PHP

<?php ?>

Full Guide

Bryan Ravensky

Contents:

Welcome to the world of web programming! During this course I will try to teach you, and you, in turn, will try to learn to create various Web applications, from elementary examples, to full-function products.

At once I will notice that I write, calculating that you know bases of a markup language of HTML and you have at least brief experience of programming. Otherwise... well you understood.

But, before to begin studying directly of the PHP language, let's understand that such web programming.

I. What is the web programming. Main technologies

I-1. Client-server

If you already tried (and can be, even it is not unsuccessful) to program, for example, on Delphi, either Visual Basic, or even the Visual C ++, then got used to such scheme of a program runtime: the button is clicked - the code is carried out - the result is displayed, and all this is carried out on one computer.

In web programming everything is in a different way.

You thought what occurs when you enter in an address bar of the URL browser (Universal Resource Location, or in a popular speech - the address)? Scheme of work following:

1. The browser opens connection with the server

2. The browser sends the server a request for obtaining the page

3. The server creates the answer (most often - a HTML code) to the browser and closes connection

4. The browser processes a HTML code and displays the page

Pay attention on highlighted in bold. Still before you saw the requested page on the screen, connection with the server is closed, and he forgot about you. And when you will enter another (or the same) the address, or click on the link, or you will press the HTML forms button - the same scheme will repeat again.

Such scheme of work call "client-server". The client in this case - the browser.

So, connection with the Web server lasts only several seconds (or a share of seconds) is a period between click on the link (or in a different way a request) and the beginning of page display. The majority of browsers during connection time display a certain indicator, for example, MS Internet Explorer displays animation in the upper right corner.

The attentive reader can notice here - and how so, I already read the page, and the indicator still shows connection process? The matter is that tag <img src=...> (loading of the image) and some other are no more than one more server request - and it is carried out just as also any other - according to the same scheme. And the picture request, from the point of view of the server, is completely independent of a HTML nickname request.

Forever to get rid of HTTP perception as "a black box", "we will pretend to be" the browser by means of telnet:

1. Let's start telnet www.php5.com 80

2. Let's enter in a terminal window the following (if input is not displayed - nothing terrible):

GET/HTTP/1.0 [we will click Enter here]

Host: www.php5.com [we will click Enter twice here]

Clicking of Enter corresponds, as a rule, to a combination of characters of CR + LF designated as \by r\n. This designation will be used further.

On the screen will run a http://www.php5.com/ page HTML code. As you can see - anything difficult.

The source code of the current page can be browsed practically in any browser, having selected in the View|Source menu.

Pictures, frames - all this additional requests, just the same. Actually, from where pictures in a browser window undertake: when parsing (processing) of a HTML code, the browser, stumble on a tag <img src= "picture"> makes additional server request - a picture request, and displays it on site where there is a tag <img...>.

Try:

telnet www.php5.com 80

GETphp/php5com.png HTTP/1.0\r\n

Host: www.php5.com\r\n\r\n

On the screen will run what you will see if you browse this png-file in a text editor.

I-2. HTML forms. Methods of sending data for the server

You for certain already met HTML forms:

1. <form method= "GET" action= "/cgi-bin/form_handler.cgi">
2. Enter your name: <input type= "text" name= "name">
3.
4. <input type= "submit" of name= "okbutton" value= OK>
5. </form>

Having saved this code in the HTML file and having browsed it by means of your favourite browser, you will see a usual HTML form:

Beginning of a form

Enter your name:

End of a form

Let's consider the tags used in this small example in more detail.

The tag <form> having pair ending tag </form>, actually also sets a form. Its attributes - both are optional:

J action - specifies URL (full or relative) to which the form will be sent. Sending a form is the same server request, as well as all others (as I already described above).

If not to specify this attribute, the majority of browsers (more precisely, all browsers known to me) send a form to the current document, that is "to themselves". This convenient reduction, but according to the action HTML attribute standard it is obligatory.

J method - a way of sending a form. They are two.

o GET - sending this form in an address bar.

You could notice presence at the end of URL of the character on the different websites "?" and

the following data behind it in a format parameter = value. Here "parameter" corresponds to value of name attribute of elements of a form (see below about a tag <input>), and "value" - to value attribute contents (it, for example, contains input of the user in a text box of the same tag <input>).

For an example - try to look for something in Google and pay attention to an address bar of the browser. It is also the way GET.

o POST - these forms go in a request body. If it is not absolutely clear (or it is absolutely unclear) what is it - do not worry, we will return to this question soon.

If the method attribute is not specified - "GET" is meant.

The tag <input> - sets the form element determined by type attribute:

J text Value sets an input single-line text box

J submit Value sets the button when which clicking there is a sending a form for the server

Also other values are possible (and <input> - not the only tag setting a form element), but we will consider them in the following chapters.

So, what occurs when we click OK?

1. The browser browses the elements incoming a form and creates these forms of their name and value attributes. The name Bryan is Let's say entered. In this case these forms - name=Bryan&okbutton=OK

2. The browser establishes connection with the server, sends a request of the document specified in action attribute of a tag <form> for the server, using the method of sending data specified in method attribute (in this case - GET), transferring these forms in a request.

3. The server analyzes the received request, creates the answer, sends it to the browser and closes connection

4. The browser displays the document received from the server

Sending the same request manually (with the help of telnet) looks as follows (we will assume that domain name of the website - www.example.com):

telnet www.example.com 80

GET /cgi-bin/form_handler.cgi?name=Bryan&okbutton=OK HTTP/1.0\r\n

Host: www.example.com\r\n

\r\n

As you, most likely, already guessed, clicking of the submit-button in shape with method of sending "GET" is similar to input of the relevant URL (a signed question and data of a form at the end) in an address bar of the browser: http://www.example.com/cgi-bin/form_handler.cgi?name=Bryan&okbutton=OK

Actually, the GET method is used always when you request any document from the server, having just entered its URL, or having clicked on the link. When using <form method= "GET"...>, the question mark and these forms are just added to URL.

Perhaps, all these technical details and exercises from telnet-ohms seem to you incredibly boring and even unnecessary ("and at what here PHP?"). And in vain. These are the basics under the HTTP protocol which each Web programmer needs to know by heart, and it is not theoretical knowledge - all this is useful in practice.

Now we will replace the first line of our form with following:

1. <form method= "POST" action= "/cgi-bin/form_handler.cgi">

We specified a method of sending "POST". In this case data go to the server a little in a different way:

telnet www.example.com 80

POST/cgi-bin/form_handler.cgi HTTP/1.0\r\n

Host: www.example.com\r\n

Content-Type: application/x-www-form-urlencoded\r\n

Content-Length: 22\r\n

\r\n

name=Bryan&okbutton=OK

When using the POST method these forms go after "two Enter-ov" - in a request body. Everything that above - actually request heading (and when we used the GET method, these forms went in heading). In order that the server knew on what byte to finish reading a body of a request, at heading there is a line Content-Length; about the same that these forms will be transferred a look параметр1=значение1&параметр2=значение2..., and values are

transferred in a type of urlencode - that is, in the same way as by means of the GET method, but in a request body, - to the server reports heading "to Content-Type: application/x-www-form-urlencoded".

That such urlencode - is slightly lower.

Advantage of the POST method - lack of restriction for length of a line with data of a form.

When using the POST method it is impossible to send a form, it is simple "having visited at the link" as was with GET.

For brevity statements, we will enter the terms "GET form" and "POST form" where the prefix corresponds to value of method attribute of a tag <form>.

When using a POST form, it is possible to specify in its action attribute after a question mark and GET form parameters. Thus, the POST method includes also the GET method.

I-3. CGI technology

We understood the previous chapter how to create the HTML form and as the browser sends the data entered into it to the server. But it is meanwhile unclear what the server will do with these data.

The Web server in itself just is able to give the requested page, both nothing moreover, and all transferred these forms, in general, are absolutely indifferent for it. In order that it was possible to process these data with the help of any program and to dynamically create the answer to a browser, and the CGI technology (Common Gateway Interface) was invented.

Let's look at this URL: http://www.example.com/cgi-bin/form_handler.cgi. The first assumption which can be made into its account is normal it: the server gives form_handler.cgi file contents from the cgi-bin directory. However, in a case with CGI technology the situation is in a different way. The server starts the form_handler.cgi program and transfers it these forms. The program creates the text which is transferred to the browser as reply to the request.

The form_handler.cgi program can be written in any programming language, the main thing - to observe the CGI standard in the program. It is possible to use, for example, the popular scripting Perl language. And it is possible to write everything and on the SI. Or on shell-scripts... But we, for an example, will write this program on the SI. But at first we will understand as there is a data exchange between the Web server and the CGI program.

J Before start of the CGI program, the server sets variable environments (they are for certain familiar to you at the command of PATH). In everyone in the slightest degree a serious programming language there are means for reading variables of an environment. The CGI

standard defines very considerable set of variables which should be defined before start of the CGI program. Let's consider now only three of them:

o REQUEST_METHOD - the transmission method of data - GET or POST (there are also others, but so far we do not consider them)

o QUERY_STRING - supports a part of URL after a question mark, or, in other words, these GET forms.

o CONTENT_LENGTH - the request body length (these POST forms).

J the Server starts the CGI program. The body of a request is transferred to the program in the form of standard input (stdin) - as if these data were entered from the keyboard.

J the Program issues the response of a browser to standard output (stdout) - "on the screen". This output is intercepted by the Web server and transferred to the browser.

```
1.  #include <stdio.h>
2.  #include <stdlib.h>
3.
4.  int main(void)
5.  {
6.  //We read the variable environments set by the Web server
7.  char * query_string = getenv ("QUERY_STRING");
8.  char * request_method = getenv ("REQUEST_METHOD");
9.
10. char * post_data;//The buffer for data of a POST request
11. int post_length = 0;//Request body length
12.
13. if (strcmp (by request_method, "POST") == 0) {//If is received a POST request,
14. post_length = atoi (getenv ("CONTENT_LENGTH"));//at first we read from
```

```
15.  //variable environment its length,
16.  if (post_length) {//if it not zero,
17.  post_data = (char *) malloc(post_length+1);//we select memory for the buffer,
18.  fread (post_data, post_length, 1, stdin);//we read a request body from standard input,
19.  post_data[post_length] = 0;//we complete a line in zero byte.
20.  }
21.  }
22.
23.  //We display answer heading...
24.  printf ("Content-type: text/html\r\n\r\n");
25.
26.  //and his body:
27.  printf (" <h1> Hello! </h1> \r\n");
28.
29.  if (strlen(query_string)) {
30.  printf (" <p> GET form Parameters: %s\r\n", query_string);
31.  }
32.
33.  if (post_length) {
34.  printf (" <p> POST form Parameters: %s (request body length: %d) \r\n", post_data, post_length);
35.  free(post_data);//we do not forget to release the memory selected in line 17
```

```
36.    }

37.

38.    return 0;

39.    }
```

It is the elementary CGI program on the SI displaying contents of the parameters of forms received from the Web server. The browser as a result will receive approximately following code (if "submit" on this program a POST form from the last example):

<h1> Hello! </h1>

<p> POST form parameters: name=Bryan&okbutton=OK (request body length: 22)

What at the same time will be displayed on the user's screen, I think, clear no comments.

As you can see, even the simplest output program of parameters is not so simple. Moreover, according to the HTTP standard almost all not alphanumeric characters are transferred in a so-called UrlEncoded-look (%XX where XX - a hexadecimal code of the character), and if to add UrlEncode interpretation code to the provided SI program, it will not be located on the screen any more. And it - only basic transactions. And how the program will grow by the SI if it is necessary to work with the database?

However, writing of CGI programs on the SI - quite rare perversion. Most often it is done on Perl - the language developed especially for processing of text data, and existence of the CGI module does writing of CGI scripts by much simpler task. Here I will not acquaint you with Perl, I will note only that problems remain enough: nevertheless Perl is not intended for Web, it is a universal language. And the CGI technology is imperfect: at each address there is a start of the program (in a case to Perl - the language interpreter), and this transaction quite resource-intensive: for a homepage of Bryan Ravensky of productivity, of course, it is enough, but the serious portal in about tens and in hundreds of thousands of hits a day will demand already huge hardware capacities.

And now we will look at the Apache Web server. Inherently it modular, also allows to connect expansions by adding one line in a configuration file. (Well, well, well, two lines.). It would be fine if there was scripting language, whether ground under Web connected by the module to the Apache, so? Well, you already understood what I drive at - it is and there is PHP.

In principle, PHP can be compiled and as a CGI-application, and to use as well as Perl - but it for non-standard Web servers or special perverts.

II. PHP: Hypertext preprocessor

In the 1994th year, one programmer, by the name of Rasmus Lerdorf, tortured with the classical pearl-barley CGI module, decided to write several own Perl-scripts that it was simpler to create own home page, and called all this business of Personal Home Page (PHP). After a while it needed to process forms, well and for increase in productivity everything was rewritten on C - Personal Home Page/Forms Interpreter (PHP/FI) 2.0 so appeared. The works Rasmus, following the principles of Open Source, laid out on a public inspection, and, in principle, on some quantity of the websites PHP/FI it was quite successfully used though it was quite primitive.

In the 1997th two other programmers - Andi Gutmans and Zeev Suraski came across PHP/FI - in search of the tool for convenient Web scripting-. The idea was pleasant to them, but functionality and speed of work of PHP/FI left much to be desired, and Andi and Zeev decided to rewrite PHP from scratch. Language turned out universal and powerful, and soon drew attention of a great number of web developers: by the end of 1998 of PHP3 it was used on ~ 10% of Web servers. The modest name "Personal Home Page" already not really corresponded to reality, and the name was changed on - in the best Unix-traditions - recursive: PHP: Hypertext Preprocessor.

The "engine" of PHP 4 called by Zend Engine was developed by efforts already created and since then is continuous growing PHP community, and in the 2000th year there was the 4th version of PHP which became less than in half a year the standard for Web development for Unix (and not only): each hoster respecting himself provided support of PHP. Now development of PHP5 based on new Zend Engine 2 comes to an end...

However, there will be enough lyrics. Let's look at a simple PHP script. At first we will a little change the HTML form from the previous section:

```
1.  <form method= "POST" action= "form_handler.php">

2.  Enter your name: <input type= "text" name= "name">

3.  <br>

4.  <input type= "submit" of name= "okbutton" value= OK>

5.  </form>
```

And now - form_handler.php:

```
1.  <html>

2.  <body>
```

```
3.  <?
4.  echo" <h1> Hi, <b>". $ _POST ['name']." </b> </h1>!";
5.  ?>
6.  </body>
7.  </html>
```

Unlike the SI or Perl, the PHP script represents the normal, in general, HTML page: "just like that" the written tags are transferred "as is", as if it is a normal html-nickname. The script consists in special tags <? and?>, in which we use for output of the text echo operator. Such blocks there can be as much as necessary, everything that between them, is interpreted as normal html.

Variables of a GET request get to an array of $ _GET, POST request variables - in an array of $ _POST, server variables (the IP addresses type, a script name, etc.) - in $ _SERVER, "point" (.) operator - association of lines... And all housekeeping operations (reading stdin and variables of the environment, Url-decoding) were already made by PHP. It is convenient, isn't it?

Further. Why to us two files - HTML with a form and a PHP script? One script suffices:

```
1.  <html>
2.  <body>
3.  <?
4.  if ($ _SERVER ['REQUEST_METHOD'] == 'POST') {
5.  echo" <h1> Hi, <b>". $ _POST ['name']." </b> </h1>!";
6.  }
7.  ?>
8.  <form method= "POST">
9.  Enter your name: <input type= "text" name= "name">
10. <br>
```

11. <input type= "submit" of name= "okbutton" value= OK>

12. </form>

13. </body>

14. </html>

We removed action attribute from form tag - it means that the form goes "to itself", i.e. to the current URL. It is called sometimes by "postback form". In line 4 by means of if operator it is checked whether the POST method was used for loading of the document (an analog of a line 13 of an example on the SI), and - if it so - the next line is displayed a greeting.

On this simple program - some kind of Web option "Hello World" - we will also complete input chapter.

2. Installation and Apache+PHP configuration

Keeping of the head

Printer-friendly version

1. Apache installation

1. -in Windows OS

2. -in Unix family OS

2. PHP installation

1. -in Windows OS

2. -in Unix family OS

3. Apache+PHP configuration

4. PHP5 installation

Before to start studying of PHP, it is quite good to set it. And, of course, the Web server is required - we will stop on Apache 1.3 as on the most popular and stable together with PHP.

If you decided to use Apache 2, is strongly recommended to bring together it with prefork MPM - see documentation.

Also (it concerns users of Windows) you should not be tempted with convenience of installation of ready sets, the Denver-2 type: in total, of course, "will earn itself", but if necessary it is required to change configuration files - and it to you surely - lack of experience of independent installation will not help you at all. As option - you can set Apache+PHP manually, understand as to configure everything, and then to uninstall fruits of the works and to set same Denver if it so is pleasant to you.

We will consider the Apache+PHP installation in Windows OS (considering only the "real" OS - NT/2000/XP/2003) and Unix (in particular, Linux and FreeBSD).

Apache+PHP... "And how MySQL?" - the advanced user will ask... And we will set MySQL later - so far it is not necessary to us.

I. Apache installation

I-1. Apache installation in Windows family OS

1. Download Windows distribution kit - Apache 1.3.xx of Win32 Binary (Self extracting) - from the next to you a mirror:

httpd.apache.org

2. Make sure that login under which you work enters the Administrators group

3. Make sure that the 80th TCP port is not occupied with any other service or the application, for example, the Microsoft IIS. If the 80th port is occupied, change port in the IIS-a settings (or other service) to another, either disconnect or uninstall this application.

At desire, you can independently set PHP under the Microsoft IIS (following the instruction in install.txt), to start a simple script, to look how all this brakes then to demolish IIS and to set Apache.

4. Start the downloaded file... Well under Windows you to put software, I hope, you are able? When asks whether to start Apache service (for all users) - select "service" (it is default setting).

I-2. Apache installation in Unix family OS

Classically, in Unix the software is established by assembly from sources:

J we Download and we unpack a distribution kit with httpd.apache.org by means of a browser like lynx/links or the fetch/wget team, for example:

$ fetch http://apache.rin.com/dist/httpd/apache_1.3.31.tar.gz

$ tar xzf apache_1.3.31.tar.gz

J we Configure Apache:

$ cd apache_1.3.31

$. / configure

For the thin Apache configuration, it is necessary to study the INSTALL file and to specify necessary parameters in a line. / configure. However, as a rule, default settings quite approach.

J we Compile Apache:

$ make

J we Set Apache:

$ su - we come under the superuser (root) if all previous transactions made from under the normal login)

make install

At the stage configure probably you will see the errors connected with absence in system of necessary libraries. They need to be downloaded and set according to the instructions which are in files of INSTALL of distribution kits (usually - it is the same sequence. / configure && make && make install). This business quite long and boring, therefore for a long time the majority of Unix-systems include this or that more convenient installation facility of the software.

FreeBSD

FreeBSD includes special means for software installation - ports (ports) representing a set of the scripts which are automatically downloading, configuring, compiling and installing software products including libraries, necessary for start (dependences - dependences). I will not paint in detail here as in FreeBSD to work with ports - it is described in FreeBSD Handbook and a set of books. Let's assume that in your system the library of ports is set and updated.

J we Pass into the directory of Apache port:

$ cd/usr/ports/www/apache13

J we Start the team of assembly and the Apache installation from under root'a (the distribution kit will be downloaded automatically):

$ su

make install

J If it is necessary to start automatically Apache when loading - we register in / etc/rc.conf:

apache_enable= "YES"

Actually, that's all.;)

In need of thin configuration study Makefile and add necessary parameters after the make install team (for example, make install WITH_APACHE_SUEXEC=yes - for support of suexec).

Red Hat Linux and other RPM-based distribution kits (ASP, Mandrake...)

Red Hat Linux includes the RPM package manager. The installation approach from RPM essentially another - .rpm-packets contain already compiled binary files. It gives a gain in time (it is necessary to compile nothing), however, deprives of an opportunity to collect the program as you want.

For the Apache installation from rpm it is necessary to load the Apache 1.3 .rpm-file (from where - look for Google: Apache Software Foundation (apache.org) does not extend RPM and start: rpm - Uvh apache.

Gentoo Linux

Gentoo Linux - the "BSD style Linux" which is inherently - contains portages - system of ports similar on used in FreeBSD. Installation is made by means of the emerge team, for example:

$ cd/usr/portage/net-www/apache

$ su

emerge apache-1.3.31.ebuild

Do not start just emerge apache - it will lead to the Apache 2 installation.

For automatic start of Apache when loading, enter

rc-update add apache default

Other Linux distribution kits

Other Linux distribution kits (Debian, Slackware...) contain own package managers - address documentation. Anyway (even in case of other Unix OS) you can compile and set Apache manually - as it is described above.

III. PHP installation

III-1. PHP installation in Windows family OS

Download from the www.php.net/downloads.php page Windows Binaries ZIP archive (PHP 4.x.x Zip package).

Do not swing "PHP 4.x.x installer" - there are no many necessary files!

Unpack archive in C:\PHP (or to any other place - but it will be assumed further that you selected C:\PHP). Copy the php4ts.dll file in the C:\WINDOWS\SYSTEM32 (or the corresponding directory). Copy the php.ini-dist file in the C:\WINDOWS и rename it in php.ini directory.

III-2. PHP installation in Unix family OS

Installation process of PHP is similar to the Apache installation except that at assembly of PHP you need to specify much more configuration options.

At manual assembly from sources - for this purpose it is necessary to download and unpack the source code with www.php.net/downloads.php - at first start. / configure - help also study the parameter list of assembly (very impressive).

Anyway, parameter - with-apxs=/put/k/apxs is required - it is necessary for assembly of the module of the Apache of mod_php. For example, if Apache is set in / usr/local, then parameter will look as follows: - with-apxs=/usr/local/sbin/apxs.

After configuring, start make, and make install.

The more you use the options connecting different expansions the more it is required to download and set necessary libraries manually. It is better to use, of course, a package manager or ports of your OS.

Irrespective of the selected way, after successful installation, copy php.ini-dist in php.ini:

cd/usr/local/etc

cp php.ini-dist php.ini

(depending on OS, instead of / usr/local/etc can be used by directory/etc, etc.).

FreeBSD

It is convenient to PHP to set from FreeBSD ports.

Having used the su team for obtaining the rights of the superuser, we will pass into directory/usr/ports/lang/php4.

There are two options of assembly of PHP from ports - interactive and by means of make team parameters.

In the first case, we will just start the make install team. After make will download and will unpack the PHP distribution kit, on the screen the window with the list of possible options of configuring will appear - just put "crosses" against necessary.

The interactive way, at all its convenience, has a shortcoming - the parameters selected once cannot be saved for use of the same configuration repeatedly. It is possible to specify all necessary parameters in the command line, for example make install WITH_MYSQL=YES WITH_GD=YES. The list of possible options is in the /usr/ports/lang/php4/scripts/php4_options file (this file, by the way, and the script displaying a window in an interactive mode uses).

Update: recently (probably, from a big hangover) the meyntener of PHP port decided to remake everything. And not only solved (alas). Now assembly of PHP with static linking of expansions (i.e. compilation of PHP and the selected expansions in one so-file) from ports is impossible. Or handles, or - if dynamic (shared) expansions arrange - cd/usr/ports/lang/php4, make install, cd/usr/ports/lang/php4_extensions, make install. For php5 - similarly. Corresponding lines extension=. will register in php.ini automatically. However, when using PHP as Apache-module, dynamic expansions at all not more slowly static so you should not be upset especially about it.

Red Hat Linux and other RPM-based distribution kits (ASP, Mandrake...)

As as it was already told, Red Hat-packets contain already compiled binary files, you should find on the Internet ready rpm with the configuration which is most suitable you and to set it, having used the rpm team.

Gentoo Linux

Gentoo-portage PHP (Apache module) there is in / usr/potrage/dev-php/mod_php. For the

indication of the configure parameters use the USE variable (the incomplete list of possible USE values in Gentoo Linux: www.gentoo.org/dyn/use-index.xml). It is possible to edit etc/make.conf, but it is more convenient so:

USE= "gd mysql pdf xml xslt-X" emerge mod_php

In this case, the USE variable is established only on runtime of the emerge team. (" - X" - for a guarantee that any library will not pull along XFree86).

After installation, the php.ini file (to be exact - symlink on it) is in / etc/apache/conf.

IV. Apache+PHP configuration

If it was not made yet by the installation program of packets/ports, add the next lines to the configuration file of Apache httpd.conf:

J For OS Windows:

LoadModule php4_module C:/php/sapi/php4apache.dll

J For OS Unix:

after the last directive LoadModule

LoadModule php4_module libexec/apache/libphp4.so

J For all OS:

after the last directive AddModule

AddModule mod_php4.c

In the <IfModule mod_mime.c> block

AddType application/x-httpd-php .php

Prescribe in the line DirectoryIndex to index.php:

DirectoryIndex index.php index.html

Connection of expansions

In Windows, for connection of expansions it is necessary to uncomment the corresponding lines extension=... in php.ini, without having forgotten to register a way to them in the line

extension_dir=. (usually they are in c:\php\extensions for php4 and in c:\php\ext for php5).

In Unix (if only you did not specify shared attribute of parameters - with... at assembly), expansions are connected by reassembly of PHP.

php.ini settings

If you got on this article not accidentally (from Google, having thought that for the downloaded forum or a chat the Web server is necessary), and are going to study PHP - set values of the specified variables quite so. Also do not argue. That why and why - I will explain later.

register_globals = off

magic_quotes_gpc = off

magic_quotes_runtime = off

error_reporting = E_ALL - and quite so, any E_ALL & ~ E_NOTICE and so forth!

display_errors = on

Apache start

Now everything is ready to Apache start. In Windows:

> net start apache

In Unix:

apachectl start

V. PHP5 installation

Apache+PHP5 is established in the same way as Apache+PHP4 - just everywhere replace in this instruction of PHP4 with PHP5 (incl. and in names of files and configuration directives).

In particular, for Windows/Apache/PHP5, it is necessary to register in httpd.conf:

LoadModule php5_module C:/php5/php5apache.dll

AddModule mod_php5.c

AddType application/x-httpd-php .php

Self-instruction manual of PHP: Chapter 3. Bases of syntax of PHP

Keeping of the head

Printer-friendly version

1. As the PHP program looks
2. Variables and data types
3. Conditional statements
   1. if
   2. switch
4. Cycles
   1. while
   2. do... while
   3. for
5. Arrays
   1. Foreach loop
   2. Constructions of list and each
6. Constants

As the PHP program looks

Unlike traditional scripting languages (such as Perl), the PHP program represents the HTML page with code inserts. For comparison:

Perl-script:

```
#!/usr/local/bin/perl

print "Content-type: text/html\n\n";

print" html\n<head> <title> Hello World </title> </head> \n";

print" <body> <h1> of Hello World! </h1> </body> \n";

print" </html>";
```

PHP script (yes, it is the program for PHP;)):

```
<html>

<head> <title> of Hello World </title> </head>

<body> <h1> of Hello World! </h1> </body>

</html>
```

As you can see, the elementary program for PHP is a normal HTML page. About output of the heading Content-type: text/html PHP took care independently too.

Directly a PHP code (which - not HTML is placed between tags <? and?>. Everything that is located between these tags, is replaced on displayed by a script in this block a HTML code (in that specific case - if the script displays nothing - just "disappears").

In general, universal (that is - PHP which is guaranteed working at any configuration), but longer way of the specification of a PHP code - tags <? php...?>. Such long form of record is used at combination of XML and PHP as a tag <?...?> it is used in the XML standard.

For recognition of a tag <? as the beginnings of the PHP block the directive short_open_tag of the php.ini file (by default - it is included) answers. If you want to develop the scripts working irrespective of this configuration use long opening tag <? php. I will use the contracted form.

Let's review a simple example.

1. <html>

2. <head> <title> of Hello World </title> </head>

3. <body> <h1> of Hello World! </h1>

```
4.  <p> Current date:
5.  <?
6.  echo date ("d.m.Y");
7.  ?>
8.  </body>
9.  </html>
```

For accomplishment of examples, copy them in the file located in the directory, corresponding to the directive DocumentRoot of the configuration file Apache httpd.conf (for example, in the file with the name test.php), and execute them, having addressed the saved script (test.php) from an address bar of the browser (http://localhost/test.php). And if you did not set Apache+PHP yet (as so?;), address the previous chapter.

If today - the 27th of July, 2016, as a result of execution of a script the browser receives the following HTML code:

```
<html>

<head> <title> of Hello World </title> </head>

<body> <h1> of Hello World! </h1>

<p> Current date:

27.07.2016 </body>

</html>
```

Lines 5,6,7 - a PHP code insert. On lines 5 and 7 are located respectively the opening and closing tag. Them it is absolutely optional to locate on separate lines - it is made for reasons of convenience of reading.

In line 6 the echo operator used for output in the browser is located. It displays a result of execution of the function of date - in this case it is current date.

The line 6 is the finished expression. Each expression in PHP comes to an end with a semicolon-;. A semicolon, but not line feed - do not forget about it, especially if you programmed on Visual Basic or ASP earlier.

The attentive reader will notice that the tag </body> is located on the same line, as the text created by the date function () though he </body> is in the source code on a separate line. The matter is that PHP discards the line feed following right after closing tag?> - it is made specially that in HTML fragments where excess spaces are undesirable, there was no need to endow readability of a script, writing the closing PHP tag on one line with the subsequent HTML code. If the space is necessary - insert later?> blank line.

Variables and data types

Variables in PHP begin with a sign of $ which any set of Latin letters, digits and a sign of underlining follows: _, at the same time the digit cannot follow $ sign at once.

The letter case in a name of a variable matters: $A and $a are two different variables.

For assignment of a variable of value the operator = is used.

Example:

```
1. <?
2. $a = 'test';
3. $copyOf_a = $a;
4. $Number100 = 100;
5. $a echo;
6. $copyOf_a echo;
7. $Number100 echo;
8. ?>
```

This code will display: testtest100.

You monitor what you name variables: it is unlikely you in half a year remember for what the variable of $a21 or $zzz is used. And here for what $username variable is used, to understand quite easily.

In a line of the 2nd variable of $a line test value is appropriated. Lines in PHP register in quotes - unary or double (we will consider distinction between records in different quotes a bit later). The expression is also fair that the variable of $a is initialized by test value: in PHP the variable is created at the first assignment of value by it; if value was not appropriated to a

variable - the variable is not defined, that is it just does not exist.

In line 3 the variable of $copyOf_a is initialized by variable value of $a; in this case (we watch a line 2) this value - the line 'test'. In line with number 4 of a variable with a name of $Number100 numerical value 100 is appropriated.

As you can see, in PHP there is a typification, that is language distinguishes data types - lines, numbers, etc. However, at this PHP is language with weak typification - conversions between data types happen automatically by the set rules. For example, program <? echo '100' + 1;?> will display number 101: the line will automatically be transformed to number when using in a numerical context (in this case - the line '100', when using as composed, will be transformed to number 100 as addition operation for lines is not defined).

Such behavior (and also lack of need (and even opportunities) is explicit to define variables) makes related PHP with Perl and Basic, however, it is possible, will be given a hostile reception by adherents of strict languages, such as the SI and Pascal. Of course, need of accurate definition of variables and the requirement of explicit reduction of types reduces number of possible errors of programming, however, of PHP, first of all, - an interpreted language for rapid development of scripts, and not severity of syntax is with interest compensated by coding speed. And will always obligingly report about unassigned variable of PHP - if, of course, not to prohibit it to it... However, I run forward.

Let's review one more example:

1. <?
2. $greeting = 'Hi';
3. $name = Bryan;
4. $message = "$greeting, $name!";
5. $message echo;
6. ?>

The special attention is deserved by the fourth line. In double quotes the variables defined the last lines are specified. If to execute this program (you already made it?), in a window of the browser the line Hi, Bryan will be displayed!. Actually, the main feature of double quotes also consists in it: names of the variables specified in couple of characters" are replaced with the values corresponding to these variables.

In addition, in double quotes the special managing combinations consisting of two characters, first of which - the return slash will be recognized (\). The following control characters are

most often used:

J \r - carriage return (CR)

J \n - line feed (NL)

J \" - a double quote

J \$ - the character of dollar ($)

J \\-actually, the return slash (\)

Characters \r and \are usually used by n together, in the form of combination \r\n - line feed in Windows and many is so designated by TCP/IP protocols. In Unix the new line is designated by one character \n; usually such way of line feed is used also in HTML documents (of course, it influences only a HTML code, but not display in the browser (if only the text is not put into couple of tags <pre>... </pre>): for the displayed line feed, as we know, the tag
 is used).

The remained three points from the provided list of application of the return slash are examples of shielding - cancellings of special action of the character. So, the double quote would designate a line end, the dollar character - the beginning of a name of a variable, and the return slash - the beginning of a managing combination (about which we here also speak;)). When shielding, the character is perceived "as it is", and any special actions is not made.

Shielding (not the return slash, but the principle;)) in PHP it is used in many cases - so we will meet this approach still more than once.

If in given to replace quotes on unary, in the browser what in them is written ($greeting, $name is displayed!). The combinations of characters beginning with \in single quotes also will not be transformed in any way, behind two exceptions: \' - a single quote in a line; \\-the return slash (in quantity one piece.

Let's a little change our last example:

```
1.  <?
2.  $greeting = 'Hi';
3.  $name = Bryan;
4.  $message = $greeting.','. $name.'!';
```

5. $message echo;

6. ?>

This time we did not begin to use "complaisance" of double quotes: in line 4 names of variables and string constants are written through the operator of concatenation (association of lines). In PHP concatenation is designated by a point-. The result of accomplishment of this program is similar to the previous example.

I recommend to use this writing method - on that is enough reasons:

J Names of variables are more accurately visually separated from line values that it is the best of all considerably in the editor with code illumination;

J the PHP Interpreter processes such record a little quicker;

J PHP will be able to trace more accurately a typo in a variable name;

J you do not make a mistake, similar following: $message = "$greetingBryan" - PHP in this case will bring not "HiBryan", but blank line because $greetingBryan will be recognized as a name variable, and we do not have that.

However, double quotes are very popular, and you for certain will meet them in a set of the scripts available in network.

By the way, in the last two examples there is no need for definition of a variable of $message at all: lines 4 and 5 can be reduced to $greeting echo.','. $name.'!';. And if $message variable can be necessary for us in the following code - it is possible to write $message echo = to $greeting.','. $name.'!'; and it will work. It is connected with the fact that result of the expression containing assign is the appropriated value. It is especially convenient at assignment of the same value to several variables. For example, if variable $a and $b need to appropriate the same value (we will tell, number from a floating point 10.34), it is possible to write $a = to $b = 10.34;.

The set of the embedded functions for work with lines is provided in PHP. Them you can find the description in official documentation.

In addition to lines and numbers, there is one more simple, but important data type - Boolean (bool) to which two special values belong: true (truth) and false (lie). At automatic reduction of types, false there corresponds to number 0 and blank line (''), true - to all the rest. Boolean values are often applied together with conditional statements about which we farther also will talk.

Conditional statements

if

Often (yes what here to say, practically in any program) there is a need of accomplishment of a different code depending on certain conditions. Let's review an example:

```
1.  <?
2.  $i = 10;
3.  $j = 5 * 2;
4.  if ($i == $j)
5.  echo 'Variable $i and $j Have Identical Values';
6.  else
7.  echo 'Variable $i and $j Have Different Values';
8.  ?>
```

Here the if operator is used. else - conditional statement. In a general view it looks so:

```
if (condition)
expression_1;
else
expression_2;
```

In this case, a condition is the result of comparison of variables values of $i and $j. Comparison operator - == - two equality signs. As 5*2 equals 10, and, respectively, 10 equals;), the line 5 will be executed, and we will see that variables have equal values. Change, for example, a line 2 to $i = 11, and you will see that the echo operator will be executed from a line 7 (as the condition is false). In addition to ==, there are also other comparison operators:

! = - not equally;

<-it is less;

> - it is more;

<= - it is less or equally;

> = - it is more or equally.

Experiment, changing comparison operator and variables values. (For logical correctness of output to the screen, it will be required to change, of course, and the texts output by echo operators).

Do not confuse comparison operator == to assignment statement =! If you make such mistake, the condition will always be correct if the value corresponding to Boolean true, and always false is appropriated - if value corresponds to false. (See above about boolean values and what they correspond to).

If the else block is only required to perform operation if the condition is satisfied... it is possible to lower:

```
1.  <?
2.  $i = 10;
3.  $j = 5 * 2;
4.  if ($i == $j)
5.  echo 'Variable $i and $j Have Identical Values';
6.  ?>
```

In this case, if the condition is false, in the browser nothing will be displayed.

Indents before the lines echo... are made for convenience of reading, but PHP they do not speak about anything. The following example works not as it is possible to expect:

```
1.  <?
2.  $i = 10;
3.  $j = 11;
4.  if ($i> of $j)
5.  $diff = $j - $i;
```

```
6.  echo' is more than $j, than $i; the difference between $j and $i makes '. $diff;//INCORRECTLY!

7.  ?>
```

Contrary to possible expectations, the line 6 will be executed though the condition ($i> of $j) is false. The matter is that to if (...) only the following expression - a line 5 belongs. The line 6 is carried out anyway - action of if (.) does not extend to it any more. For obtaining the necessary effect it is necessary to use a statement block which is set by curly brackets:

```
1.  <?

2.  $i = 10;

3.  $j = 11;

4.  if ($i> of $j) {

5.  $diff = $j - $i;

6.  echo' is more than $j, than $i; the difference between $j and $i makes '. $diff;

7.  }

8.  ?>
```

Now everything works correctly.

Curly brackets can be used even if inside - only one operator. I recommend to arrive quite so - less chances to be mistaken. It does not affect productivity in any way, but increases readability.

It is often necessary to enter additional conditions (if so... and if in a different way... otherwise) or even (if so. and if in a different way. and if still in a different way... otherwise):

```
1.  <?

2.  $i = 10;

3.  $j = 11;

4.  if ($i> of $j) {

5.  echo' is more than $i, than $j';
```

```
6.  } else if ($i <$j) {
7.  echo' is less than $i, than $j';
8.  } else {//nothing, except equality, remains
9.  echo' to $i is equal to $j';
10.  }
11.  ?>
```

For additional "forks" the if operator is used... else if... else. As well as in a case about if, the else block can be absent. Following the to the recent recommendation, I concluded everything operators echo in curly brackets though everything perfectly would work also without it.

By the way, in line 8 - the comment. It is information for the person, PHP ignores it. Comments happen two types: one-line as here - begins with//and extends to a line end, and multiline - everything that is located between couples of characters/* and */is considered the comment.

The comment of a look//-one of the few cases when the instruction comes to an end with line feed. I will remind - PHP are in most cases indifferent translations of lines: all previous examples could be written in one line.

Switch.

There is a need of implementation of "fork" depending on value of the same variable or expression. It is possible to write something it seems:

```
 if ($i == 1) {
   //code, corresponding $i == 1
 } else if ($i == 2) {
   //code, corresponding $i == 2
 } else if ($i == 3) {
   //code, corresponding $i == 3...
 }
```

But there is an operator, more convenient for this case, - switch. It looks so:

```
1.  <?
2.  $i = 1;
3.
4.  switch ($i) {
5.  case 1:
6.  echo 'one';
7.  break;
8.  case 2:
9.  echo 'two';
10. break;
11. case 3:
12. echo 'tri';
13. break;
14. default:
15. 'I am able to consider echo only to three!;)';
16. }
17. ?>
```

Observe result of program execution, changing value of $i in the second line. As you already for certain understood, after switch in brackets the variable (though there can be also an expression - for example, - try $i+1) is specified, and the lines case XXX correspond to value of the fact that in brackets.

The operators who are between case-a it is not necessary to conclude in curly brackets - each branch comes to an end with break operator.

The special condition of default corresponds "to all the rest" (else analog in if... else if. else). default raspolagtsya always by the last so break is optional here. As well as in a case with else, the condition of default can be absent.

If you suddenly forget to specify break, all next lines - will be carried out from the subsequent case-! For example, if in our example to delete a line 6, at $i == 1 in the browser "onetwo" will be displayed. Some too cunning programmers use this trick for the indication of several options of values:

```
1.  <?
2.  $i = 1;
3.
4.  switch ($i) {
5.  case 0://break is absent intentionally!
6.  case 1:
7.  echo 'zero or one';
8.  break;
9.  case 2:
10. echo 'two';
11. break;
12. case 3:
13. echo 'tri';
14. break;
15. }
16. ?>
```

or for accomplishment at a certain value of a condition of two actions in a row. But it already tricks - is the best of all to use switch "as it is necessary", finishing each case with the of

break-ohm; and if you "shift" - do not forget to put the comment as it is made in a line 5 of the last example.

Cycles

Any more or less serious programming language contains operators of an ogranization of cycles for repeated accomplishment of fragments of a code. In PHP there are three such operators.

while

Let's begin with the cycle while:

```
1.  <?
2.  $i = 1;
3.  while ($i <10) {
4.  $i echo. "br\n";
5.  $i ++;
6.  }
7.  ?>
```

The cycle while (line 3) works as follows. At first truth of expression in brackets is checked. If it is not true, a loop body (everything that is located between the subsequent curly brackets - or if they are absent - the following instruction) is not carried out. If it is true, after the code execution which is in a loop body truth of expression, etc. is again checked.

In a loop body (line 4,5) the current variable value of $i then value of $i increases by unit is displayed.

The variable used like $i in this example is often called a variable loop counter, or just the counter.

$i ++, incrementing transaction (increases in value on 1) - an abbreviated notation for $i= $i+1; a similar abbreviated notation - $i+ =1. By the last rule it is possible to reduce any binary operations (for example, concatenation: $s. = 'foo' - $s analog = $s. 'foo'); however, similar to incrementing it is possible to write only decrementing (reduction of value on 1): $i-.

Also record ++ $i is possible (and - $i); distinction in an arrangement of signs of transaction is shown only at direct use of result of this calculation: if to $i it is equal 1, in case of $j= $i ++ the variable of $j will receive value 1 if $j= ++ $i, $j equals to two. Because of this feature transaction ++ $i is called a preincrement, and $i ++ - a post-increment.

If we did not increasing value of $i, loop termination would never happen ("an eternal cycle").

Let's write the same example in shorter form:

```
1.  <?
2.  $i = 1;
3.  while ($i <10) {
4.  $i echo ++. "br\n";
5.  }
6.  ?>
```

And one more option:

```
1.  <?
2.  $i = 0;
3.  while (++ $i <10) {
4.  $i echo. "br\n";
5.  }
6.  ?>
```

I advise a little to think why all these three programs work equally. Notice that depending on starting value of the counter this or that form of record is more convenient.

do. while

Cycle do. while is almost similar to the cycle while, differing from it in the fact that the condition is in the end of a cycle. Thus, do loop body. while is carried out at least once.

Example:

```
1.  <?
2.  $i = 1;
3.  do {
4.  $i echo. "br\n";
5.  } while ($i ++ <10);
6.  ?>
```

for

The cycle for - rather universal construction. It can look both it is simple, and is very tangled. Let's consider for a start classical option of its use:

```
1.  <?
2.  for ($i=1; $i <10; $i ++) {
3.  $i echo. "br\n";
4.  }
5.  ?>
```

As well as in the previous examples, this script displays in the browser of number from 1 to 9. Syntax of the cycle for generally such:

for (expression_1; expression_2; expression_3) where expression_1 it is carried out before accomplishment of a cycle, expression_2 - the condition of accomplishment of a cycle (while is similar), and expression_3 is carried out after each iteration of a cycle.

Got confused?;) Let's rewrite "the general case" of the cycle for in transposition on the cycle while:

for

while

for (expression _1; expression _2; expression_3) {

```
  body_cycle
}
expression_1;
while (expression_2) {
  body_cycle
  expression_3;
}
```

I hope, now everything is clear. Precisely it is clear? Then understand this cycle:

```
1.  <?
2.  $i=0;
3.  for ($i ++; - $i <10; $i+ =2) {
4.  $i echo. "br\n";
5.  }
6.  ?>
```

If long understood - nothing terrible the Cycle for is most often used in more clear form - as in the first example.

Break and continue operators. Nested loops

Can occur need of loop termination under the certain condition checked in a loop body. For this purpose the break operator whom we already met serves, considering switch.

```
1.  <?
2.  $i = 0;
3.  while (++ $i <10) {
4.  $i echo. "br\n";
```

```
5.  if ($i == 5) break;
6.  }
7.  ?>
```

This cycle will display only values from 1 to 5. At $i == 5 conditional statement of if in line 5 will work, and accomplishment of a cycle will stop.

The continue operator begins new iteration of a cycle. In the following example by means of continue output of number 5 "is missed":

```
1.  <?
2.  for ($i=0; $i <10; $i ++) {
3.  if ($i == 5) continue;
4.  $i echo. "br\n";
5.  }
6.  ?>
```

Break and continue operators it is possible to share with all types of cycles.

Cycles can be enclosed (as practically all in PHP): in one cycle other cycle, etc. can be located. Break and continue operators have the optional numerical parameter specifying what cycle in the order of an enclosure - including from below up from current position - they treat (actually, break are an abbreviated notation of break 1 - similarly and with continue). Example of an output from two cycles at once:

```
1.  <?
2.  for ($i=0; $i <10; $i ++) {
3.  for ($j=0; $j <10; $j ++) {
4.  if ($j == 5) break 2;
5.  echo of' $i='. $i.', $j='. $j. "br\n";
6.  }
```

```
7.  }

8.  ?>
```

Arrays

The array represents a set of the variables combined by one name. Each value of an array is identified by an index which is specified after a variable array name in square brackets. The combination of an index and the value corresponding to it is called an array cell.

```
1.  <?

2.  $i = 1024;

3.  $a [1] = 'abc';

4.  $a [2] = 100;

5.  $a ['test'] = $i - $a [2];

6.

7.  echo of $a [1]. "br\n";

8.  echo of $a [2]. "br\n";

9.  echo of $a ['test']. "br\n";

10.  ?>
```

In the given example, $a array cell with an index 1 appears in a line three; line abc value is appropriated to an array cell. The same line also $a array as this first mentioning of a variable of $a in the context of an array, the array is created automatically appears. In line 4 numerical value 100 is appropriated to an array cell with an index 2. In line 5 the value equal to a difference of $i and $a [2] be assigned to $a array cell with the line test index.

As you can see, the index of an array can be both number, and line.

In other programming languages (for example, Perl) the arrays having line indexes are called hashes (hash), and are separate data type. In PHP, in fact, all arrays are hashes, however both the line, and number can serve as an index.

In the previous example the array was created automatically at the description of the first array cell. But the array can be set and it is explicit:

```
1.  <?
2.  $i = 1024;
3.  $a = array (1=>'abc', 2=> 100, 'test '=> of $i-100);
4.  print_r ($a);
5.  ?>
```

The array of $a created in the last example is completely similar to an array from the previous example. Each array cell here is set in a look an index => value. During creation of the test element it was necessary to specify value 100 directly as this time we create an array "at one stroke", and values of its elements at a stage of creation are unknown to PHP.

In line 4 for output of value of an array we used the print_r function () which is very convenient for output of contents of arrays to the screen - first of all, for debugging.

Lines in output of the print_r function are divided by normal line feed \n, but not a tag
. For convenience of reading, the line print_r (.) it is possible to surround with operators of output of tags <pre>... </pre>:

```
echo of '<pre>';
print_r ($a);
echo' </pre>';
```

If obviously not to specify indexes, then the property of arrays of PHP characteristic of numerical array in other languages is shown here: the next element will have a serial numerical index. Numbering starts from scratch. Example:

```
1.  <?
2.  $operating_systems = array ('Windows', 'Linux', 'FreeBSD', 'OS/2');
3.  $operating_systems [] = 'MS-DOS';
4.
5.  echo of "<pre>";
```

```
6.  print_r ($operating_systems);
7.  echo" </pre>";
8.  ?>
```

Output:

```
Array
(
    [0] => Windows
    [1] => Linux
    [2] => FreeBSD
    [3] => OS/2
    [4] => MS-DOS
)
```

Here we obviously did not specify indexes: PHP automatically appropriated numerical indexes, since zero. When using such form of record the array can be touched by means of the cycle for. The quantity of array cells is returned by count operator (or its synonym, size of):

```
1.  <?
2.  $operating_systems = array ('Windows', 'Linux', 'FreeBSD', 'OS/2');
3.  $operating_systems [] = 'MS-DOS';
4.
5.  echo of '<table border=1>';
6.  for ($i=0; $i <count ($operating_systems); $i ++) {
7.  echo' <tr> <td>'. $i.' </td> <td>'. $operating_systems [$i].' </td> </tr>';
```

```
8.  }
9.  echo' </table>';
10.  ?>
```

Styles of record can be mixed. Pay attention to what indexes are automatically appropriated to PHP after installation of some indexes manually.

```
1.  <?
2.  $languages = array (
3.  1 => 'Assembler',
4.  'C ++',
5.  'Pascal',
6.  'scripting' => 'bash'
7.  );
8.  $languages ['PHP'] = 'PHP';
9.  $languages [100] = 'Java';
10.  $languages [] = 'Perl';
11.
12.  echo of "<pre>";
13.  print_r ($languages);
14.  echo" </pre>";
15.  ?>
```

Output:

```
Array
(
```

[1] => Assembler

[2] => C ++

[3] => Pascal

[scripting] => bash

[php] => PHP

[100] => Java

[101] => Perl

)

Foreach loop

Array, similar to previous, it is difficult to touch by means of for. For search of array cells the special for each loop is provided:

```
1.  <?
2.  $languages = array (
3.  1 => 'Assembler',
4.  'C ++',
5.  'Pascal',
6.  'scripting' => 'bash'
7.  );
8.  $languages ['PHP'] = 'PHP';
9.  $languages [100] = 'Java';
10. $languages [] = 'Perl';
11. ?>
```

```
12. <table>
13. <tr>
14. <the> Index </th>
15. <th> Value </th>
16. </tr>
17. <?
18. for each ($languages as of $key => $value) {
19. echo' <tr> <td>'. $key.' </td> <td>'. $value.' </td> </tr>';
20. }
21. ?>
22. </table>
```

This cycle works as follows: as emergence in a code of the program of array cells $languages, variable $key and $value be assigned respectively an index and value of the next element, and the loop body is carried out.

If indexes do not interest us, the cycle can be written as follows: for each ($languages as of $value).

Constructions of list and each

In addition to already considered construction of array, there is a construction of list supplementing it being some kind of antipode of array: if the last is used for creation of an array from a set of values, then list, on the contrary, fills the listed variables with values from an array.

Let's say we have $lang array = array ('php', 'Perl', 'basic'). Then construction of list ($a, $b) = $lang will appropriate to $a variable PHP value, and $b - 'perl'. Respectively, list ($a, $b, $c) = $lang will in addition appropriate $c = to 'basic'.

If in an array of $lang there was only one element, PHP would issue the note on lack of the second array cell.

And if we are interested not only in values, but also indexes? Let's use construction of each

which returns couples an index value.

```
1.  <?
2.  $browsers = array (
3.  'MSIE' => 'Microsoft Internet Explorer 6.0',
4.  'Gecko' => 'Mozilla Firefox 0.9',
5.  'Opera' => 'Opera 7.50'
6.  );
7.
8.  list ($a, $b) = each ($browsers);
9.  list ($c, $d) = each ($browsers);
10. list ($e, $f) = each ($browsers);
11. $a echo.': '. $b. "br\n";
12. $c echo.': '. $d. "br\n";
13. $e echo.': '. $f. "br\n";
14. ?>
```

At first sight the fact that in lines different values are appropriated to 8-10 variables though expressions to the right of an assignment sign absolutely identical can surprise. The matter is that each array has a hidden pointer of the current element. Initially he points to the first element. Construction of each advances the pointer on one element.

This feature allows to touch an array by means of the normal cycles while and for. Of course, earlier considered foreach loop is more convenient, and it is worth preferring it, but construction with use of each is quite widespread, and you can meet her in a set of scripts in network.

```
1.  <?
2.  $browsers = array (
```

```
3.  'MSIE' => 'Microsoft Internet Explorer 6.0',
4.  'Gecko' => 'Mozilla Firefox 0.9',
5.  'Opera' => 'Opera 7.50'
6.  );
7.
8.  while (list ($key, $value) =each ($browsers)) {
9.  $key echo. ':'. $value. "br\n";
10.  }
11.
12.  ?>
```

After end of a cycle, the pointer of the current element indicates the end of an array. If the cycle needs to be executed several times, the pointer should be reset compulsorily by means of reset operator: reset ($browsers). This operator sets the pointer of the current element in the beginning of an array.

We covered only the basics of arrays. In PHP there is a set of various functions of work with arrays; their detailed description is in appropriate section of documentation.

Constants

Unlike variables, value of a constant is established once and is not subject to change. Constants do not begin with the character of $ and decide on the help of define operator:

```
1.  <?
2.  define ('MY_NAME', );
3.
4.  echo 'My name is '. MY_NAME;
5.  ?>
```

Constants standard (and convenient) is optional to call uppercase letters, but this the

agreement.

As the name of a constant does not begin with any special character, in double quotes value of a constant cannot be placed (as there is no opportunity to distinguish where a constant name and where - just the text).

4. Forms

Keeping of the head

Printer-friendly version

1. HTML forms. Arrays of $ _GET and $ _POST
2. htmlspecialchars ()
3. phpinfo ()

HTML forms. Arrays of $ _GET and $ _POST

If you did not forget chapter 1 material yet, then it is hardly worth reminding: forms are the main way of data exchange between the Web server and the browser, that is provide user interaction - actually what and requires web programming.

For further reading a clear understanding of the web programming bases described in chapter 1 is obligatory. If you the beginner, I advise once again attentively to re-read it.

So, we will take already familiar to you on chapter 1 an example:

```
1. <html>
2. <body>
3. <?
4. if ($ _SERVER ['REQUEST_METHOD'] == 'POST') {
5. echo' <h1> Hi, <b>'. $ _POST ['name'].' </b> </h1>!';
6. }
7. ?>
8. <form method= "POST" action=" <?= $ _SERVER ['PHP_SELF']?>">
```

9. Enter your name: <input type= "text" name= "name">

10.

11. <input type= "submit" of name= "okbutton" value= OK>

12. </form>

13. </body>

14. </html>

The form given in lines 8-12 contains two elements: name and okbutton. The method attribute specifies a method of sending the POST form (see chapter 1), the action attribute specifying URL to which the form goes is filled with value of the server PHP_SELF variable - the address of the script which is carried out at present.

<?= $ _SERVER ['PHP_SELF']?> - contracted form of record for echo: <? echo $ _SERVER ['PHP_SELF'];?>.

Let's assume, in the field of name we entered value Bryan, and pressed the OK button. At the same time the browser sends a POST request to the server. Request body: name=okbutton=OK. PHP autocompletes an array of $ _POST:

$ _POST ['name'] = Bryan

$ _POST ['okbutton'] = 'OK'

Actually, Bryan value goes the browser in an urlencode-look; for the coding windows-1251 this value looks as %C2%E0%F1%FF. But, as PHP automatically carries out necessary decoding, we can "forget" about this feature - yet it is not necessary to work with HTTP requests manually.

As in a body of a request only names and values are entered, but not types of elements of forms, PHP does not know, there corresponds $ _POST ['name'] to entry line, the button, or the list. But this information, in general, is not necessary to us at all.

As know what is written on the submit button, to us it is optional, in line 11 it is possible to delete name attribute, having reduced the description of the button to <input type= "submit" of value= OK>. In this case, the browser will send name= POST request Bryan.

And now - the same, but for a GET form:

```
1.  <html>
2.  <body>
3.  <?
4.  if (isset ($ _GET ['name'])) {
5.  echo' <h1> Hi, <b>'. $ _GET ['name'].' </b> </h1>!';
6.  }
7.  ?>
8.  <form action=" <?= $ _SERVER ['PHP_SELF']?>">
9.  Enter your name: <input type= "text" name= "name">
10. <br>
11. <input type= "submit" of value= OK>
12. </form>
13. </body>
14. </html>
```

In line 8 it would be possible to write <form method= "GET"> equally well: GET - a method by default. This time the browser sends a GET request which is equivalent to input in an address bar of the address: http://address-name/script-name.php?name.

PHP with GET forms arrives in the same way as with POST, with that difference that is filled (guess from three times) an array of $ _GET.

Cardinal difference - in line 4. As simple input of the address in a line of the browser is a GET request, check of if ($ _SERVER ['REQUEST_METHOD'] == 'GET') is senseless: everything that we in this case will find out - what someone from not figs to do did not send a POST form to our script.;) Therefore we resort to construction of isset () which returns true if this variable is defined (i.e. value was appropriated to it), and false - if the variable is not defined. If the form was filled - as you already understood, PHP automatically appropriates to $ _GET ['name'] the corresponding value.

Way of check by means of isset () - universal, it would be possible to use also for a POST

form. Moreover, it is more preferable as allows to find out what fields of a form are filled.

In many old books and articles it is claimed that:

1. Data both from GET, and from POST forms get directly to variables (in our case - $name), and POST data are more priority, i.e. "rub clean" GET data;

2. Also data of GET and POST forms are stored respectively in arrays of $HTTP_GET_VARS and $HTTP_POST_VARS.

This information is very long time ago obsolete. Years three as. Web programming, and, in particular, PHP develops in high gear. Remember this moment not to be trapped with old books or scripts.

Inquisitive readers can receive particulars here, and also proceed here.

Little more difficult example.

```
1.  <html>
2.  <body>
3.  <?
4.  if (isset ($ _POST ['name'], $ _POST ['year'])) {
5.  if ($ _POST ['name'] ==' ') {
6.  echo 'Enter a name! <br>';
7.  } else if ($ _POST ['year'] <1900 || $ _POST ['year']> 2016) {
8.  'Specify by echo year of birth! Tolerance range of values: 1900. 2016 <br>';
9.  } else {
10.  echo 'Hello, '. $ _POST ['name'].'! <br>';
11.  $age = 2016 - $ _POST ['year'];
12.  echo 'to you '. $age.' years <br>';
```

```
13.  }
14.  echo' <hr>';
15.  }
16.  ?>
17.  <form method= "post" action=" <?= $ _SERVER ['PHP_SELF']?>">
18.  Enter your name: <input type= "text" name= "name">
19.  <br>
20.  Enter your year of birth: <input type= "text" name= "year">
21.  <input type= "submit" of value= OK>
22.  </form>
23.  </body>
24.  </html>
```

No new receptions are used here. Understand, execute a code, try to modify...

Let's change the last example that the user did not need to fill fields repeatedly. For this purpose we will fill value attributes of elements of a form with just entered values.

```
1.  <html>
2.  <body>
3.  <?
4.  $name = isset ($ _POST ['name'])? $ _POST ['name']:' ';
5.  $year = isset ($ _POST ['year'])? $ _POST ['year']:' ';
6.
7.  if (isset ($ _POST ['name'], $ _POST ['year'])) {
8.  if ($ _POST ['name'] ==' ') {
```

```
9.  echo 'Enter a name! <br>';
10.  } else if ($ _POST ['year'] <1900 || $ _POST ['year']> 2016) {
11.  'Specify by echo year of birth! Tolerance range of values: 1900. 2016 <br>';
12.  } else {
13.  echo 'Hello, '. $ _POST ['name'].'! <br>';
14.  $age = 2016 - $ _POST ['year'];
15.  echo 'to you '. $age.' years <br>';
16.  }
17.  echo' <hr>';
18.  }
19.  ?>
20.  <form method= "post" action=" <?= $ _SERVER ['PHP_SELF']?>">
21.  Enter your name: <input type= "text" name= "name" value=" <?= $name?>">
22.  <br>
23.  Enter your year of birth: <input type= "text" name= "year" value=" <?= $year?>">
24.  <input type= "submit" of value= OK>
25.  </form>
26.  </body>
27.  </html>
```

Lines 4 and 5 can appear a little unclear. Everything is very simple: X = A? B: C - abbreviated notation of a condition of if (A) X=B else X=C. The line 4 could be written so:

```
if (isset ($ _POST ['name']))
```

$name = $ _POST ['name'];

else

$name =' ';

The construction used in lines 21 and 23 <?= $foo?> - and it is simpler than that: this reduction for <? $foo echo?>.

There can be a question - why not to throw out a line 4-5 and not to write:

Enter your name: <input type= "text" name= "name" value=" <?= $ _POST ['name']?>">

Enter your year of birth: <input type= "text" name= "year" value=" <?= $ _POST ['year']?>">

The matter is that if these POST variables are not defined - and will be if the form was not filled yet, - PHP will give warnings of use of unassigned variables (and, it is quite reasonable: such message allows to find quickly hardly detectable typos in names of variables, and also warns about possible "holes" on the website). It is possible to place, of course, a code with isset... directly in a form, but it will turn out it is too bulky.

Understood? And now try to find an error in the given code. Well, not absolutely an error, - but a defect.

htmlspecialchars ()

Did not find? I will prompt. Enter, for example, in the field "name" a double quote and some text, for example, "123. Send a form, and look at the source code of the received page. In the fourth line there will be something like:

Enter your name: <input type= "text" of name= "name" value= "" 123">

That is - anything good. And if the cunning user entered a JavaScript-code?

For a solution of this problem it is necessary to use the htmlspecialchars function () which will replace sample digits with their HTML representation (for example, a quote - on "):

1. <html>
2. <body>
3. <?
4. $name = isset ($ _POST ['name'])? htmlspecialchars ($ _POST ['name']):' ';

```
5.  $year = isset ($ _POST ['year'])? htmlspecialchars ($ _POST ['year']):' ';
6.
7.  if (isset ($ _POST ['name'], $ _POST ['year'])) {
8.  if ($ _POST ['name'] ==' ') {
9.  echo 'Enter a name! <br>';
10.  } else if ($ _POST ['year'] <1900 || $ _POST ['year']> 2016) {
11.  'Specify by echo year of birth! Tolerance range of values: 1900. 2016 <br>';
12.  } else {
13.  echo 'Hello, '. $name.'! <br>';
14.  $age = 2016 - $ _POST ['year'];
15.  echo 'to you '. $age.' years <br>';
16.  }
17.  echo' <hr>';
18.  }
19.  ?>
20.  <form method= "post" action=" <?= $ _SERVER ['PHP_SELF']?>">
21.  Enter your name: <input type= "text" name= "name" value=" <?= $name?>">
22.  <br>
23.  Enter your year of birth: <input type= "text" name= "year" value=" <?= $year?>">
24.  <input type= "submit" of value= OK>
25.  </form>
26.  </body>
```

```
27. </html>
```

Repeat experiment and be convinced that now the HTML code is correct.

Remember - the htmlspecialchars function () needs to be used always when contents of a variable at which there can be HTML special characters are displayed.

phpinfo ()

The phpinfo function () - one of the most important in PHP. It outputs information on the PHP settings, values of various configuration variables...

Why I mention it in chapter devoted to forms? phpinfo () - the most convenient debugging tool. phpinfo (), in addition, displays values of all $ _GET, $ _POST and $ _SERVER - variables. So if the variable of a form "was lost" to find the easiest way in what business - to use the phpinfo function. In order that function displayed only variables values (and you had not to scroll ten pages), it should be caused as follows: phpinfo(INFO_VARIABLES); or - that they is absolute are phpinfo(32);.

Example:

```
1.  <html>
2.  <body>
3.  <form method= "post" action=" <?= $ _SERVER ['PHP_SELF']?>">
4.  Enter your name: <input type= "text" name= "name">
5.  <input type= "submit" of value= OK>
6.  </form>
7.  <?
8.  phpinfo(32);
9.  ?>
10. </body>
11. </html>
```

Or, for example, such situation: you want to learn the visitor's IP address. You remember that the corresponding variable is stored in an array of $ _SERVER, but - here an ill luck - forgot, how exactly the variable is called. Besides, we cause phpinfo(32); we look for the IP address in the plate and we find it - in the line _SERVER ["REMOTE_ADDR"].