Министерство образования Республики Беларусь

Учреждение образования

«Белорусский государственный университет информатики и радиоэлектроники»

Филиал

«Минский радиотехнический колледж»

Учебная дисциплина «Веб-программирование на стороне сервера»

**Инструкция**

по выполнению лабораторной работы

«Создание изображений средствами PHP»

Минск

2020

**Лабораторная работа № 4**

**Тема работы: «Создание изображений средствами PHP»**

**1. Цель работы**

Формирование навыков по использованию основных команд для создания и редактирования изображений средствами PHP.

**2. Задание**

Создать диаграмму, используя PHP-средства.

**3. Оснащение работы**

ПК, веб-сервер, интерпретатор РНР, текстовый редактор, браузер.

**4. Основные теоретические сведения**

PHP приобрел всемирную популярность не только как удобное средство вывода HTML и обработки форм, но и, во многом, благодаря набору расширений, позволяющих выполнять всевозможные стоящие перед веб-разработчиком задачи. Одним из таких расширений является библиотека GD, предназначенная для работы с растровыми изображениями.

Для успешного выполнения примеров необходимо, чтобы PHP был собран с поддержкой второй версии библиотеки GD.

Для того, чтобы узнать наличие и используемую версию графической библиотеки, воспользуемся функцией [phpinfo()](http://www.php.net/phpinfo), и взглянем на раздел "GD", информация о котором представлена в таблице 1.

Таблица 1 – Информационный раздел

|  |  |
| --- | --- |
| **GD Support** | enabled |
| **GD Version** | bundled (2.0.15 compatible) |
| **FreeType Support** | enabled |
| **FreeType Linkage** | with freetype |
| **GIF Read Support** | enabled |
| **JPG Support** | enabled |
| **PNG Support** | enabled |
| **WBMP Support** | enabled |
| **XBM Support** | enabled |
| **JIS-mapped Japanese Font Support** | enabled |

Если ничего похожего вы не нашли — значит, PHP собран без поддержки графической библиотеки, и придется его пересобирать.

В противном случае, сначала взглянем на строку **"GD Version"**. Вторая версия может быть разная. Обратите внимание на слово **"bundled"**. Его присуствие означает, что PHP собран с библиотекой GD, поставляемой вместе с PHP: именно эту библиотеку рекомендуется использовать, так как разработчики PHP исправили множество ошибок и недочетов, присущих оригинальной GD. Впрочем, если используется внешняя библиотека (слово **"bundled"** в строке "GD Version" отсутствует) — ничего страшного, все приведенные ниже примеры будут работать.

В нижеследующих строках мы видим, какие графические форматы поддерживаются данной конфигурацией библиотеки GD. В частности, работа с форматом **GIF** поддерживается в режиме "только чтение" из-за проблем с лицензированием используемого в GIF алгоритма компрессии.

Итак, если требуется пересборка PHP, следует это и сделать, загрузив при необходимости последнюю версию PHP, и указать при сборке параметр:

--with-gd[=location]

Необязательный параметр location указывает путь к внешней библиотеке GD. Если его опустить (в смысле, не указывать) — используется bundled-версия.

Для поддержки работы с шрифтами TrueType, также понадобится наличие библиотеки [FreeType](http://www.freetype.org/) и следующие параметры конфигурационной строки:

--enable-gd-native-ttf

--with-freetype-dir=*/путь/к/библиотеке/FreeType*

#### Примечание:

Для сборки GD с поддержкой GIF Write под FreeBSD, необходимо предварительно установить переменную среды WITH\_LZW:

export WITH\_LZW=yes

#### Примечание:

Выше подразумевалось, что используется Unix-подобная ОС. Если же PHP установлен локально, да еще и под Windows — то надо просто раскомментировать одну строчку в php.ini.

### ***Генерация изображения с помощью PHP***

Теперь нарисуем и выведем в браузер какую-нибудь картинку с помощью PHP. Нельзя вот так вот просто взять и вывести картинку посреди HTML-кода в том же самом скрипте: в HTML-документе мы разместим, как и обычно, тэг <img>, а в его атрибуте src укажем не картинку, как обычно, а PHP-скрипт:

<img src="/i/articles/image/image.php">

Теперь приступим к написанию этого самого image.php.

#### Заголовок. Он же header.

Прежде всего, как браузер узнает, что image.php - это не HTML-документ и не что-то еще, а картинка?

Тип документа браузер определяет по заголовку Content-type. На самом деле, этот заголовок — обязательный и всегда присутствует; по умолчанию, PHP услужливо "отдает" заголовок Content-type: text/html. Обычно это поведение PHP нам прекрасно подходит - но не в данном случае. Придется нам вывести нужный заголовок самим:

<?php

header('Content-type: image/png'); // устанавливаем тип документа - "изображение в формате PNG".

?>

Приступим теперь непосредственно к генерации картинки.

#### Создание изображения

Для создания изображения, в нашем распоряжении две функции:

**1.** [imagecreate()](http://www.php.net/imagecreate). С помощью этой функции можно создать изображение на основе палитры, содержащей фиксированный набор цветов. Каждый цвет палитры необходимо описать с помощью функции [imagecolorallocate()](http://www.php.net/imagecolorallocate). Этот способ создания изображения был единственным при работе с первой версией GD, и необходим при работе с ориентированными на палитру форматами, такими как GIF. Однако введенный во второй версии (и отныне рекомендуемый разработчиками) способ, на мой взгляд, гораздо более удобен.

**2.** [imagecreatetruecolor()](http://www.php.net/imagecreatetruecolor). Эта функция создает TrueColor-изображение, то есть цвет каждой точки определяется произвольным цветом, задаваемым в координатах RGB. Помимо того, что это удобнее, чем работа с палитрой, такой подход позволяет производить масштабирование изображения с гораздо меньшими потерями качества. Но об этом позже. Сейчас просто создадим изображение:

<?php

$image = imagecreatetruecolor(80,60) // создаем изображение...

or die('Cannot create image'); // ...или прерываем работу скрипта в случае ошибки

imagedestroy($image); // освобождаем память, выделенную для изображения

?>

Функция [imagecreatetruecolor()](http://www.php.net/imagecreatetruecolor) (как, кстати, и функция [imagecreate()](http://www.php.net/imagecreate)), принимает два обязательных целочисленных параметра — ширину (в нашем примере — 80 пикселей) и высоту (60 пикселей) картинки, и возвращает идентификатор ресурса (в данном случае — изображения), который мы присваиваем переменной $image, которой в дальнейшем будем постоянно пользоваться.

Если по какой-либо страшной причине (недостатке памяти, например) создать изображение не получается, функция возвращает false — в этом случае мы прерываем работу скрипта.

Хотя мы с этим изображением, пока что, ничего не делаем, мы обязаны освободить выделенную для него память с помощью функции [imagedestroy()](http://www.php.net/imagedestroy): к сожалению, автоматическое освобождение памяти происходит не всегда, и игнорирование рекомендации всегда использовать imagedestroy может привести к весьма серьезным утечкам памяти.

Итак, с помощью функции [imagecreatetruecolor()](http://www.php.net/imagecreatetruecolor) мы создали "труколорное" и, если верить документации, черное изображение размером 80x60. Так что, те, в чьи планы входит создание Web-галереи репродукций "квадрата Малевича", могут смело переходить к следующей главе. :) Для остальных, продолжим. Нарисуем на картинке что-нибудь содержательное.

<?php

$image = imagecreatetruecolor(80,60) // создаем изображение...

or die('Cannot create image'); // ...или прерываем работу скрипта в случае ошибки

// "Зальем" фон картинки синим цветом...

imagefill($image, 0, 0, 0x000080);

// Нарисуем желтый контурный эллипс...

imageellipse($image, 40, 30, 50, 50, 0xFFFF00);

// ...и еще пару, но сплошных...

imagefilledellipse($image, 30, 20, 10, 10, 0xFFFF00);

imagefilledellipse($image, 50, 20, 10, 10, 0xFFFF00);

// ...вертикальную линию...

imageline($image, 40, 28, 40, 38, 0xFFFF00);

// ...и дугу.

imagearc($image, 40, 30, 40, 40, 45, 135, 0xFFFF00);

// Устанавливаем тип документа - "изображение в формате PNG"...

header('Content-type: image/png');

// ...И, наконец, выведем сгенерированную картинку в формате PNG:

imagepng($image);

imagedestroy($image); // освобождаем память, выделенную для изображения

?>

Результат представлен на рисунке 1.

http://webmasterschool.ru/articles/images/article50/example1.png

Рисунок 1 – Результат работы скрипта

Используемые для рисования функции весьма просты для понимания. Просто "поиграйтесь", меняя значения параметров — как вы, несомненно, уже догадались, это ни что иное, как координаты, и цвета в виде 0xRRGGBB.

Подробнее же мы разберем вот эту строку:

imagepng($image);

Выше мы создали в памяти изображение, и всячески над ним извращались. :) Это мы делали без привязки к какому-либо выходному формату — просто работали с набором байтов в памяти. А вот функции вида image*формат* ([imagepng()](http://www.php.net/imagepng), [imagejpeg()](http://www.php.net/imagejpeg), [imagewbmp()](http://www.php.net/imagewbmp)...) генерируют на основе этого самого набора байтов, на который ссылается идентификатор $image, картинку в соответствующем формате, и выводят ее в выходной поток — то бишь, проще говоря, в браузер.

Нелишне здесь вспомнить и о строке header('Content-type: image/png'), где мы указали тип документа - обратите внимание: здесь png, и там png. :) Справедливости ради, надо заметить, что большинство браузеров воспринимают только часть "image" этого заголовка, а формат самой картинки уже определяют по ее заголовкам, характерным для каждого формата; однако, лучше все же не надеяться на "интеллект" браузера и указывать правильный формат изображения.

Используя полученные в результате рисования смайликов знания, попробуем порисовать диаграммы, а заодно и научимся выводить на картинках текст (не забудьте только о библиотеке FreeType).

Предположим, вы располагаете следующей важной статистической информацией о коллекционерах почтовых марок:

<?php

// Назовем этот файл data.php - он нам еще понадобится.

$Title = 'Количество марок у моих друзей';

$Data = array(

'Коля' => 16,

'Петя' => 14,

'Федя' => 11,

'Маша' => 17,

'Ипполит' => 8

);

?>

Хотите представить эту информацию в виде "столбиков" — для удобного сравнения.

<?php

define('GRAPH\_WIDTH', 400); // ширина картинки

define('GRAPH\_HEIGHT', 300); // высота картинки

define('GRAPH\_OFFSET\_TOP', 40); // отступ сверху

define('GRAPH\_OFFSET\_LEFT', 40); // отступ слева

define('GRAPH\_OFFSET\_RIGHT', 5); // отстут справа

define('GRAPH\_OFFSET\_BOTTOM', 30); // отступ снизу

$colors = array(0xFF0000,0x00FF00,0x0000FF, // цвета столбцов

0xFFFF00,0x00FFFF,0xFF00FF);

require('data.php'); // Вот и понадобился data.php :)

// Считаем ширину столбцов

$col\_width = (GRAPH\_WIDTH - GRAPH\_OFFSET\_LEFT - GRAPH\_OFFSET\_RIGHT) / count($Data);

// Считаем высоту столбца, соответствующего максимальному значению

$col\_maxheight = (GRAPH\_HEIGHT - GRAPH\_OFFSET\_TOP - GRAPH\_OFFSET\_BOTTOM);

// Ищем максимальное значение в массиве, соответствующее столбцу макс высоты

$max\_value = max($Data);

$image = imagecreatetruecolor(GRAPH\_WIDTH,GRAPH\_HEIGHT) // создаем изображение...

or die('Cannot create image'); // ...или прерываем работу скрипта в случае ошибки

imagefill($image, 0, 0, 0xFFFFFF); // белый фон

// рисуем столбцы

$x = GRAPH\_OFFSET\_LEFT;

$y = GRAPH\_OFFSET\_TOP + $col\_maxheight;

$i = 0;

foreach($Data as $value) {

imagefilledrectangle( // рисуем сплошной прямоугольник

$image,

$x,

$y - round($value\*$col\_maxheight/$max\_value),

$x + $col\_width - 1,

$y,

$colors[$i++%count($colors)]

);

$x += $col\_width;

}

// рисуем координатную ось

imageline($image, GRAPH\_OFFSET\_LEFT - 5, GRAPH\_OFFSET\_TOP,

GRAPH\_OFFSET\_LEFT - 5, $y, 0xCCCCCC);

for($value=0; $value<=$max\_value; $value++) {

imageline($image, GRAPH\_OFFSET\_LEFT - 7, $Y = $y - round($value\*$col\_maxheight/$max\_value),

GRAPH\_OFFSET\_LEFT - 5, $Y, 0xCCCCCC);

imagestring($image, 1, GRAPH\_OFFSET\_LEFT / 2, $Y - 4, $value, 0x000000);

}

// Устанавливаем тип документа - "изображение в формате PNG"...

header('Content-type: image/png');

// ...И, наконец, выведем сгенерированную картинку в формате PNG:

imagepng($image);

imagedestroy($image); // освобождаем память, выделенную для изображения

?>

Результат представлен на рисунке 2.

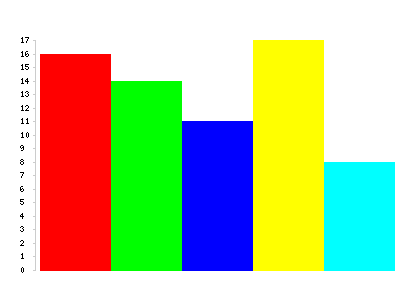


Рисунок 2 – Результат работы скрипта

Обратите внимание на строку

imagestring($image, 1, GRAPH\_OFFSET\_LEFT / 2, $Y - 4, $value, 0x000000);

с помощью которой мы выводим числа на координатной оси. Второй параметр - один из встроенных в GD шрифтов (от 1 до 5, чем больше число — тем крупнее шрифт).

Так зачем же, скажете вы, нам какие-то там TrueType-шрифты и FreeType-библиотеки, если мы и так прекрасно пишем на картинке? А вот затем, скажу я вам, что писать-то мы хотим по-русски, а встроенные шрифты о существовании кириллицы даже и не подозревают. А нам надо бы подписать столбики именно по-русски. Да и выбор встроенных шрифтов невелик.

Итак, нам понадобится:

* Функция [imagettftext()](http://www.php.net/imagettftext), которая рисует выбранным TrueType-шрифтом на картинке,
* Какой-нибудь кириллический TrueType-шрифт. Возьмем, например, arial.ttf из всеми нами любимой Винды, да не просто возьмем, а положим его туда, где лежит наш скрипт,
* Поскольку функция [imagettftext()](http://www.php.net/imagettftext) воспринимает кодировку Unicode, но никак не Windows-1251, то нам пригодится вот такая функция для соответствующего преобразования:

<?php // Разместим этот код в файле win2uni.php...

// Преобразование Windows 1251 -> Unicode

function win2uni($s)

{

$s = convert\_cyr\_string($s,'w','i'); // преобразование win1251 -> iso8859-5

// преобразование iso8859-5 -> unicode:

for ($result='', $i=0; $i<strlen($s); $i++) {

$charcode = ord($s[$i]);

$result .= ($charcode>175)?"&#".(1040+($charcode-176)).";":$s[$i];

}

return $result;

}

?>

* Функция [imagettfbbox()](http://www.php.net/imagettfbbox), которая поможет нам вычислить высоту и ширину выводимого шрифтом текста.

Сначала потренируемся:

<?php

require('win2uni.php');

define('WIDTH', 200);

define('HEIGHT', 60);

define('FONT\_NAME', 'arial.ttf');

define('FONT\_SIZE', 20);

$image = imagecreatetruecolor(WIDTH,HEIGHT)

or die('Cannot create image');

// Не забываем преобразовать текст в кодировку Unicode

$text = win2uni('Всем привет! :)');

$coord = imagettfbbox(

FONT\_SIZE, // размер шрифта

0, // угол наклона шрифта (0 = не наклоняем)

FONT\_NAME, // имя шрифта, а если точнее, ttf-файла

$text // собственно, текст

);

/\* Функция imagettfbbox возвращает нам массив из восьми элементов

содержащий всевозможные координаты минимального прямоугольника,

в который можно вписать данный текст. Индексы массива  
 удобно обозначить на схеме в виде координат (x,y):

(6,7) (4,5)

+---------------+

|Всем привет! :)|

+---------------+

(0,1) (2,3)

Число элементов массива может на первый взгляд показаться избыточным,

но не следует забывать о возможности вывода текста под произвольным

углом.

По этой схеме легко вычислить ширину и высоту текста:

\*/

$width = $coord[2] - $coord[0];

$height = $coord[1] - $coord[7];

// Зная ширину и высоту изображения, располагаем текст по центру:

$X = (WIDTH - $width) / 2;

$Y = (HEIGHT + $height) / 2;

imagettftext(

$image, // как всегда, идентификатор ресурса

FONT\_SIZE, // размер шрифта

0, // угол наклона шрифта

$X, $Y, // координаты (x,y), соответствующие левому нижнему

// углу первого символа

0xFFFFFF, // цвет шрифта

FONT\_NAME, // имя ttf-файла

$text

);

header('Content-type: image/png');

imagepng($image);

imagedestroy($image);

?>

Результат представлен на рисунке 3.

http://webmasterschool.ru/articles/images/article50/example3.png

Рисунок 3 – Результат работы скрипта

Тренировка прошла успешно — всех поприветствовали. Можно теперь приступать к нашим диаграммам.

<?php

define('GRAPH\_WIDTH', 400); // ширина картинки

define('GRAPH\_HEIGHT', 300); // высота картинки

define('GRAPH\_OFFSET\_TOP', 40); // отступ сверху

define('GRAPH\_OFFSET\_LEFT', 40); // отступ слева

define('GRAPH\_OFFSET\_RIGHT', 5); // отстут справа

define('GRAPH\_OFFSET\_BOTTOM', 30); // отступ снизу

define('FONT\_NAME', 'arial.ttf'); // Имя шрифта

define('FONT\_SIZE', 12); // Размер шрифта

$colors = array(0xFF0000,0x00FF00,0x0000FF, // цвета столбцов

0xFFFF00,0x00FFFF,0xFF00FF);

require('data.php');

require('win2uni.php');

// Считаем ширину столбцов

$col\_width = (GRAPH\_WIDTH - GRAPH\_OFFSET\_LEFT - GRAPH\_OFFSET\_RIGHT) / count($Data);

// Считаем высоту столбца, соответствующего максимальному значению

$col\_maxheight = (GRAPH\_HEIGHT - GRAPH\_OFFSET\_TOP –

GRAPH\_OFFSET\_BOTTOM);

// Ищем максимальное значение в массиве, соответствующее столбцу максимальной высоты

$max\_value = max($Data);

$image = imagecreatetruecolor(GRAPH\_WIDTH,GRAPH\_HEIGHT) // создаем изображение...

or die('Cannot create image'); // ...или прерываем работу скрипта в случае ошибки

imagefill($image, 0, 0, 0xFFFFFF); // белый фон

// рисуем столбцы

$x = GRAPH\_OFFSET\_LEFT;

$y = GRAPH\_OFFSET\_TOP + $col\_maxheight;

$i = 0;

foreach($Data as $name => $value) {

imagefilledrectangle( // рисуем сплошной прямоугольник

$image,

$x,

$y - round($value\*$col\_maxheight/$max\_value),

$x + $col\_width - 1,

$y,

$colors[$i++%count($colors)]

);

// Выводим текст:

// .. преобразование в Unicode...

$text = win2uni($name);

// .. расчет координат...

$coord = imagettfbbox(FONT\_SIZE,0,FONT\_NAME,$text);

$text\_x = $x + ($col\_width - $coord[2] - $coord[0]) / 2;

$text\_y = GRAPH\_HEIGHT - 5;

// .. и вывод текста

imagettftext($image,FONT\_SIZE,0,$text\_x,$text\_y,0x000000,FONT\_NAME,$text);

$x += $col\_width;

}

// Выводим заголовок

$text = win2uni($Title);

$coord = imagettfbbox(FONT\_SIZE,0,FONT\_NAME,$text);

$text\_x = $x + ($col\_width - $coord[2] - $coord[0]) / 2;

$text\_y = (GRAPH\_OFFSET\_TOP - $coord[1] - $coord[7]) / 2;

imagettftext($image,FONT\_SIZE,0,$text\_x,$text\_y,0x000000,FONT\_NAME,$text);

// рисуем координатную ось

imageline($image, GRAPH\_OFFSET\_LEFT - 5, GRAPH\_OFFSET\_TOP,

GRAPH\_OFFSET\_LEFT - 5, $y, 0xCCCCCC);

for($value=0; $value<=$max\_value; $value++) {

imageline($image, GRAPH\_OFFSET\_LEFT - 7, $Y = $y –

round($value\*$col\_maxheight/$max\_value),

GRAPH\_OFFSET\_LEFT - 5, $Y, 0xCCCCCC);

imagestring($image, 1, GRAPH\_OFFSET\_LEFT / 2, $Y - 4, $value, 0x000000);

}

header('Content-type: image/png');

imagepng($image);

imagedestroy($image);

?>

Запускаем... Результат представлен на рисунке 4.

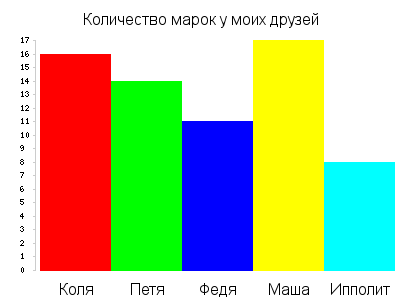


Рисунок 4 – Результат работы скрипта

### Изменение размера: thumbnails, или "превьюшки"

Часто перед веб-разработчиком стоит задача генерации уменьшенных копий изображений для предварительного просмотра; можно даже сказать, что это — одно из самых распространенных применений библиотеки GD.

В GD1, поддерживающей только изображения на основе палитры, присуствовала лишь функция [imagecopyresized()](http://www.php.net/imagecopyresized); качество уменьшенных изображений, генерируемых этой ей, мягко говоря, оставляло желать лучшего: работая с фиксированной палитрой, ограниченной 255 цветами, весьма затруднительно обеспечить качественный антиалиасинг.

Во второй версии библиотеки, с появлением поддержки TrueColor и [imagecreatetruecolor()](http://www.php.net/imagecreatetruecolor), введена новая функция — [imagecopyresampled()](http://www.php.net/imagecopyresampled), обеспечивающая весьма достойное качество "превьюшек".

Продемонстрируем работу с этой функцией. Предположим, у нас есть файл original.jpg, допустим, 400x250 пикселей, и мы хотим создать ее уменьшенный вариант small.jpg — 100x60. Можно поступить так:

<?php

define('SOURCE', 'original.jpg'); // исходный файл

define('TARGET', 'small.jpg'); // имя файла для "превьюшки"

define('NEWX', 100); // ширина "превьюшки"

define('NEWY', 60); // высота "превьюшки"

// Определяем размер изображения с помощью функции getimagesize:

$size = getimagesize(SOURCE);

// Функция getimagesize, требуя в качестве своего параметра имя файла,

// возвращает массив, содержащий (помимо прочего, о чем можно прочитать

// в документации), ширину - $size[0] - и высоту - $size[1] –

// указанного изображения. Кстати, для ее использования не требуется наличие

// библиотеки GD, так как она работает непосредственно с заголовками

// графических файлов. В случае, если формат файла не распознан, getimagesize

// возвращает false:

if ($size === false) die ('Bad image file!');

// Читаем в память JPEG-файл с помощью функции imagecreatefromjpeg:

$source = imagecreatefromjpeg(SOURCE)

or die('Cannot load original JPEG');

// Создаем новое изображение

$target = imagecreatetruecolor(NEWX, NEWY);

// Копируем существующее изображение в новое с изменением размера:

imagecopyresampled(

$target, // Идентификатор нового изображения

$source, // Идентификатор исходного изображения

0,0, // Координаты (x,y) верхнего левого угла

// в новом изображении

0,0, // Координаты (x,y) верхнего левого угла копируемого

// блока существующего изображения

NEWX, // Новая ширина копируемого блока

NEWY, // Новая высота копируемого блока

$size[0], // Ширина исходного копируемого блока

$size[1] // Высота исходного копируемого блока

);

// Сохраняем результат в JPEG-файле:

// Функции генерации графических файлов, такие как imagejpeg,

// могут выводить результат своей работы не только в броузер,

// но и в файл. Для этого следует указать имя файла в необязательном

// втором параметре.

// Именно функция imagejpeg имеет и третий необязательный параметр –

// качество изображения. Установим максимальное качество - 100.

imagejpeg($target, TARGET, 100);

// Как всегда, не забываем:

imagedestroy($target);

imagedestroy($source);

?>

Этот код работает, однако искажения, получаемые при непропорциональном изменении размера, выглядят не особенно симпатично. Более того, код получился не особенно-то универсальным: мало того, что мы можем работать только с JPEG-файлами, у нас еще и жестко заданы имена файлов и размеры получаемого изображения.

Итак, пусть у нас есть файл в любом поддерживаемом GD формате, и мы хотим создать "превьюшку" заданного размера в формате JPEG. Разработаем для этой цели функцию imgResize. Комментариев здесь почти не будет — так как используются уже изученные приемы и обычная арифметика. Постарайтесь разобраться в этом коде самостоятельно.

<?php

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Функция img\_resize(): генерация thumbnails

Параметры:

$src - имя исходного файла

$dest - имя генерируемого файла

$width, $height - ширина и высота генерируемого изображения, в пикселях

Необязательные параметры:

$rgb - цвет фона, по умолчанию – белый

$quality - качество генерируемого JPEG, по умолчанию - максимальное (100)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

function img\_resize($src, $dest, $width, $height, $rgb=0xFFFFFF, $quality=100)

{

if (!file\_exists($src)) return false;

$size = getimagesize($src);

if ($size === false) return false;

// Определяем исходный формат по MIME-информации, предоставленной

// функцией getimagesize, и выбираем соответствующую формату

// imagecreatefrom-функцию.

$format = strtolower(substr($size['mime'], strpos($size['mime'], '/')+1));

$icfunc = "imagecreatefrom" . $format;

if (!function\_exists($icfunc)) return false;

$x\_ratio = $width / $size[0];

$y\_ratio = $height / $size[1];

$ratio = min($x\_ratio, $y\_ratio)

$use\_x\_ratio = ($x\_ratio == $ratio);

$new\_width = $use\_x\_ratio ? $width : floor($size[0] \* $ratio);

$new\_height = !$use\_x\_ratio ? $height : floor($size[1] \* $ratio);

$new\_left = $use\_x\_ratio ? 0 : floor(($width - $new\_width) / 2);

$new\_top = !$use\_x\_ratio ? 0 : floor(($height - $new\_height) / 2);

$isrc = $icfunc($src);

$idest = imagecreatetruecolor($width, $height);

imagefill($idest, 0, 0, $rgb);

imagecopyresampled($idest, $isrc, $new\_left, $new\_top, 0, 0,

$new\_width, $new\_height, $size[0], $size[1]);

imagejpeg($idest, $dest, $quality);

imagedestroy($isrc);

imagedestroy($idest);

return true;

}

?>

Разобрались? Поместим этот код в файл imgresize.php.

Пример использования функции img\_resize:

<?php

require ('imgresize.php');

if (img\_resize('original.jpg', 'small.jpg', 100, 60))

echo 'Image resized OK';

else

echo 'Resize failed!';

?>

**5. Порядок выполнения работы**

Используя PHP-средства, создайте диаграмму вашего среднего балла по предметам.

**6. Форма отчета о работе**

*Лабораторная работа № \_\_\_*

*Номер учебной группы \_\_\_\_\_\_\_\_\_\_\_\_*

*Фамилия, инициалы учащегося \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Дата выполнения работы \_\_\_\_\_\_\_\_\_\_\_\_\_*

*Тема работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Цель работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Оснащение работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Результат выполнения работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

**7. Контрольные вопросы и задания**

1. Что такое массив в РНР?

2. Способы получения данных в РНР.

**8. Рекомендуемая литература**

**Колисничеко, Д.Н.**  Самоучитель РНР5 / Д.Н. Колисниченко. – Спб.:Наука и техника, 2004.

**Прохоренок, Н.А**. HTML, JavaScript, PHP и MySQL. Джентльменский набор Web-мастера / Н.А.Прохоренок. – СПб.: БХВ-Петербург, 2010.

**Никсон, Р.** Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5 / Р. Никсон. — СПб.: Питер, 2016.