" name="formSubmit" value="Submit" />

</form>
```

В PHP скрипте (*checkbox-form.php*) мы можем получить выбранный вариант из массива $_POST. Если $_POST['formWheelchair'] имеет значение "*Yes*", то флажок для варианта установлен. Если флажок не был установлен, $_POST['formWheelchair'] не будет задан.

Вот пример обработки формы в PHP:

```
<?phpif(isset($_POST['formWheelchair']) &&
$_POST['formWheelchair'] == 'Yes')

{ echo "Need wheelchair access.";}
```

```
else

{  echo "Do not Need wheelchair access.";}

?>
```

Для $_POST[‘formSubmit’] было установлено значение “*Yes*”, так как это значение задано в атрибуте чекбокса value:

```
<input type="checkbox" name="formWheelchair" value="Yes" />
```

Вместо “*Yes*” вы можете установить значение "*1*" или "*on*". Убедитесь, что код проверки в скрипте PHP также обновлен.

Задание 3

3.1 Создайте переменную $name и присвойте ей значение, содержащее Ваше имя, например "Владимир" (обязательно в кавычках).

3.2 Создайте переменную $age и присвойте ей значение, содержащее Ваш возраст, например 19.

3.3 Выведите с помощью echo (или print) фразу "Меня зовут: ваше_имя".

3.4 Выведите фразу "Мне ваш_возраст лет", например: "Мне 25 лет".

3.5 Удалите переменную $age.

Решение:

```
<?php
   $name = "Владимир";
   $age = 19;
?>


<html>
   <head>
     <title>Переменные и вывод</title>
   </head>

<body>
    <h1>Переменные и вывод</h1>
  <?php
    echo "Меня зовут: $name", "<br />";
    echo "Мне $age лет";
    unset($age);
  ?>
 </body>
</html>
```

Задание 4

4.1 Создать сайт, содержащий 4 страницы.

Страница 1 – Каталог интернет-магазина

Интернет магазин

Компьютерная техника2045

Каталог **Заказать продукцию**

Периферийная техника

Беспроводные мышки

Клавиатуры

Мыши

Звуковая техника

Колонки

Наушники

Страница 2 – Выбор товаров

Товар

Мышь
Клавиатура
Наушники
Монитор

Кол-во: 1

Производитель

Китай

Заказать!

Страница 3 – Заказ

Ваш заказ

Ваш товар: Наушники
Количество покупок: 3
Производитель: Германия

Подтвердить заказ

Заказать продукцию

Страница 4 – Подтверждение заказа

Здравствуйте.
Ваш заказ подтвержден!
Заказ записан в файл test.txt

4.2 Этапы выполнения работы

1 Создать файл page1.php

```
<html>
<body>
    <h1 align="center" color="red"> Интернет магазин </h1>
    <h2 align="center">Компьютерная техника2045<h1>
    <br>
    Каталог <a href = "page2.php">Заказать продукцию</a>
    <h4>Периферийная техника<p>
        <h5>Беспроводные мышки<p>
        <h5>Клавиатуры<p>
        <h5>Мыши<p>
     <h4> Звуковая техника<p>
        <h5>Колонки<p>
        <h5>Наушники<p>
</body>
</html>
```

2 Создать файл page2.php

```
<html>
<body>
      <form action="page3.php" type="multipart/form-data" >
   <H2> Товар <br>

  <!--Список с видимыми 4 элементами-->

      <select name="tovar" size="4" >
     <option value="Мышь ">Мышь</option>
     <option value="Клавиатура ">Клавиатура</option>
     <option value="Наушники ">Наушники</option>
     <option value="Монитор ">Монитор</option>
     <option value="Колонки ">Колонки</option>
      </select><br/>
   Кол-во:
  <!--Список с видимым 1 элементом-->

      <select name="kolvo" size = "1" >
     <option value="1 ">1</option>
     <option value="2 ">2</option>
     <option value="3 ">3</option>
     <option value="4 ">4</option>
     <option value="5 ">5+</option>
      </select><br>
   Производитель<br>
      <select name="proiz" size"1" >
     <option value="Китай ">Китай</option>
     <option value="Россия ">Россия</option>
     <option value="Германия ">Германия</option>
     <option value="Америка ">Америка</option>
     <option value="Дания ">Дания</option>
```

```
      </select><br>

     <input type="Submit" value="Заказать!">
</form>
<br>

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

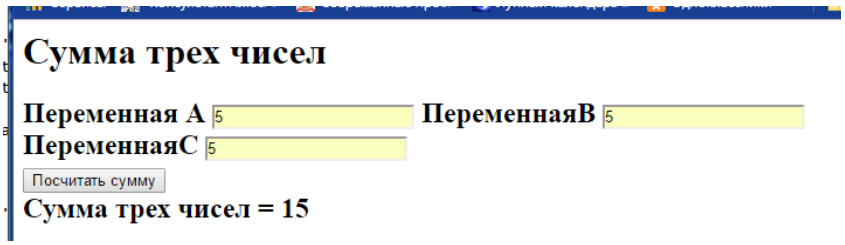
3 Создать файл page3.php

```
<!-- Запись информации в файл-->

<html>
<body>
      <h1> Ваш заказ </h1>

         <?php
             echo "  <B>Ваш товар:</B> ";
             @$tovar=$_GET['tovar'];
             echo $tovar."<br>";
             echo "<B>Количество покупок:</B> ";
             @$kolvo=$_GET['kolvo'];
             echo $kolvo."<br>";
             echo '<B>Производитель: </B> ';
             @$proiz=$_GET['proiz'];
             echo $proiz."<br>";
         ?>
        <form type="multipart/form-data" action="page4.php">
        <input type="Submit" value="Подтвердить заказ">
           <?php
               $fp=fopen("test.txt","a+");
               $contents = $tovar;
               $qwe = $kolvo;
               $asd = $proiz;
               $k=chr(13);
               fwrite($fp,$contents);
               fwrite($fp,$qwe);
               fwrite($fp,$asd);
               fwrite($fp,$k);
               fclose($fp);
            ?>

         </form>
         <a href = "page2.php">Заказать продукцию</a>
</body>
</html>

4 Создать файл page4.php
<?php
          echo '<B>Здравствуйте.</B><br> Ваш заказ
подтвержден!<br>';
```

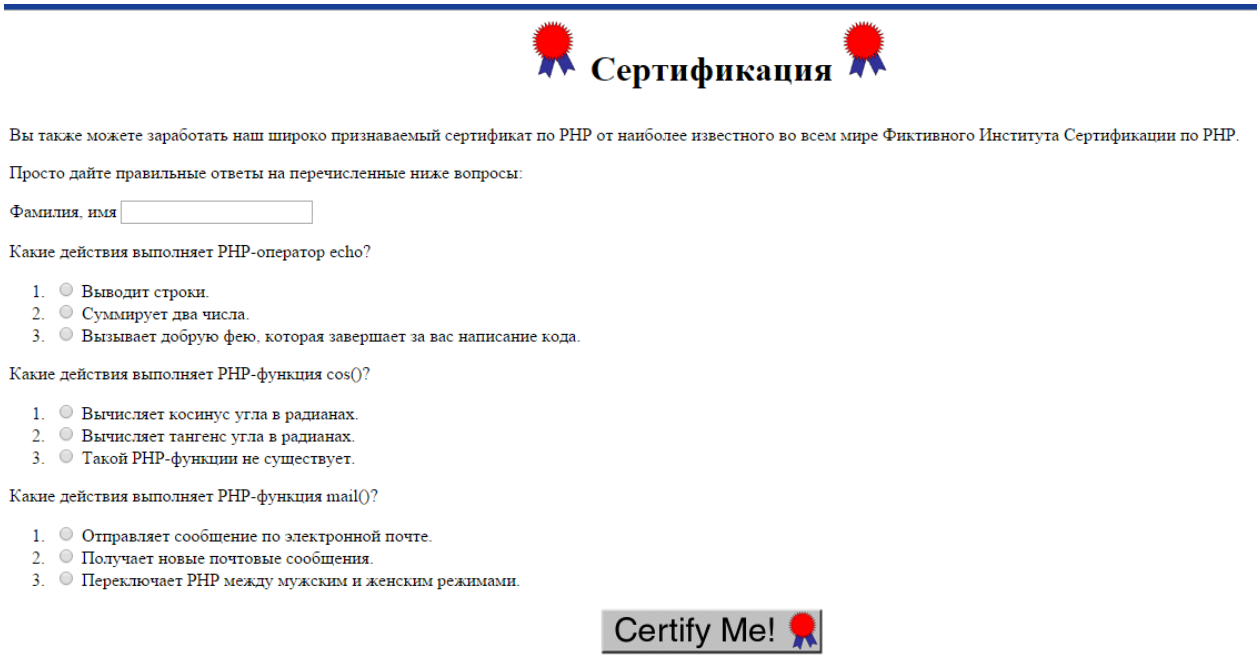
```
        echo "Заказ записан в файл test.txt";
?>
```

САМОСТОЯТЕЛЬНАЯ РАБОТА

1. Написать скрипт, вычисляющий сумму трех чисел. Интерфейс результата представлен ниже:



2. Написать скрипт, вычисляющий тестирование. Интерфейс результата представлен ниже:



3. **Создайте форму** из двух полей для ввода логина и пароля на сайте. Получите данные из формы, отфильтруйте их и распечатайте на экране. Примечание: форма должна быть отправлена методом

4. *Создайте форму с **одним полем ввода**. В PHP создайте массив с названиями городов. Примите данные формы и пройдите по всем элементам массива: если элементы массива содержат введенный фрагмент, они должны быть распечатаны на экране. Примечание: форма должна быть отправлена методом

5. Создайте **форму для нахождения** ипотечного аннуитетного платежа.

6. Создайте форму с двумя полями: логина и пароля. При введении логина «john» и пароля «qwerty» **методом POST**, показывать секретную часть страницы, иначе говорить, что данные введены некорректно.

7. * Создайте форму с **многострочным полем ввода**. Отправляя форму **методом POST** найдите часто встречаемости слов из форму и выведите их в порядке убывания

частоты встречаемости слов. Примечание: могут пригодится функции – разбиения строки по символам и нахождения встречаемости элементов в массиве.

8. Создайте массив имен (например, Вася, Коля, Даша и т.д.). Создайте форму с полем ввода, которая позволяет вводить текст с шаблоном **@name@** и обрабатывая этот текст заменять шаблон на произвольное имя из массива.

9. *Создайте форму с многострочным полем ввода. Подключите к этому полю **WYSIWYG-редактор**. Принимая данные формы, очистите все теги, кроме h1-h6, p, section и распечатайте полученный фрагмент.

10. Создайте форму со всеми возможными элементами управления, присвоим им различные имена. Выведите на экран результат отправки формы **методом GET**. Примечание: все параметры должны быть распечатаны.

11. *Создайте форму, атрибут action которой, должен содержать строку «?param=2» и методом отправки POST. Распечатайте **содержимое массивов GET и POST**-данных из формы.

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ № 10

Работа с графикой

Цель занятия: формирование навыка работы по созданию сценариев с изображением и обработкой текста на нем

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

1. Создание рисунка (рисунок хранится в растровом формате) из файла, например,

```
$image = ImageCreateFromPng("test_img.png");
```

и/или средствами PHP.

2. Указание типа содержимого для браузера (image/png, image/gif или image/jpeg)

```
Header("Content-type:image/png");
```

3. Вывод готового изображения.
   Например, передача рисунка в браузер

```
ImagePNG($image);
```

4. Освобождение памяти, выделенной для рисунка

```
ImageDestroy($image);
```

Пример создания изображения с текстом поверх существующего png-изображения кнопки

```
<?php
     header("Content-type: image/png");
     $string = $_GET['text'];
     $im     = imagecreatefrompng("images/button.png");
     $orange = imagecolorallocate($im, 220, 210, 60);
     $px     = (imagesx($im) - 7.5 * strlen($string)) / 2;
     imagestring($im, 3, $px, 9, $string, $orange);
     imagepng($im);
     imagedestroy($im);
?>
```

Использование этого сценария на веб-странице:

```
<img src="button.php?text=text">
```

Веб-мастерам часто бывает необходимо динамически создавать и/или изменять рисунки на своих страницах. Это требуется в тех случаях, когда изображения несут не исключительно декоративную функцию, а содержат некую полезную информацию. И если веб-мастер знаком с основами PHP, решение этой задачи становится более чем тривиальным. Для этого достаточно подключить модуль расширения GD.

Загрузить модуль GD можно по адресу www.boutell.com/gd. Для его подключения необходимо убрать знак комментария в строке extension=php_gd.dll (для сервера с ОС Windows; в случае Unix-систем расширение файла может быть другим) в php.ini и

перезапустить веб-сервер. Различные версии GD могут работать с разными форматами графических файлов. Так, при использовании библиотеки версии 1.6 и ниже можно создавать изображения в форматах JPEG, GIF и SWF, но не PNG. Более новые версии позволяют использовать PNG, но отказываются поддерживать формат GIF из лицензионных соображений.

gd_info	Вывод информации о текущей установленной GD библиотеке
getimagesize	Получение размера изображения
getimagesizefromstring	Получение размера изображения из строки данных
image_type_to_extension	Получение расширения файла для типа изображения
image_type_to_mime_type	Получение Mime-типа для типа изображения, возвращаемого функциями getimagesize, exif_read_data, exif_thumbnail, exif_imagetype
image2wbmp	Выводит изображение в браузер или пишет в файл
imageaffine	Return an image containing the affine transformed src image, using an optional clipping area
imageaffinematrixconcat	Concatenate two affine transformation matrices
imageaffinematrixget	Get an affine transformation matrix
imagealphablending	Задание режима сопряжения цветов для изображения
imageantialias	Требуется ли применять функции сглаживания или нет
imagearc	Рисование дуги
imagebmp	Output a BMP image to browser or file
imagechar	Рисование символа по горизонтали
imagecharup	Рисование символа вертикально
imagecolorallocate	Создание цвета для изображения
imagecolorallocatealpha	Создание цвета для изображения
imagecolorat	Получение индекса цвета пиксела
imagecolorclosest	Получение индекса цвета ближайшего к заданному
imagecolorclosestalpha	Получение индекса цвета ближайшего к заданному с учетом прозрачности
imagecolorclosesthwb	Получение индекса цвета, имеющего заданный тон, белизну и затемнение
imagecolordeallocate	Разрыв ассоциации переменной с цветом для заданного изображения
imagecolorexact	Получение индекса заданного цвета
imagecolorexactalpha	Получение индекса заданного цвета и альфа компонента
imagecolormatch	Делает цвета палитровой версии изображения более соответствующими truecolor версии
imagecolorresolve	Получает идентификатор конкретного цвета или его ближайший аналог
imagecolorresolvealpha	Получает идентификатор конкретного цвета и альфа компонента или его ближайший аналог
imagecolorset	Установка набора цветов для заданного индекса палитры
imagecolorsforindex	Получение цветов, соответствующих индексу
imagecolorstotal	Определение количества цветов в палитре изображения
imagecolortransparent	Определяет цвет как прозрачный
imageconvolution	Наложение искривляющей матрицы 3x3, используя коэффициент и смещение
imagecopy	Копирование части изображения
imagecopymerge	Копирует часть изображения с наложением

imagecopymergegray	Копирует часть изображения с наложением в градациях серого
imagecopyresampled	Копирование и изменение размера изображения с ресемплированием
imagecopyresized	Копирование и изменение размера части изображения
imagecreate	Создание нового палитрового изображения
imagecreatefrombmp	Создает новое изображение из файла или URL
imagecreatefromgd2	Создание нового изображения на основе GD2 или URL
imagecreatefromgd2part	Создание нового изображения на основе части GD2 файла или URL
imagecreatefromgd	Создание нового изображения на основе GD файла или URL
imagecreatefromgif	Создает новое изображение из файла или URL
imagecreatefromjpeg	Создает новое изображение из файла или URL
imagecreatefrompng	Создает новое изображение из файла или URL
imagecreatefromstring	Создание нового изображения из потока представленного строкой
imagecreatefromwbmp	Создает новое изображение из файла или URL
imagecreatefromwebp	Создает новое изображение из файла или URL
imagecreatefromxbm	Создает новое изображение из файла или URL
imagecreatefromxpm	Создает новое изображение из файла или URL
imagecreatetruecolor	Создание нового полноцветного изображения
imagecrop	Crop an image to the given rectangle
imagecropauto	Crop an image automatically using one of the available modes
imagedashedline	Рисование пунктирной линии
imagedestroy	Уничтожение изображения
imageellipse	Рисование эллипса
imagefill	Заливка
imagefilledarc	Рисование и заливка дуги
imagefilledellipse	Рисование закрашенного эллипса
imagefilledpolygon	Рисование закрашенного многоугольника
imagefilledrectangle	Рисование закрашенного прямоугольника
imagefilltoborder	Заливка цветом
imagefilter	Применяет фильтр к изображению
imageflip	Flips an image using a given mode
imagefontheight	Получение высоты шрифта
imagefontwidth	Получение ширины шрифта
imageftbbox	Определение границ текста выводимого шрифтом freetype2
imagefttext	Нанесение текста на изображение, используя шрифты FreeType 2
imagegammacorrect	Применение гамма коррекции к GD изображению
imagegd2	Вывод GD2 изображения в браузер или файл
imagegd	Вывод GD-изображения в браузер или в файл
imagegetclip	Get the clipping rectangle
imagegif	Выводит изображение в браузер или пишет в файл
imagegrabscreen	Захватывает изображение с экрана
imagegrabwindow	Захватывает изображение окна
imageinterlace	Включение или выключение интерлейсинга
imageistruecolor	Определяет, является ли изображение полноцветным
imagejpeg	Выводит изображение в браузер или пишет в файл
imagelayereffect	Установка флага альфа сопряжения для использования

	эффектов наложения изображений
imageline	Рисование линии
imageloadfont	Загрузка шрифта
imageopenpolygon	Draws an open polygon
imagepalettecopy	Копирование палитры из одного изображения в другое
imagepalettetotruecolor	Converts a palette based image to true color
imagepng	Вывод PNG изображения в браузер или файл
imagepolygon	Рисование многоугольника
imagepsbbox	Выдает параметры рамки, обрамляющей текст написанный шрифтом PostScript Type1
imagepsencodefont	Изменение вектора кодировки шрифта
imagepsextendfont	Растягивание или сжатие шрифта
imagepsfreefont	Освобождение памяти, занятой шрифтом PostScript Type
imagepsloadfont	Загрузка шрифта PostScript Type 1 из файла
imagepsslantfont	Наклон шрифта
imagepstext	Рисование текста поверх изображения, используя шрифты PostScript Type1
imagerectangle	Рисование прямоугольника
imageresolution	Get or set the resolution of the image
imagerotate	Поворот изображения с заданным углом
imagesavealpha	Установка флага сохранения всей информации альфа компонента (в противовес одноцветной прозрачности) и сохранение PNG изображения
imagescale	Scale an image using the given new width and height
imagesetbrush	Установка изображения (кисти), посредством которого будут рисоваться линии
imagesetclip	Set the clipping rectangle
imagesetinterpolation	Set the interpolation method
imagesetpixel	Рисование точки
imagesetstyle	Установка стиля рисования линий
imagesetthickness	Установка толщины линий
imagesettile	Установка изображения, которое будет использовано в качестве элемента мозаичной заливки
imagestring	Рисование строки текста горизонтально
imagestringup	Рисование строки текста вертикально
imagesx	Получение ширины изображения
imagesy	Получение высоты изображения
imagetruecolortopalette	Преобразование полноцветного изображения в палитровое
imagettfbbox	Получение параметров рамки обрамляющей текст написанный TrueType шрифтом
imagettftext	Рисование текста на изображении шрифтом TrueType
imagetypes	Возвращает список типов изображений, поддерживаемых PHP сборкой
imagewbmp	Выводит изображение в браузер или пишет в файл
imagewebp	Output a WebP image to browser or file
imagexbm	Вывод XBM изображения в браузер или файл
iptcembed	Встраивание двоичных IPTC данных в JPEG изображение
iptcparse	Разбор двоичных IPTC данных на отдельные тэги
jpeg2wbmp	Конвертирует изображение из формата JPEG в WBMP
png2wbmp	Преобразование PNG файла в WBMP

ХОД РАБОТЫ

1. Создать сценарий, выводящий изображение в браузер.
2. Создать сценарий, выводящий черный квадрат размером 300 x 300 в браузер.
3. Нарисовать в этом квадрате желтые диагонали.
4. Нарисовать смайлик.
5. Залить область смайлика цветом.

САМОСТОЯТЕЛЬНАЯ РАБОТА

1. Создать открытку, содержащую изображение и текст.

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ № 11-12

Работа с файлами, каталогами и базами данных

Цель занятия: формирование навыков по созданию сценариев, содержащих функции обработки файлов, каталогов и баз данных.

ТЕОРЕТИЧЕСКИЙ МАТЕРИАЛ

Работа с файлами

Правильно работать с файлами должен уметь каждый программист. Данная статья ориентирована на начинающих PHP программистов, однако «сборник рецептов» будет полезен и продвинутым пользователям.

Работа с файлами разделяется на 3 этапа:

1. Открытие файла.
2. Манипуляции с данными.
3. Закрытие файла.

I. Открытие файла

Для того чтобы открыть файл в среде PHP используется функция fopen(). Обязательными параметрами этой функции является имя файла и режим файла.

```
$fp = fopen('counter.txt', 'r');
```

Согласно документации PHP выделяют следующие виды режимов файлов:

– r – открытие файла только для чтения.

– r+ - открытие файла одновременно на чтение и запись.

– w – создание нового пустого файла. Если на момент вызова уже существует такой файл, то он уничтожается.

– w+ - аналогичен r+, только если на момент вызова фай такой существует, его содержимое удаляется.

– a – открывает существующий файл в режиме записи, при этом указатель сдвигается на последний байт файла (на конец файла).

– a+ - открывает файл в режиме чтения и записи при этом указатель сдвигается на последний байт файла (на конец файла). Содержимое файла не удаляется.

Примечание: в конце любой из строк может существовать еще один необязательный параметр: b или t. Если указан b, то файл открывается в режиме бинарного чтения/записи. Если же t, то для файла устанавливается режим трансляции перевода строки, т.е. он воспринимается как текстовый.

Для демонстрации рассмотрим следующий сценарий:

```
<?php
//Открывает файл в разных режимах
$fp = fopen('counter.txt', 'r'); // Бинарный режим
$fp = fopen('counter.txt', 'rt'); // Текстовый режим
$fp = fopen("http://www.yandex.ru", "r");// Открывает HTTP
соединение на чтение
$fp = fopen("ftp://user:password@example.ru", 'w'); //Открываем
FTP соединение с указанием логина и пароля
?>
```

II. Манипуляции с данными файла

Записывать данные в файл при помощи PHP можно при помощи функции fwrite(). Это функция принимает 2 обязательных параметра и 1 необязательный. В качестве обязательных параметров выступает дескриптор файла и режим файла:

```
<?php
$fp = fopen("counter.txt", "a"); // Открываем файл в режиме
записи
$mytext = "Это строку необходимо нам записать\r\n"; // Исходная
строка
$test = fwrite($fp, $mytext); // Запись в файл
if ($test) echo 'Данные в файл успешно занесены.';
else echo 'Ошибка при записи в файл.';
fclose($fp); //Закрытие файла
?>
```

Для построчного считывания файла используют функцию fgets(). Функция принимает 2 обязательных параметра:

```
<?php
$fp = fopen("counter.txt", "r"); // Открываем файл в режиме
чтения
if ($fp)
{
while (!feof($fp))
{
$mytext = fgets($fp, 999);
echo $mytext."<br />";
}
}
else echo "Ошибка при открытии файла";
fclose($fp);
?>
```

Примечание:

В данном примере значение 999 определяет количество символов, которые будут считываться до тех пор, пока указатель не достигнет конца файла (EOF).

Для того, чтобы считать файл как единое целое, нужно использовать функцию readfile(), принимающая 1 обязательный параметр. Функция открывает файл, отображает его содержимое в окне браузера, а затем закрывает файл:

```
<?php
echoreadfile("counter.txt");
?>
```

Также можно использовать функцию fpassthru() которая принимает 1 обязательный параметр. Перед использованием этой функции необходимо открыть файл в режиме чтения. По окончанию считывания файла функция автоматически закрывает файл(при этом дескриптор файла становиться недействительным).

```
<?php
$fp = fopen("counter.txt", "r"); // Открываем файл в режиме
чтения
if ($fp) echo fpassthru($fp);
elseecho "Ошибка при открытии файла";
?>
```

Очень часто встречаются ситуации, когда необходимо содержимое сайта считать в массив. Эту возможность предусматривает использование функции file(). При вызове этой функции, каждая строка файла сохранятся в отдельном элементе указанного массива.

Примечание: Не следует применять функцию file() к двоичным файлам (binary-safe), т.к. она не является безопасной в плане считывания двоичных файлов, если при этом, где-то встретиться символ конца файла (EOF), то она не гарантирует вам чтение всего двоичного файла.

```
<?php
$file_array = file("counter.txt"); // Считывание файла в массив
$file_array

// Работа с данными массива

?>
```

В конце статьи, вы найдете хороший «сборник рецептов» по массивам, который дает решение многих проблем, с которыми ежедневно встречается веб-программист.

Давайте представим ситуацию, когда файл необходимо считать по символам. Для этого мы можем воспользоваться функцией fgetc(). Функция принимает единственный параметр. Функция полезна если нам необходимо найти какой-либо символ или количество одинаковых символов.

```
<?php
$fp = fopen("counter.txt", "r"); // Открываем файл в режиме
чтения
if ($fp)
{
  while(!feof($fp))
  {
  $char = fgetc($fp);
  if ($char == 'c') $i = $i + 1;// Находим символ «c»
  }
echo 'Количество букв "c" в файле: '. $i;
}
else echo "Ошибка при открытии файла";
?>
```

III. Закрытие файла

Закрытие файла происходить с помощью функции fclose(), которая принимает 1 обязательный параметр.

```
<?php
$fp = fopen("counter.txt", "r");
```

```
if ($fp)
{
echo 'Файл открыт';
fclose($fp); // Закрытие файла
}
?>
```

Работа с каталогами

Начнём с самого простого: создание каталога в PHP:

```
<?php
  mkdir("new_dir");
?>
```

После запуска этого скрипта у Вас будет создан пустой каталог "new_dir".

Удалить пустой каталог очень просто. Для этого используется функция rmdir().

```
<?php
  rmdir("new_dir");
?>
```

А вот теперь перейдём к работе с содержимым каталогов через PHP. Здесь есть очень простые правила, которые необходимо соблюдать. Все эти правила очень логичны, и Вы их применяете, когда вручную просматриваете содержимое каталогов:

1. Открыть каталог.
2. Считать содержимое.
3. Закрыть каталог.

Чтобы не мучить Вас в ожиданиях, сразу приведу код, который выводит имена файлов и категорий внутри заданного каталога:

```
<?php
  $dir = opendir("images");
  while (($f = readdir($dir)) !== false)
    echo $f."<br />";
  closedir($dir);
?>
```

В результате Вы увидите список всех файлов и каталогов внутри каталога "images". Также Вы увидите два интересных имени "." и "..". Первый означает "текущий каталог", а ".." - родительский.

Теперь подробно о функциях, используемых в этом примере:

– Функция opendir(string $path) - открывает каталог, находящийся по пути $path, а также возвращает дескриптор, необходимый для работы с этим каталогом.

– Функция readdir(resource $dir) - считывает текущий элемент в каталоге dir. Текущий элемент задаётся указателем, который сдвигается при каждом вызове. Поэтому получается, что каждый раз эта функция возвращает новый элемент из каталога. Когда все элементы закончились, то функция readdir()возвращает false.

– Функция closedir(resource $dir) - закрывает каталог dir.

Это все самые важные функции для работы с каталогами в PHP. Однако, хочется добавить ещё одну очень важную деталь по поводу функции rmdir(), которая удаляет каталог. Если Вы внимательно читали, то я написал, что эта функция удаляет "пустой

каталог", то есть в котором нет ни одного файла и каталога (кроме "." и ".."). Другими словами, если в каталоге будет хотя бы один файл, то функция rmdir() не сработает. Вот как решить эту проблему Вы узнаете в следующей статье, поэтому подписывайтесь на обновления, чтобы не пропустить её появление.

Работа с базами данных

1. Администрирование базы данных

Способы администрирования БД в порядке убывания удобства:

- phpMyAdmin (весьма рекомендую!)
- Написать скрипт, который бы передёргивал базу (см. пример)
- mysql.exe в пакете mysql

С 1 способом не придётся изучать запросы ALTER TABLE, ADD COLUMN и т.п. Пару слов о втором способе. Это обходная технология, которую применяют, не зная про phpMyAdmin и утилиту mysqldump. В скрипте пишутся команды, удаляющие базу и создающие её вновь.

На будущее: если у вас будет несколько сайтов, использующих БД, то хотя бы в пределах домашнего сервера создайте несколько баз. Это облегчит работу серверу и исключит возможность путаницы таблиц. В общем, правила работы с БД те же, что и с сайтом - держать в отдельной директории от других.

2. Соединение с сервером БД

Осуществляется при помощи функции mysql_connect:

```
$connect = mysql_connect(<хост>, <логин>, <пароль>);
```

По умолчанию, на mysql-сервере в таблице пользователей есть пользователь root, который может иметь доступ только с localhost-а, то есть с того же самого компьютера, где стоит сервер mysql.

Что происходит, когда мы вызываем функцию mysql_connect?

С началом выполнения вашего скрипта, php выделяет в своей памяти место для информации о нём и его переменных. В информации о выполняемом скрипте хранится, в том числе, и информация о соединениях с базами данных. Переменная $connect - грубо говоря указатель на место, где данная информация хранится. Переменная эта точно такая же, как и остальные - если вы используете функции, то надо объявлять глобальные переменные, чтобы обратиться к ней.

Почему вообще используется переменная? Это на случай, если для работы вам необходимо использовать несколько серверов баз данных (или, например, для обеспечения большей безопасности вы используете разные логины, у которых могут быть разные привилегии). В таких случаях в каждом запросе нужна определённость, по какому, так сказать, каналу идёт команда. Но если вы используете только одно соединение, указывать его в параметрах функций запросов (о них - ниже) не нужно - php находит первое (и в данном случае единственное) установленное соединение и использует его.

3. Запрос-выборка и обработка результатов

Механизм работы функций запросов к БД такой же, как и у функции соединения: функции передаются параметры запроса и (если надо) соединения, а результат записывается в переменную:

```
$result = mysql_db_query(string база данных, string
запрос [, переменная соединения]);
```

или

```
$result = mysql_query(string запрос [, переменная
соединения]);
```

ВНИМАНИЕ! Чтобы использовать функцию mysql_query, в которой база данных не указывается, надо предварительно выбрать используемую базу данных:

```
mysql_select_db(string база данных);
```

Теперь у нас есть переменная $result. Это указатель на результат выполнения запроса. Там есть сколько-то строк таблицы. Получить эти строки можно через функции mysql_fetch_row и mysql_fetch_array:

```
echo "<table>";
while ($row = mysql_fetch_array($result))
  echo "<tr><td>", $row["field1"], "</td><td>",
$row["field2"], "</td></tr>";
echo "</table>";
```

Функция mysql_fetch_array выдаёт в указанную переменную (в данном случае $row) массив, индексы которого - имена полей (причём, если вы в списке полей запроса пишете table.field, то индекс массива будет field).

Mysql_fetch_row выдаёт массив, индексы которого - числа, начиная с 0.

Какой функцией лучше пользоваться?

Если вы запрашиваете звёздочку, т.е. все поля таблицы, а выводить поля нужно в определённой последовательности (когда, например, у таблицы рисуется шапка), лучше пользоваться mysql_fetch_array.

Если вы запрашиваете одно-два-три поля, чётко зная их последовательность, можно делать mysql_fetch_row - это уменьшит объем кода программы.

4. Запросы-действия

Это команды DELETE и UPDATE. Подобные запросы - в "правах" такие же, как и SELECT, поэтому отправка команды серверу происходит тем же способом - mysql_query (mysql_db_query). Но в данном случае функция не возвращает результата:

```
$result = mysql_query("SELECT * FROM sometable");
но
mysql_query("DELETE FROM sometable WHERE id=...");
```

Соответственно, если мы выполним запрос-выборку и не запишем результат в переменную, данные не будут храниться нигде.

5. Обработка ошибок запросов

Сообщение о последней ошибке можно получить через функцию mysql_error:

```
echo "Ошибка базы данных. MySQL пишет:", mysql_error();
```

Если результат функции пишется в переменную, можно проверить её:

```
$result = mysql_query($request);
if (!$result)
  echo "Ошибка базы данных. MySQL пишет:", mysql_error();
else {
  echo "<table>";
  while ($row = mysql_fetch_array($result))
    echo "<tr><td>", $row["field1"], "</td><td>",
$row["field2"], "</td></tr>";
  echo "</table>";
  };
```

Если в переменную не пишем, то так:

```
$request = "UPDATE (...)";
mysql_query($request);
if (!mysql_error())
  echo "Обновление данных прошло успешно!";
else echo "Ошибка базы данных. MySQL пишет:",
mysql_error();
```

Если запрос генерируется автоматически, можно выводить и сам запрос (полезно создавать переменную, которая бы его содержала, и использовать её в качестве параметра функции).

6. Хранение файлов в базе данных mySQL

Если Вы собрались хранить загружаемы файлы в базе данных, Вам необходимо помнить следующие моменты:

- Необходимо использовать поле типа BLOB
- Перед тем, как класть в базу, не забыть применить к строке mysql_escape_string()
- При отображении файла необходимо указывать заголовок content/type

Помните, что скрипт отображающий ваш HTML никак не связан со скриптом, который должен выводить изображение. Это должны быть два различные приложения.

Хранение картинок в базе не является хорошем стилем. Гораздо удобней хранить в базе лишь пути к файлам изображений.

ХОД РАБОТЫ

1. Создайте простую форму добавления заметок. При добавлении заметки, содержимое должно сохраняться в базе. Примечание: используйте пример работы со стеной сайта, который мы рассмотрели на занятии

2. Напишите скрипт, который выведет выпадающий список из записей базы данных

3. *Выведите на страницу несколько произвольных записей из базы (это могут быть заметки или товары). Рядом с каждой разместите ссылку «Нравится» или «Like» и количество отметок. При нажатии на ссылку, счетчик отметок должен меняться в базе и новое значение отображаться на странице

САМОСТОЯТЕЛЬНАЯ РАБОТА

1. Написать скрипт, создающий базу данных «Библиотека». Таблица «Книги»: id, автор, название, кол-во страниц, год издания. Продемонстрировать возможности добавления, удаления данных о книгах.
2. Подготовиться к тестированию – выучить команды по работе с файлами, каталогами и базами данных.

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ № 13

Работа с датой и временем, с регулярными выражениями, с cookies

Цель занятия: формирование навыков по созданию сценариев, выводящих дату и время, созданию сценариев с регулярными выражениями, созданию сценариев, содержащих COOKIES.

ТЕОРЕТИЧЕСКИЙ МАТЕРИАЛ

В распределенных системах, таких, как Интернет, время играет особую роль. Из-за незначительного расхождения системных часов игрок на рынке Forex может потерять десятки тысяч долларов в течение нескольких минут; система деловой разведки ошибется в составлении прогноза; серверы NNTP в процессе синхронизации потеряют важную информацию, нужную пользователю и т.д.

PHP содержит множество функций для работы с датой и временем. Наиболее употребимыми являются:

– **time( )** - возвращает текущее абсолютное время. Это число равно количеству секунд, которое прошло с полуночи 1 января 1970 года (с начала эпохи UNIX).

– **getdate( )** - считывает информацию о дате и времени. Возвращает ассоциативный массив, содержащий информацию по заданному или по текущему (по умолчанию) времени. Массив содержит следующие элементы:

seconds	Секунды (0-59)
minutes	Минуты (0-59)
hours	Часы (0-23)
mday	День месяца (1-31)
wday	День недели (0-6), начиная с воскресенья
mon	Месяц (1-12)
year	Год
yday	День года (0-365)
weekday	Название дня недели (например, Friday)
month	Название месяца (например, January)
0	Абсолютное время

– **date( )** - форматирование даты и времени. Аргументы: строка формата и абсолютное время. Второй аргумент необязателен. Возвращает строку с заданной или текущей датой в указанном формате. Строка формата может содержать следующие коды:

a	Включено обозначение "am" или "pm"
A	Включено обозначение "AM" или "PM"
d	День месяца (01-31)
D	Сокращенное название дня недели (три буквы)
F	Полное название месяца
g	Часы (12-часовой формат без ведущих нулей)
G	Часы (24-часовой формат без ведущих нулей)
h	Часы (12-часовой формат)
H	Часы (24-часовой формат)

i	Минуты (00-59)
j	День месяца без ведущих нулей (1-31)
l	Полное название дня недели
L	Признак високосного года (0 или 1)
m	Месяц (01-12)
M	Сокращенное название месяца (три буквы)
n	Месяц (1-12)
s	Секунды (00-59)
t	Количество дней в данном месяце (от 28 до 31)
U	Абсолютное время
w	Номер дня недели (0 - воскресенье, 6 - суббота)
y	Год (два разряда)
Y	Год (четыре разряда)
z	День года (0-365)
Z	Смещение часового пояса в секундах (от -43200 до 43200)

– **mktime( )** - возвращает абсолютное время, которое затем можно использовать с функциями date() или getdate(). Принимает до шести целочисленных аргументов в следующем порядке:

– часы
– минуты
– секунды
– месяц
– день
– месяца
– год

– **checkdate( )** - проверка правильности даты. Аргументы: месяц, день, год. Возвращает true, если дата правильная, т.е.

– месяц - целое число от 1 до 12;
– день - целое число, не превышающее общего количества дней в данном месяце. При этом високосные годы обрабатываются корректно;
– год - целое число от 1 до 32767.

– **strftime( )** - формирование локальной даты и времени. Аргументы: строка формата и абсолютное время. Второй аргумент необязателен. Возвращает строку с заданной или текущей датой в указанном формате. При этом названия месяцев и дней недели извлекается из локали, выбранной с помощью функции setlocate( ) Строка формата может содержать следующие коды:

%a	Сокращенное название дня недели
%A	Полное название дня недели
%b	Сокращенное название месяца
%B	Полное название месяца
%c	Предпочтительный формат даты и времени
%C	Номер века
%d	День месяца (1-31)

%D	То же, что и %m/%d/%y
%e	Месяц (1-12)
%h	То же, что и %b
%H	Часы (24-часовой формат)
%I	Часы (12-часовой формат)
%j	День года (0-365)
%m	Месяц (1-12)
%M	Минуты
%n	Символ новой строки
%p	Включено обозначение "am" или "pm"
%r	Время с использованием a.m./p.m.-нотации
%R	Время в 24-часовом формате
%S	Секунды (00-59)
%t	Символ табуляции
%T	То же, что и %H:%M:%S
%u	Номер дня недели (1 - понедельник, 7 - воскресенье)
%U	Номер недели. Отсчет начинается с первого воскресенья года
%V	Номер недели по ISO 8601:1988. Первая неделя должна иметь не менее четырех дней, а понедельник считается первым днем
%W	Номер недели. Отсчет начинается с первого понедельника года
%w	Номер дня недели (0 - воскресенье, 6 - суббота)
%x	Предпочтительный формат даты без времени
%X	Предпочтительный формат времени без даты
%y	Год (два разряда)
%Y	Год (четыре разряда)
%Z	Часовой пояс (имя или сокращение)
%%	Символ "%"

Любая другая информация, включенная в строку формата, будет вставлена в возвращаемую строку.

Функции регулярных выражений

В PHP существует несколько функций для работы с регулярными выражениями. Все они используют один и тот же парсер регулярных выражений для своей работы, но при этом преследуют различные цели. Ниже мы рассмотрим все эти функции. Я буду приводить описание синтаксиса каждой функции в том виде, в котором она описана в PHP Manual, чтобы вам легче было разобраться.

Функция preg_match()

Синтаксис:

```
int preg_match (string pattern, string subject [, array
matches])
```

Эта функция предназначена для проверки того, совпадает ли заданная строка (subject) с заданным регулярным выражением (pattern). В качестве результата функция

возвращает 1, если совпадения были найдены и 0, если нет. Если при вызове функции был задан необязательный параметр matches, то после работы функции ему будет присвоен массив, содержащий результаты поиска по заданному регулярному выражению. Заметьте, что вне зависимости от того, сколько именно совпадений было найдено при поиске - вам будет возвращено только первое совпадение. Рассмотрим пример того, как это работает:

```
<?php
$str = "123 234 345 456 567";                    // Строка
для поиска
$result = preg_match('/\d{3}/',$str,$found);     //
Производим поиск
echo "Matches: $result<br>";                     // Выводим
количество найденных совпадений
print_r($found);                                 // Выводим
результат поиска
?>
```

Результатом работы этой программы будет:

```
Matches: 1

Array
(
    [0] => 123
)
```

Если вы внимательно прочитали предыдущий выпуск и понимаете, как работают регулярные выражения, то вы должны заметить, что реально функция preg_match() обнаружила в заданной строке 5 совпадений с заданным выражением, но вернула только первое из них. Казалось бы, что в этом случае было бы логичнее возвращать результаты поиска в виде строки, а не в виде массива, но это не так. Вспомните, что регулярное выражение может содержать в себе внутренние регулярные выражения, которые также возращают результат. А для того, чтобы вернуть результаты поиска по всем регулярным выражениям нам как раз и требуется массив. Для того, чтобы проиллюстрировать сказанное выше давайте немного изменим регулярное выражение и посмотрим на результат:

```
<?php
$str = "123 234 345 456 567";
// Теперь мы не просто ищем трехзначное число,
// но и получаем его среднюю цифру
$result = preg_match('/\d(\d)\d/',$str,$found);
echo "Matches: $result<br>";
print_r($found);
?>
```

Результат будет следующим:

```
Matches: 1

Array
(
```

```
    [0] => 123
    [1] => 2
)
```

Как видите - здесь присутствуют результаты поиска по всем имеющимся регулярным выражениям.

Функция preg_match_all()

Синтаксис:

```
int preg_match_all (string pattern, string subject, array
matches [, int order])
```

Эта функция очень похожа на предыдущую и предназначена для тех же самых целей. Единственное ее отличие от preg_match() состоит в том, что она осуществляет "глобальный" поиск в заданном тексте по заданному регулярному выражению и, соответственно, находит и возвращает все имеющиеся совпадения. Посмотрим, как отличается работа этой функции на том же самом примере:

```
<?php
$str = "123 234 345 456 567";
$result = preg_match_all('/\d{3}/',$str,$found);
echo "Matches: $result<br>";
print_r($found);
?>
```

Результат работы:

```
Matches: 5
   Array
   (
       [0] => Array
           (
               [0] => 123
               [1] => 234
               [2] => 345
               [3] => 456
               [4] => 567
           )
)
```

Как видите - здесь мы получили все найденные совпадения и их количество в качестве результата.

Необходимо обратить ваше внимание на дополнительный параметр, появившийся в этой функции по сравнению с preg_match(): order. Значение этого параметра определяет структуру выходного массива с найденными совпадениями. Его значение может быть одним из перечисленных ниже:

– PREG_PATTERN_ORDER - результаты поиска будут сгруппированы по номеру регулярного выражения, которое возвратило этот результат (это значение используется по умолчанию).

– PREG_SET_ORDER - результаты поиска будут сгруппированы по месту их нахождения в тексте

Для того, чтобы лучше понять разницу между этими значениями, посмотрим на результат работы одного и того же скрипта при использовании каждого из них:

Сначала посмотрим на то, как выглядит результат при использовании PREG_PATTERN_ORDER:

```
<?php
$str = "123 234 345 456 567";
$order = PREG_PATTERN_ORDER;
$result = preg_match_all('/\d(\d)\d/',$str,$found,$order);
print_r($found);
?>
```

Результат:

```
Array
  (
      [0] => Array
          (
              [0] => 123
              [1] => 234
              [2] => 345
              [3] => 456
              [4] => 567
          )
      [1] => Array
          (
              [0] => 2
              [1] => 3
              [2] => 4
              [3] => 5
              [4] => 6
          )
)
```

Как видите - массив результатов содержит внешние индексы, соответствующие номерам регулярных выражений, от которых получен результат (индекс 0 имеет основное регулярное выражение). По этим индексам в массиве расположены массивы, содержащие непосредственно найденную информацию, причем индекс в этом внутреннем массиве соответствует "порядковому номеру" данного фрагмента в исходном тексте.

Теперь попробуем то же самое, но с PREG_SET_ORDER:

```
<?php
$str = "123 234 345 456 567";
$order = PREG_SET_ORDER;
$result = preg_match_all('/\d(\d)\d/',$str,$found,$order);
print_r($found);
?>
```

Результат:

```
Array
 (
     [0] => Array
           (
               [0] => 123
               [1] => 2
           )
     [1] => Array
           (
               [0] => 234
               [1] => 3
           )
     [2] => Array
           (
               [0] => 345
               [1] => 4
           )
     [3] => Array
           (
               [0] => 456
               [1] => 5
           )
     [4] => Array
           (
               [0] => 567
               [1] => 6
           )
)
```

Как видите - здесь основной массив содержит результаты поиска, сгруппированные по порядку их нахождения в тексте, причем каждый результат представляет собой массив с результатами поиска по этому найденному фрагменту для всех имеющихся регулярных выражений.

Функция preg_replace()

Синтаксис:

```
mixed preg_replace (mixed pattern, mixed replacement, mixed
subject [, int limit])
```

Эта функция позволит вам произвести замену текста по регулярному выражению. Как и в предыдущих функциях, здесь производится поиск по регулярному выражению pattern в тексте subject, и каждый найденный фрагмент текста заменяется на текст, заданный в replacement. Задание необязятельного параметра limit позволит ограничить количество заменяемых фрагментов в тексте.

Например, нам необходимо "сжать" текст, убрав из него все лишние пробелы и символы перевода строки:

```
<?php
$text = "there     is\t\n\t\t some text  \n  \t just  \n\n\n
for     test";
```

```
echo "<b>Перед заменой:</b>\n$text\n\n";
$text = preg_replace("/(\n \s{2,})/"," ",$text);
echo "<b>После замены:</b>\n$text";
?>
```

Результатом работы данной программы будет следующий текст:

```
Перед заменой:
there     is
          some text
     just


 for     test

После замены:
there is some text just for test
```

Как видите - всего одна строчка позволила нам решить достаточно нетривиальную в обычной практике задачу. Объяснять само регулярное выражение я не буду, если вы внимательно прочитали предыдущий выпуск - понять его вам будет несложно.

Однако основная прелесть этой функции, которая и придает ей всю ее мощь - это тот факт, что вы можете ссылаться на результаты поиска при генерации замещающего текста. В качесте примера покажу, как можно очень быстро и элегантно решить задачу, которая возникает достаточно часто - конвертация дат из одного формата в другой. Как вы знаете, на Западе обычно используется формат mm/dd/yyyy, тогда как у нас обычно - dd.mm.yyyy. Следующий пример осуществляет конвертацию дат между этими форматами в заданном тексте:

```
<?php
$text = 'Today is 11/16/2001';
$text =
preg_replace("/(\d{2})\/(\d{2})\/(\d{4})/","\\2.\\1.\\3",$text)
;
echo $text;
?>
```

Результат работы этой программы:

```
Today is 16.11.2001
```

Обратите внимание на текст, используемый для замены. В нем использованы т.н. backreferences, т.е. ссылки на найденный ранее текст. Всего таких ссылок может быть не более 100 с номерами от 0 до 99 (соответственно в тексте они выглядят как \0, \1, \2 ... \99). Backreference с номером 0 будет заменена на весь найденный текст, \1 - на текст, найденный первым внутренним регулярным выражением, \2 - вторым и т.д. Номерв внутренним регулярным выражениям присваиваются по мере их находжения в тексте, т.е. слева-направо. В нашем случае \1 - это месяц, \2 - день, \3 - год.

Помимо стандартного синтаксиса регулярных выражений, в PHP, совместно с функцией preg_replace() используется еще один дополнительный модификатор - 'e'. Его использование заставляет PHP рассматривать текст замены не как текст, а как PHP код,

что дает возможность еще больше расширить сферу применения этой функции в вашем коде. Следующий пример демонстрирует использование этого модификатора - он производит замену всех целых десятичных чисел в тексте на их шестнадцатиричные эквиваленты:

```
<?php
$text = "123 234 345 456 567";
$text = preg_replace("/\d+/e","'0x'.dechex('\\0')",$text);
print_r($text);
?>
```

Результатом работы этой программы будет:

```
0x7b 0xea 0x159 0x1c8 0x237
```

И еще одно. Функция preg_replace() также умеет работать с массивами регулярных выражений. Т.е. это позволит вам осуществить поиск и замену сразу по множеству условий! В качестве примера приведу фрагмент кода, описанный в PHP Manual и осуществляющий конвертацию HTML документа в текст при помощи всего лишь одного вызова preg_replace()!

```
// $document should contain an HTML document.
// This will remove HTML tags, javascript sections
// and white space. It will also convert some
// common HTML entities to their text equivalent.

$search = array ("'<script[^>]*?>.*?</script>'si",  // Strip
out javascript
                 "'<[\/\!]*?[^<>]*?>'si",           // Strip
out html tags
                 "'([\r\n])[\s]+'",                 // Strip
out white space
                 "'&(quot #34);'i",                 // Replace
html entities
                 "'&(amp #38);'i",
                 "'&(lt #60);'i",
                 "'&(gt #62);'i",
                 "'&(nbsp #160);'i",
                 "'&(iexcl #161);'i",
                 "'&(cent #162);'i",
                 "'&(pound #163);'i",
                 "'&(copy #169);'i",
                 "'&#(\d+);'e");                    // evaluate
as php

$replace = array ("",
                  "",
                  "\\1",
                  "\"",
                  "&",
                  "<",
                  ">",
                  " ",
```

```
                    chr(161),
                    chr(162),
                    chr(163),
                    chr(169),
                    "chr(\\1)");

$text = preg_replace ($search, $replace, $document);
```

Сами по себе регулярные выражения очень просты, интересно лишь их совместное использование для решения общей задачи.

Функция preg_replace_callback()

Синтаксис:

```
mixed preg_replace_callback (mixed pattern, mixed callback,
mixed subject [, int limit])
```

Эта функция является расширенной версией функции preg_replace() (хотя, казалось бы, чего еще можно пожелать?). Единственным отличием ее от preg_replace() является то, что в качестве текста для замены в ней задается не сам текст, а имя функции, которая будет производить обработку найденного текста и возвращать замещающий текст. Т.е. с использованием этой функции мощь инструментария PHP по обработке текста становится поистине безграничной! В качестве примера хочу привести фрагмент кода, который выполняет работу, аналогичную той, что производится механизмом сессий в PHP: добавление дополнительного аргумента (идентификатора сессии) к каждой ссылке внутри HTML документа.

```
<?php
// Список тегов и аттрибутов, к котроым необходимо
// добавить дополнительный параметр.
// Формат строки:
// <имя тега>[ <имя аттрибута>]+
// Т.е. сначала идет имя тега, а затем, через пробел,
// имена одного или нескольких аттрибутов.
$tagsList = array(
    'a href',
    'area href',
    'frame src',
    'input src',
    'img src',
    'form action'
);
// Идентификатор сессии
$sid = 12345;
// HTML документ для обработки. Здесь, в качестве примера
// мы берем его из внешнего файла, но вообще-то метод
// получения исходного документа может быть различным.
$document = join('',file('document.html'));
// Начинаем обработку всех тегов, указанных в массиве $tagsList
foreach($tagsList as $tag)
{
```

```
// Разделяем список аттрибутов на составляющие
    $attrs = explode(' ',$tag);
// Получаем имя тега (в массиве $attrs остается лишь список
аттрибутов)
    $tag = array_shift($attrs);
// Выполняем "патч" всех имеющихся в документе ссылок,
содержащихся
// в каждом из аттрибутов текущего тега
    foreach($attrs as $attr)
        $document =
preg_replace_callback("/<".$tag.".+?".$attr."=[\'\"](.+?)[\'\"]
/si",

'callback',$document);
};
// Выводим документ и выходим
echo $document;
exit();
// Эта функция будет вызываться для каждой найденной
// ссылки в тексте HTML документа.
// На входе она получает результат поиска (массив,
// аналогичный возвращаемому функцией preg_match()).
// На выходе из функции должна быть строка с текстом замены.
function callback($data)
{
// Регулярное выражение, использованное для поиска находит
полные
// HTML теги, содержащие аттрибуты, в которых могут находиться
// URL адреса. Поскольку текст, возвращаемый данной функцией
будет
// использован для замещения всего найденного фрагмента текста
-
// нам необходимо взять полный текст, чтобы не потерять его при
// дальнейшей обработке. Он будет возвращен без изменений, если
// окажется, что аттрибут не содержит URL адреса.
    $href = $data[0];
// Используем функцию PHP для разбора URL адреса на
составляющие.
// В качестве "исходного материала" передаем содержимое
интересующего
// нас аттрибута, найденного внутренним регулярным выражением.
// Подробнее о том, что возвращает эта функция см. PHP Manual.
    $parts = parse_url($data[1]);
// Мы должны добвлять идентификатор сессии только к ссылкам,
которые
// являются "локальными" для данного сайта. Т.е. мы не должны
обрабатывать:
//  - полные URL адреса (<a href="http://www.php.net/">)
//  - указатели на "якоря" внутри страницы (<a href="#part2">)
    if ((!isset($parts['scheme'])) &&   // Если URL содержит
идентификатор
```

```
        (!isset($parts['host'])) &&     // протокола или имя
домена - это
                                        // полный URL адрес.
        (substr($data[1],0,1)!='#'))    // Если URL начинается
с символа '#'
                                        // то это ссылка на
"якорь" внутри страницы
    {
// Берем путь к странице, указанный в URL и добавляем
разделитель для параметров
// потому что нам необходимо будет добавить по крайней мере 1
параметр
        $href = $parts['path'].'?';
// Если в этом URL уже были какие-либо параметры - добавляем их
и добавляем
// разделитель. Заметьте, что в качестве разделителя
используется &amp;, а не &,
// это позволяет нам добиться совместимости с XHTML.
        if (isset($parts['query']))
            $href .= $parts['query'].'&amp;';
// Добавляем наш собственный параметр - идентификатор сессии
        $href .= 'sid='.$GLOBALS['sid'];
// Если в оригинальном URL была ссылка на фрагмент документа -
возвращаем ее
// на место.
        if (isset($parts['fragment']))
            $href .= '#'.$parts['fragment'];
// "Вставляем" новый URL на место того, который был там раньше
        $href = str_replace($data[1],$href,$data[0]);
    };
// Возвращаем результат
    return($href);
};
?>
```

Пример может показаться немного громоздким, но это исключительно из-за обилия комментариев.

Функция preg_split()

Синтаксис:

```
array preg_split (string pattern, string subject [, int limit
[, int flags]])
```

Данная функция выполняет действие, аналогичное функциям split() и explode() - разбивает строку на части по какому-либо признаку и возвращает массив, содержащий части строки. Однако ее возможности по заданию правил разбиения больше, чем у этих функций, потому что в ее основе лежит механизм регулярных выражений, в мощи которого, я надеюсь, вы уже смогли убедиться. Если говорить более конкретно, то строка subject разбивается на части по разделителю, заданному регулярным выражением pattern. При этом количество фрагментов может быть ограничего необязятельным параметром limit. Кроме того эта функция поддерживает необязательный

параметр flags, который позволяет в некоторой степени контролировать процесс разбиения строки.

Параметр flags может принимать следующие значения (или их комбинации с использованием знака ' '):

– PREG_SPLIT_NO_EMPTY - возвращать только непустые части строк, полученные в результате разбиения.

– PREG_SPLIT_DELIM_CAPTURE - возвращать также результаты поиска по внутренним регулярным выражениям.

Рассмотрим пару примеров. Для начала - выражение, которое разбивает произвольный текст на отдельные слова:

```
<?php
$text = join('',file('my_text.txt'));
$words = preg_split("/\s+/s",$text);
print_r($words);
?>
```

Как видите - мы получаем содержимое файла 'my_text.txt' в виде строки, разбиваем его на отдельные слова и выводим содержимое массива слов, чтобы убедиться, что все работает правильно.

Еще один пример производит разбиение заданного слова на буквы (он описан в PHP Manual):

```
<?php
$str = 'string';
$chars = preg_split('//',$str,-1,PREG_SPLIT_NO_EMPTY);
print_r($chars);
?>
```

Значение -1 для параметра limit означает отсутствие лимита.

Функция preg_quote()

Синтаксис:

```
string preg_quote (string str [, string delimiter])
```

Эта функция - единственная, не относящаяся непосредственно к механизму регулярных выражений. Ее назначение - "квотинг" символов, имеющих специальное значение в синтаксисе регулярных выражений. Обычно это символы:

```
. \ + * ? [ ^ ] $ ( ) { } = ! < >   :
```

Все эти символы, встречающиеся в строке будут "отквочены" путем добавления символа '\' непосредственно перед каждым из них. Модифицированная таким образом строка будет возвращены в качестве результата.

Эта фцнкция также имеет необязательный параметр delimiter. Если этот параметр задан, то символ, переданный в качестве этого параметра тоже будет "отквочен" данной функцией.

Функция preg_grep()

Синтаксис:

```
array preg_grep (string pattern, array input)
```

Действие этой функции похоже на действие команды grep в Unix. Она ищет текст по регулярному выражению pattern, в массиве input и возвращает новый массив, содержащий только элементы, в которых были найдены совпадения с заданным регулярным выражением. К примеру у нас есть файл, содержащий в каждой строке числовую и текстовую информацию. Нам необходимо получить из этого файла только строки, содержащие числа:

Файл data.txt:

```
123
abc
php4
```

Код:

```
<?php
// Считываем содержимое файла в массив
$data = file('data.txt');
// Получаем массив, содержащий цифровую информацию
$numbers = preg_grep("/\d+/",$data);
// Выводим результат работы
print_r($numbers);
?>
```

Результат работы будет:

```
Array
(
    [0] => 123
    [2] => php4
)
```

Как видите - мы получили все строки, содержащие цифры. Если же нам, например нужно получить только цифры - то выражение необходимо немного изменить: /^\s*\d+\s*$/.

Работа с cookies-файлами

PHP прозрачно поддерживает HTTP cookies. Cookies - это механизм хранения данных браузером удаленной машины для отслеживания или идентификации возвращающихся посетителей. Вы можете установить cookies при помощи функций setcookie() или setrawcookie(). Cookies являются частью HTTP-заголовка, поэтому setcookie() должна вызываться до любого вывода данных в браузер. Это то же самое ограничение, которое имеет функция header(). Вы можете использовать функции буферизации вывода, чтобы задержать вывод результатов работы скрипта до того момента, когда будет известно, понадобится ли установка cookies или других заголовков.

Любые cookies, отправленные серверу браузером клиента, будут автоматически включены в суперглобальный массив *$_COOKIE*, если директива variables_order содержит букву "C". Для назначения нескольких значений одной cookie, просто добавьте *[]* к её имени.

В зависимости от значения опции register_globals, cookies могут быть представлены обычными PHP-переменными. Однако рекомендуется не полагаться на эту возможность, так как она обычно отключена в целях безопасности.

ХОД РАБОТЫ

1. Протестировать скрипты теоретической части.
2. Записать в тетрадь команды.

САМОСТОЯТЕЛЬНАЯ РАБОТА

1. Создать на странице поле для ввода электронной почты. Средствами PHP установить запрет на ввод недопустимых символов.
2. Написать скрипт по созданию **cookies.**
3. Написать скрипт по определению времени существования **cookies.**
4. Написать скрипт по удалению **cookies.**

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ № 14-16

Java Script

Цель занятия: формирование навыков по созданию сценариев Java Script.

Часть 1 Введение в JavaScript. Программное взаимодействие с HTML документами на основе DOM API.

ТЕОРЕТИЧЕСКИЙ МАТЕРИАЛ

Элементы языка JavaScript

JavaScript позволяет "оживить" веб-страницу. Это реализуется путем добавления к статическому описанию фрагмента исполняемого кода. JavaScript-сценарий может взаимодействовать с любыми компонентами HTML-документа и реагировать на изменение их состояния.

JavaScript не является строго типизированным языком, в переменных могут храниться практически любые типы данных.

Как и программа на языке Java, сценарий JavaScript выполняется под управлением интерпретатора. Однако если Java-приложение или Java-аплет компилируется в байтовый код, то сценарий JavaScript интерпретируется на уровне исходного текста.

Следует отметить, что языковые конструкции JavaScript совпадают с соответствующими средствами C++ и Java.

Структура сценария

Сценарием JavaScript считается фрагмент кода, расположенный между дескрипторами <SCRIPT> и </SCRIPT>:

```
Текст HTML-документа
<SCRIPT>
  Код сценария
</SCRIPT>
Текст HTML-документа
```

Переменные

В сценариях JavaScript переменные могут хранить данные любых типов: числа, строки текста, логические значения, ссылки на объекты, а также специальные величины, например "нулевое" значение `null` или значение `NaN`, которое сообщает о недопустимости операции.

Переменная в языке JavaScript объявляется с помощью ключевого слова `var`. Так, например, выражение

```
var selected = "first item";
```

создает переменную с именем `selected` и присваивает ей в качестве значения строку символов "`first item`". Переменные могут объявляться также автоматически. Это происходит при присвоении значения переменной, не встречавшейся ранее в данном сценарии. Так, в следующем примере создается переменная с именем `rating`, которой присваивается числовое значение, равное `512.5`:

```
rating = 512.5;
```

Объекты

В языке JavaScript не предусмотрены средства для работы с классами в том виде, в котором они реализованы в C++ или Java. Разработчик сценария не может создать подкласс на основе существующего класса, переопределить метод или выполнить какую-либо другую операцию с классом. Сценарию, написанному на языке JavaScript, в основном доступны лишь готовые объекты. Построение нового объекта приходится выполнять лишь в редких случаях.

Объекты содержат свойства (свойства объектов можно сравнить с переменными) и методы. Объекты, а также их свойства и методы идентифицируются именами. Объектами являются формы, изображения, гипертекстовые ссылки и другие компоненты веб-страницы, HTML-документ, отображаемый в окне браузера, окно браузера и даже сам браузер. В процессе работы JavaScript сценарий обращается к этим объектам, получает информацию и управляет ими.

Кроме того, разработчику сценария на языке JavaScript доступны объекты, не связанные непосредственно с HTML-документом. Их называют предопределенными,или независимыми объектами. С помощью этих объектов можно реализовать массив, описать дату и время, выполнить математические вычисления и решить некоторые другие задачи.

Первый объект, с которым необходимо познакомиться, чтобы написать простейший сценарий, - это объект `document`, который описывает HTML документ, отображаемый в окне браузера. Ниже приведен исходный текст веб-страницы, содержащей сценарий, действия которого сводятся к выводу строки текста в окне браузера.

```
<HTML>
  <HEAD>
    <TITLE>Первый сценарий JavaScript</TITLE>
  </HEAD>
  <BODY>
    <SCRIPT LANGUAGE="JavaScript">
      document.write("Проверка сценария JavaScript");
    </SCRIPT>
  </BODY>
</HTML>
```

Имена чувствительны к регистрам символов, и если вы попытаетесь обратиться к текущему документу по имени Document, интерпретатор JavaScript отобразит сообщение об ошибке.

Основное назначение сценариев JavaScript - создавать динамически изменяющиеся объекты, корректировать содержимое HTML-документов в зависимости от особенностей окружения, осуществлять взаимодействие с пользователем и т.д.

Операции

Набор операторов в JavaScript, их назначение и правила использования в основном совпадают с принятыми в языке C++. Исключением является операция задаваемая символом " + ".

В JavaScript символ "+" определяет как суммирование числовых значений, так и конкатенацию строк.

Так, например, в результате вычисления выражения

```
sum = 47 + 21;
```

переменной sum будет присвоено значение 68, а после выполнения операции

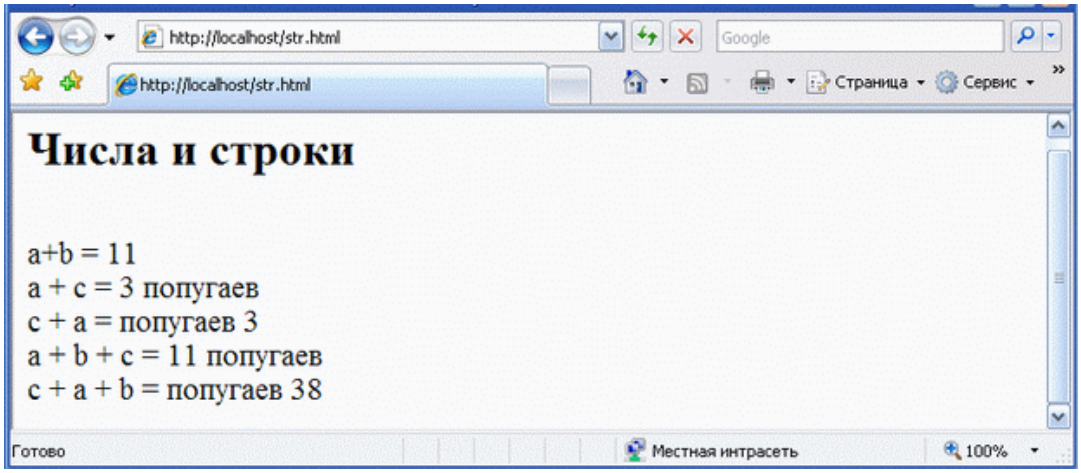
```
sum = "строка 1 " + "строка 2";
```

в переменную sum будет записана последовательность символов " строка 1 строка 2 ".

Рассмотрим еще один пример:

```
<HTML>
  <BODY>
    <H2>Числа и строки</H2><BR>
    <SCRIPT LANGUAGE="JavaScript">

      var a = 3;
      var b = 8;
      var c = " попугаев ";

      document.write("a+b="); document.write(a + b);
      document.write("<BR>");
      document.write( "a + c = "); document.write(a+c);
      document.write("<BR>");
      document.write("c + a = "); document.write (c + a);
      document.write ("<BR>");
      document.write ("a + b + c = "); document.write(a + b + c);
      document.write("<BR>");
      document.write("c + a + b = "); document.write(c + a + b);
      document.write("<BR>");

    </SCRIPT>
  </BODY>
</HTML>
```

В окне браузера приведенный выше HTML-код выглядит так, как показано на скриншоте



Первая строка отображает результат суммирования двух числовых значений, вторая и третья - результат конкатенации строки и символьного представления числа. Если операция суммирования чисел предшествует конкатенации, JavaScript вычисляет сумму чисел, представляет ее в символьном виде, затем производит конкатенацию двух строк. Если же первой в выражении указана операция конкатенации, то JavaScript сначала преобразует числовые значения в символьный вид, а затем выполняет конкатенацию строк.

Управляющие конструкции

Управляющие конструкции, используемые в языке C++, в основном применимы и в сценариях JavaScript.

В JavaScript дополнительно определены языковые конструкции, отсутствующие в C++, а именно: операторы `for...in` и `with`.

В примере с помощью оператора цикла на веб-странице формируется таблица умножения чисел.

```
<html>
<body>
<table>
<script language="JavaScript">
  document.write("<tr><td> </td>");
  for (i = 1; i < 10; i++) document.write("<td>"+i+" </td>");
  document.write("</tr>");
  for (i = 1; i < 10; i++)
  {
          document.write("<tr><td>" + i + " </td>");
          for (j = 1; j < 10; j++)
          {
                  document.write("<td    bgcolor='#00ffa0'>"  +  (i*j)  +
" </td>");}
          document.write("</tr>");
  }

</script>
</table>
</body>
</html>
```

Отдельного внимания заслуживает оператор `new`. Несмотря на то, что большинство объектов уже созданы браузером и доступны сценарию, в некоторых случаях приходится создавать объекты в процессе работы. Это относится к предопределенным объектам и объектам, определяемым разработчиком сценария. Для создания объекта используется оператор `new`, который вызывается следующим образом:

```
переменная =  new   тип_ объекта (параметры)
```

Функции

Формат объявления функции выглядит следующим образом:

```
function имя функции ([ параметры]) тело функции
```

Объявление функции начинается с ключевого слова `function`. Так же, как и в языке *C* для идентификации функции используется имя, при вызове функции могут передаваться параметры, а по окончании выполнения возвращаться значение. Однако, в отличие от *C*, тип возвращаемого значения и типы параметров не задаются. Ниже показаны два способа вызова функции

- `имя_функции ([параметры]);`
- `переменная = имя функции ([параметры]);`

Во втором случае значение, возвращаемое функцией, присваивается указанной переменной.

Область видимости переменных

Работа с переменными в теле функции подчиняется следующим правилам.

- Если переменная объявлена с помощью ключевого слова `var`, доступ к ней осуществляется по правилам, подобным тем, которые используются в языке *C*.
- Переменная, объявленная внутри функции, считается *локальной*. Область видимости такой переменной ограничивается телом функции, в которой она объявлена.
- Переменная, объявленная вне функции, считается *глобальной*. К ней можно обращаться из любой точки сценария.
- Если локальная и глобальная переменные имеют одинаковые имена, то в теле функции локальная переменная "маскирует" глобальную.

Если переменная создается автоматически, т.е. если она не объявлена с помощью ключевого слова `var`, но присутствует в левой части оператора прямого присваивания, то она считается глобальной и становится доступной из любой точки сценария.

HTML DOM

DOM (Document Object Model) – представляет собой стандарт консорциума W3C для программного доступа к документам HTML или XML. Фактически это платформо- и языково-нейтральный интерфейс, позволяющий программам и сценариям динамически обращаться и обновлять содержимое, структуру и стиль документа.

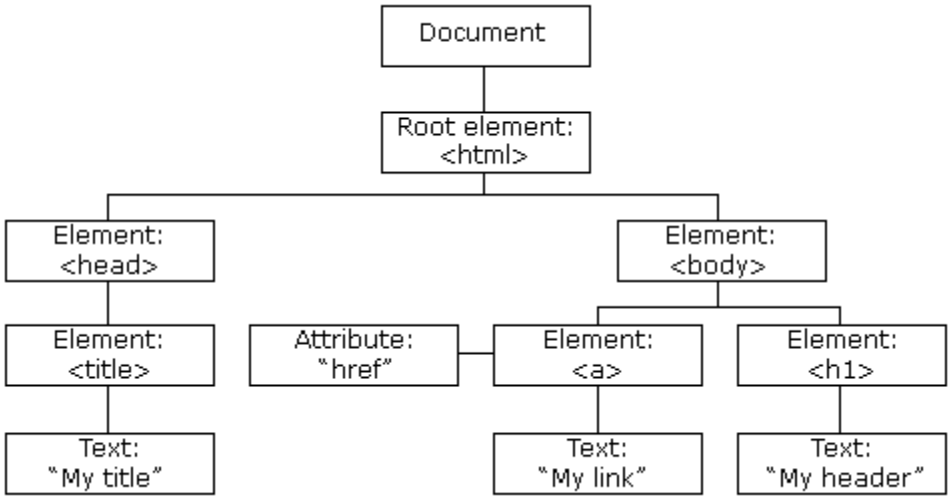
В рамках данного стандарта можно выделить 3 части:

- Core DOM – стандартная модель любого структурированного документа
- XML DOM - стандартная модель XML документа
- HTML DOM - стандартная модель HTML документа

DOM определяет объекты и свойства всех элементов документа и методы (интерфейс) для доступа к ним.

HTML DOM определяет объекты и свойства всех HTML элементов и методы (интерфейс) для доступа к ним. Иначе говоря, HTML DOM описывает каким образом необходимо получать, изменять, добавлять и удалять HTMLэлементы.

В соответствии с моделью DOM все, что содержится внутри HTML документа - является узлом. То есть HTML документ представляется в виде дерева узлов, которыми являются элементы, атрибуты и текст.



Узлы дерева HTML документа

Согласно модели DOM:

- Весь документ представляется узлом документа;
- Каждый HTML тэг является узлом элемента;
- Текст внутри HTML элементов представляется текстовыми узлами;

– Каждому HTML атрибуту соответствует узел атрибута;
– Комментарии являются узлами комментариев.

```
<html>
  <head>
    <title>HTML документ</title>
  </head>
  <body>
    <h1>Заголовок </h1>
    <p>Просто текст</p>
  </body>
</html>
```

В этом примере корневым узлом является тэг <html>. Все остальные узлы содержатся внутри <html>. У этого узла имеется два дочерних узла: <head> и <body>. Узел <head> содержит узел <title>, а узел <body>содержит узлы <h1> и <p>.

Следует обратить особое внимание на то, что текст, расположенный в узле элемента соответствует текстовому узлу. В примере <title>HTML документ</title> узел элемента <title> содержит текстовый узел " HTML документ ", то есть " HTML документ " не является значением элемента <title>. Тем не менее, в рамках HTML DOM значение текстового узла может быть доступно посредством свойства innerHTML.

Все узлы HTML документа могут быть доступны посредством дерева, при этом их содержимое может быть изменено или удалено, а также можно добавить новые элементы.

Все узлы дерева находятся в *иерархических отношениях* между собой. Для описания этих отношений используются термины *родитель*, *дочерний элемент* и *потомок*. Родительские узлы имеют дочерние узлы, а дочерние элементы одного уровня называются потомками (братьями или сестрами).

В отношении узлов дерева соблюдаются следующие принципы:

– Самый верхний узел дерева называется корневым;
– Каждый узел, за исключением корневого, имеет ровно один родительский узел;
– Узел может иметь любое число дочерних узлов;
– Конечный узел дерева не имеет дочерних узлов;
– Потомки имеют общего родителя.

Программный интерфейс HTML DOM

В рамках DOM модели HTML можно рассматривать как множество узловых *объектов*. Доступ к ним осуществляется с помощью *JavaScript* или других языков программирования. Программный интерфейс DOM включает в себя набор стандартных *свойств* и *методов*.

Свойства представляют некоторые сущности (например, <h1>), а *методы* - действия над ними (например, добавить <a>).

К типичным свойствам DOM относятся следующие:

– x.innerHTML – внутреннее текстовое значение HTML элемента x ;
– x. nodeName – имя x ;
– x.nodeValue – значение x ;
– x.parentNode – родительский узел для x ;
– x.childNodes – дочерний узел для x ;
– x.attributes – узлы атрибутов x.

Узловой объект, соответствующий HTML элементу поддерживает следующие методы:

– x.getElementById(id) – получить элемент с указанным id ;
– x.getElementsByTagName(name) – получить все элементы с указанным именем тэга (name);

– x.appendChild(node) – вставить дочерний узел для x ;
– x.removeChild(node) – удалить дочерний узел для x.

Пример 3.

Для получения текста из элемента <p> со значением атрибута id "demo" в HTML документе можно использовать следующий код:

```
txt = document.getElementById("demo").innerHTML
```

Тот же самый результат может быть получен по-другому:

```
txt=document.getElementById("demo").childNodes[0].nodeValue
```

В рамках DOM возможны 3 способа доступа к узлам:

1. С помощью метода getElementById(ID). При этом возвращается элемент с указанным ID.
2. С помощью метода getElementsByTagName(name). При этом возвращаются все узлы с указанным именем тэга (в виде индексированного списка). Первый элемент в списке имеет нулевой индекс.
3. Путем перемещения по дереву с использованием отношений между узлами.

Для определения длины списка узлов используется свойство length.

```
x = document.getElementsByTagName("p");
for (i = 0; i < x.length; i++)
 {
  document.write(x[i].innerHTML);
  document.write("<br/>");
 }
```

В данном примере внутрь HTML документа вставляется в виде списка текстовое содержимое всех элементов соответствующих тэгу <p>.

Для навигации по дереву в ближайших окрестностях текущего узла можно использовать следующие свойства:

– parentNode ;
– *firstChild* ;
– lastChild.

Для непосредственного доступа к тэгам можно использовать 2 специальных свойства:

– document.documentElement – для доступа к корневому узлу документа;
– document.body – для доступа к тэгу <body>.

Свойства узлов

В HTML DOM каждый узел является объектом, который может иметь методы (функции) и свойства. Наиболее важными являются следующие свойства:

– nodeName ;
– nodeValue ;
– nodeType.

Свойство nodeName указывает на имя узла. Это свойство имеет следующие особенности:

1. Свойство nodeName предназначено только для чтения;
2. Свойство nodeName узла элемента точно соответствует имени тэга;
3. Свойство nodeName узла атрибута соответствует имени атрибута;
4. Свойство nodeName текстового узла всегда равно #text
5. Свойство nodeName узла документа всегда равно #document

Замечание: nodeName всегда содержит имя тэга HTML элемента в верхнем регистре.

Свойство nodeValue указывает на значение узла. Это свойство имеет следующие особенности:

- Свойство nodeValue узла элемента не определено;
- Свойство nodeValue текстового узла указывает на сам текст;
- Свойство nodeValue узла атрибута указывает на значение атрибута.

Свойство nodeType возвращает тип узла. Это свойство предназначено только для чтения:

Наиболее важными типами узлов являются следующие:

Тип элемента	Тип узла
Element	1
Attribute	2
Text	3
Comment	8
Document	9

Изменение HTML элементов

HTML элементы могут быть изменены с посредством использования JavaScript, HTML DOM и событий.

В примере показано, как можно динамически изменять текстовое содержимое тэга <p>:

```
<html>
  <body>
    <p id="p1">Hello World!</p>
    <script type="text/javascript">
      document.getElementById("p1").innerHTML="New text!";
    </script>
  </body>
</html>
```

Диалоговые элементы

В JavaScript поддерживается работа со следующими диалоговыми элементами интерфейса:

1. Alert. Применяется для уведомления пользователя, работающего с веб-браузером.

Синтаксис:

```
alert("сообщение");
```

2. Confirm. Применяется для выбора пользователем одного из двух вариантов ответа "Да/Нет". Соответственно Confirm возвращает значение true/false.

Синтаксис:

```
confirm("вопрос");
```

3. Prompt. Применяется для ввода пользователем значения. При нажатии "OK" возвращается введенное значение, в случае "Cancel" возвращается значение null.

Синтаксис:

```
prompt("вопрос/запрос","значение по умолчанию");
```

Ниже приводится код веб-страницы, в которой *пользователь* имеет возможность выбрать *цвет текста* с помощью диалогового элемента

```
<html>
<body>
// здесь будет отображаться текст
<div id="c" style="color:blue">Вы выбрали цвет текста: черный</div>

<script language="JavaScript">
// пользователь выбирает цвет текста
  var tcolor = prompt("Выберите цвет текста: red, blue, green, yellow, black","black");
// задается текст
  document.getElementById("c").innerHTML = "Вы выбрали цвет текста: " + tcolor;
// задается цвет текста
  document.getElementById("c").style.color = tcolor;

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

ХОД РАБОТЫ

1. Подготовьте на основе описанных выше примеров соответствующие веб-страницы и просмотрите их с помощью веб-браузера.

2. Контрольное задание. Создание таблицы случайно выбранных цветов

Взяв за основу *сценарий* построения таблицы умножения, постройте таблицу случайно выбранных цветов. Цвет ячейки таблицы задается с помощью атрибута bgcolor. Цвет ячейки описывается в рамках трехкомпонентной модели *RGB*, например: <td bgcolor="#c0a145">. Для генерации каждой компоненты можно использовать *генератор* случайных чисел с помощью методов объекта Math и преобразование в шестнадцатиричный формат:

```
color = Math.round(255.0*Math.random());
  r = color.toString(16);
```

Результирующий цвет образуется путем конкатенации компонентов:

```
color = r + g + b;
```

Примерный вид результата работы сценария:

Часть 2 Клиентские сценарии. Использование регулярных выражений

ТЕОРЕТИЧЕСКИЙ МАТЕРИАЛ

Обработка событий в JavaScript

Популярность JavaScript во многом обусловлена именно тем, что написанный на нем *сценарий* может реагировать на действия пользователя и другие внешние события. Каждое из событий связано с тем или иным объектом: формой, гипертекстовой ссылкой или даже с окном, содержащим текущий документ.

В качестве примеров внешних событий, на которые могут *реагировать объекты* JavaScript, можно привести следующие.

- окончание загрузки документа в окно (или окончание загрузки документов во все фреймы окна). Это событие связано с объектом window;
- щелчок мышью на объекте. Это событие может быть связано с интерактивным элементом формы или с гипертекстовой ссылкой;
- получение объектом фокуса ввода. Это событие может быть связано с объектами типа Text, Password и с другими интерактивными элементами;
- передача на сервер данных, введенных пользователем с помощью интерактивных элементов. Связывается с формой.

Обработка события производится с помощью специально предназначенного для этого фрагмента кода, называемого обработчиком события. Для каждого события JavaScript предоставляет свой обработчик. Однако при построении сценария можно создавать собственный обработчик события и использовать его вместо обработчика, заданного по умолчанию.

Имя обработчика определяет, какое событие он должен обрабатывать. Так, для того чтобы *сценарий* нужным образом отреагировал на щелчок мышью, используется обработчик с именем onClick, для обработки события, заключающегося в получении фокуса ввода, - обработчик onFocus.

Для того чтобы указать интерпретатору JavaScript на то, что обработкой события должен заниматься обработчик, необходимо включить в *HTML-дескриптор* следующее *выражение*:

имя_обработчика="команды_обработчика"

Это *выражение* включается в тэг, описывающий *объект*, с которым связано событие.

Например, если необходимо обработать событие, заключающееся в получении фокуса полем ввода, *дескриптор*, описывающий этот интерактивный элемент, должен иметь примерно следующий вид:

```
<input type="text" name="Inform" onFocus="handleFocus();">
```

Имя обработчика является одним из атрибутов *HTML*-дескриптора, а команды, предназначенные для обработки события, выступают в роли значения этого атрибута. В данном случае обработка события производится в теле функции handleFocus(). В принципе, обработчиком может быть не только *функция*, но и любая последовательность команд JavaScript в виде составного оператора.

Следующий пример демонстрирует обработку события, связанного с наведением курсора мыши на гиперссылку:

```
<a href = "http://www.myhp.edu"
  onmouseover="alert('An onMouseOver event'); return false">
  <img src="my.gif" width="50" height="50">
</a>
```

Ниже приводится полный текст *HTML* документа с JavaScript сценарием, в котором обрабатывается событие нажатия кнопки мыши, и определяется, какая именно из них была нажата:

```
<html>
 <head>
  <script language = "javascript">
   function whichButton(event)
    {
     if (event.button == 2)
      {
       alert("Вы щелкнули правой кнопкой мыши!");
      }
     else
      {
       alert("Вы щелкнули левой кнопкой мыши!");
      }
    }
  </script>
 </head>

 <body onmousedown="whichButton(event)">
  <p>Щелкните любой кнопкой мыши в любом месте документа</p>
 </body>
</html>
```

Таким образом, для того чтобы обработать какое-либо стандартное событие в браузере, необходимо добавить в подходящий *HTML* тег *атрибут*, соответствующий этому событию, указав в качестве значения атрибута имя JavaScript функции.

Атрибут	**Описание**
onabort	Прерванная загрузка изображения
onblur	утрата фокуса элементом
onchange	Изменение содержимого в поле ввода
onclick	Щелчок мыши на объекте
ondblclick	Двойной щелчок мыши на объекте
onerror	Ошибка при загрузке изображения или документа
onfocus	Получение фокуса элементом
onkeydown	Нажатие клавиши
onkeypress	Клавиша нажата
onkeyup	Отжатие клавиши
onload	Завершение загрузки страницы или изображения
onmousedown	Нажатие кнопки мыши
onmousemove	Перемещение курсора мыши
onmouseout	Смещение курсора мыши с объекта
onmouseover	Наведение курсора мыши на объект

onmouseup	Отжатие кнопки мыши
onreset	Кнопка "Reset" нажата
onresize	Изменение размера окна
onselect	Выделение текста
onsubmit	Кнопка "Submit" нажата
onunload	Уход с веб-страницы

Регулярные выражения

Регулярные выражения — система поиска текстовых фрагментов в электронных документах, основанная на специальной системе записи образцов для поиска.

Образец, задающий правило поиска, называется "шаблоном". Применение регулярных выражений принципиально преобразило технологии электронной обработки текстов.

С помощью регулярных выражений можно задавать структуру искомого шаблона и его позицию внутри строки (например, в начале или в конце строки, на границе или не на границе слова).

При описании структуры шаблона используются:

- гибкая система *квантификаторов* (операторов повторения);
- операторы описания наборов символов и их типа (числовые, нечисловые, специальные).

Для того, чтобы задать положение искомого фрагмента внутри строки, можно использовать один из следующих операторов:

Представление	**Позиция**
^	Начало строки
$	Конец строки
\b	Граница слова
\B	Не граница слова
(?=шаблон)	Искомая строка следует после указанной строкой (с просмотром вперед)
(?!=шаблон)	Искомая строка не следует после указанной строки (с просмотром вперед)
(?<=шаблон)	Искомая строка следует после указанной строкой (с просмотром назад)
(?<!=шаблон)	Искомая строка не следует после указанной строки (с просмотром назад)

Кроме того, язык регулярных выражений предоставляет набор *квантификаторов*, позволяющих указать число повторений шаблона:

Представление	**Число повторений**
{ *n* }	Ровно *n*
{ *m*, *n* }	От *m* до *n* включительно
{ *m*,}	Не менее *m*
{, *n* }	Не более *n*

Имеются и более простые *квантификаторы*:

Представление	**Число повторений**	**Эквивалент**
*	Ноль или более	{ *0,*}
+	Одно или более	{ *1,*}
?	Ноль или одно	{ *0, 1* }

Для задания внутри шаблона группы символов можно использовать следующие *операторы*:

Оператор	**Описание**
[xyz]	Любой символ из указанного множества
[^xyz]	Любой символ не входящий в указанное множество
[x-z]	Любой символ из указанного диапазона
[^x-z]	Любой символ не входящий в указанный диапазон
. (*точка*)	Любой символ кроме символов разрыва или переноса строки
\w	Любой буквенно-цифровой символ, включая символ подчеркивания
\W	Любой не буквенный символ
\d	Любая цифра
\D	Любой нецифровой символ
\s	Любой неотображаемый символ
\S	Любой символ (кроме неотображаемых символов)

Для группировки отдельных частей шаблона можно использовать следующие *операторы*:

Оператор	**Описание**
( )	Поиск группы символов внутри скобок и сохранение найденного соответствия
(?:)	Поиск группы символов внутри скобок без сохранения найденного соответствия
|	Комбинирование частей в одно выражения с последующим поиском любой из частей в отдельности. Аналогично оператору "ИЛИ".

Если *шаблон* поиска включает специальные (как правило неотображаемые) символы, для их описания можно использовать следующие обозначения:

Обозначение	**Описание**
\0	Символ с нулевым кодом
\n	Символ новой строки
\r	Символ начала строки
\t	Символ табуляции
\v	Символ вертикальной табуляции
\xxx	Символ, имеющий заданный восьмеричный ASCII код *xxx*
\xdd	Символ, имеющий заданный шестнадцатиричный ASCII код *dd*
\uxxxx	Символ, имеющий ASCII код выраженный ЮНИКОДОМ *xxxx*

Квантификаторам в регулярных выражениях соответствует максимально длинная строка из возможных (т.е. *квантификаторы* являются "жадными"). Это может приводить к некоторым проблемам. Например, *шаблон* (**<.*>**) описывающий на первый взгляд теги *HTML* на самом деле будет выделять более крупные фрагменты в документе.

Например, строка вида

```
<p>
 <font color='blue'>
 <i>Регулярные выражения<i>
 </font>
 - удобный инструмент для поиска в строках
</p>
```

формально соответствует указанному выше шаблону

Для решения данной проблемы можно использовать два подхода.

1. В регулярном выражении учитываются символы, не соответствующие желаемому образцу (например, <[^>]*> для вышеописанного случая).
2. Определение квантификатора как *нежадного* (ленивого) - большинство реализаций позволяют это сделать, добавив после него знак вопроса.

Например, по шаблону (<.*?>) будут найдены все теги из рассмотренной строки.

Таким образом, получаются следующие "нежадные" модификации квантфикаторов:

Модификатор	**Описание**
*?	"не жадный" эквивалент *
+?	"не жадный" эквивалент +
{n,}?	"не жадный" эквивалент {n,}

Следует, однако, иметь в виду, что использование "ленивых" *квантификаторов* может привести к ситуации, когда выражению соответствует слишком короткая, в частности, пустая строка.

Использование регулярных выражений в JavaScript

При поиске по тексту можно использовать *шаблон*, описывающий подстроку. В JavaScript такой *шаблон* может быть описан с помощью объекта *RegExp*. В простейшем случае такой *шаблон* описывает отдельный символ, однако имеет смысл его использовать для регулярных выражений.

Следующий ниже код описывает *RegExp объект* с именем pttn, содержащий регулярное *выражение*, описывающее целое десятичное число:

```
var pattn = new RegExp("/[0-9]+/");
```

Объект RegExp имеет 3 встроенных метода: test(), exec() и compile().

- Метод test() выполняет поиск по шаблону:

```
• var pattn = new RegExp("[0-9]+");
document.write(pattn.test("38 попугаев"));
```

Результат:

```
true
```

- Метод exec() выполняет поиск подстроки по шаблону и возвращает найденные соответствия; если соответствий нет, возвращается значение null:

```
• var pattn=new RegExp("[0-9]+");
document.write(pattn.exec("38 попугаев"));
```

Результат:

```
38
```

Если необходимо найти все соответствия, то при вызове конструктора *RegExp* следует указать дополнительный параметр "g", указывающий на необходимость глобального поиска:

```
var pattn = new RegExp("[0-9]+", "g");
do
{
result  = pattn.exec("1 попугай, 2 попугая,…, 38 попугаев");
document.write(" " + result);
}
while (result != null)
```

Пример 7.2.

Результат:

```
1 2 38 null
```

- Метод compile() применяется для изменения ранее созданного шаблона:

```
• var pattn = new RegExp("[0-5]+");
• document.write(pattn.exec("38 попугаев"));
• pattn.compile("[6-9]+");
document.write(";" + pattn.exec("38 попугаев"));
```

Пример 7.3.

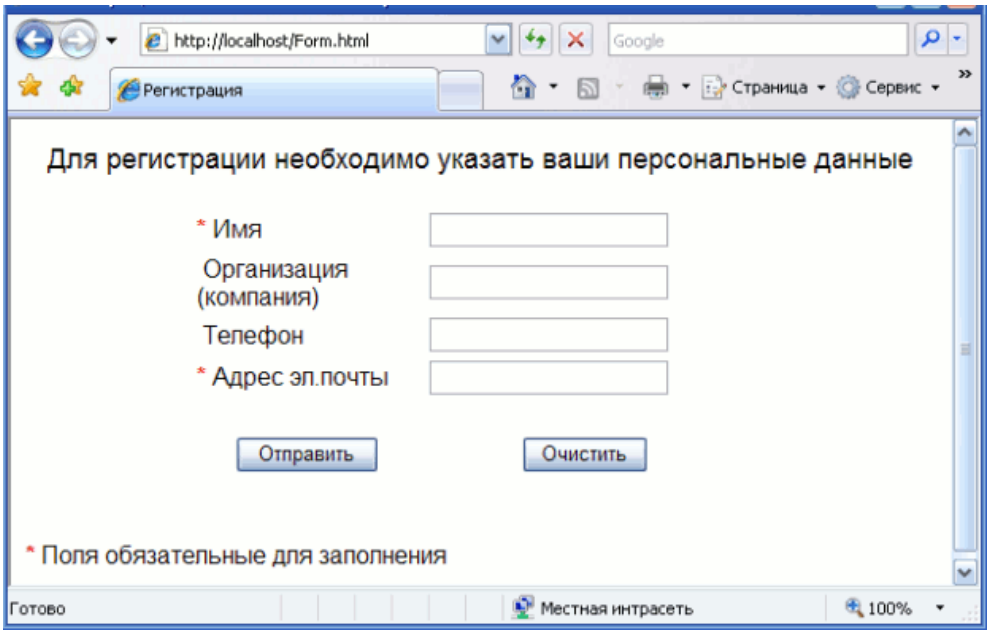
Результат:

```
3;8
```

ХОД РАБОТЫ

Проверка значений, введенных пользователем в поля формы для регистрации.

- Для выполнения лабораторной работы необходимо создать веб-страницу, содержащую форму с полями, следующего вида:



- В тэге <form> добавьте обработчик события отправки данных вида:

onSubmit = "CheckData(); return false;"

В данном случае указана функция обработчик CheckData(). Оператор return false; предотвращает автоматическую отправку данных после выполнения функции-обработчика. Отправка данных будет выполняться из обработчика.

- Добавьте на странице секцию JavaScript кода, описывающего функцию-обработчик:

```
function CheckData()
{
  var ans;
  ans = confirm("Вы уверены, что хотите отправить введенные данные ?");
  if (ans) submit();
}
```

Как это видно из кода, функция CheckData() в случае подтверждения со стороны пользователя самостоятельно вызывает метод submit() для передачи данных из формы.

- Теперь необходимо добавить проверку значений, введенных в поля формы пользователем.

Прежде всего, необходимо убедиться в том, что заполнены все поля, обязательные для ввода. Для этого можно использовать проверку на равенство нулю длины строки (свойство length), являющейся значением узла дерева документа, соответствующего полю ввода, например:

document.getElementById("uname").value.length.

Следующая проверка должна контролировать структуру и содержимое полей. Для этого можно использовать объект *RegExp*, например:

```
var validEMail, pattn;

pattn = new RegExp("^[\.\-_A-Za-z0-9]+?@[\.\-A-Za-z0-9]+?\.[A-Za-z0-9]{2,6}$");

validEMail = pattn.test(document.getElementById("email").value));
```

В данном фрагменте описана проверка структуры электронного адреса из поля формы с идентификатором "email". Для проверки был использован шаблон на основе регулярного выражения.

Контрольное задание

Самостоятельно постройте регулярное *выражение*, описывающее *шаблон* для проверки номера телефона, и внесите все необходимые изменения и дополнения в функцию CheckData().

Часть 3 Создание сценариев с применением функций, объектов и массивов Java Script

ХОД РАБОТЫ

Выполнить задания из практических занятий 6 и 7 с помощью команд Java Script.