Урок 1

Въведение в PHP като език за програмиране на уеб приложения. История и версии

1. Въведение в PHP като език за програмиране

Пълното название на PHP е Hypertext Preprocessor. Той е скриптов език за вграждане в HTML. По-голямата част от синтаксиса му е взета от C, Java и Perl с няколко специфични за PHP уникални свойсва, които са добавени. Целта на езика е да позволи да Web разработчиците да пишат лесно и бързо динамично генерирани страници.Той е един от най-популярните езици за програмиране в Интернет и популярността му расте непрекъснато.

Разпространява се под отворен лиценз (PHP License), който по своята същност е BSD лиценза и който позволява безплатно разпространяване на програмния код на интерпретатора на езика, както и създаването на производни интерпретатори под други лицензи с уговорката, че тези интерпретатори не могат да включват PHP в името си. Фактът, че PHP се разпространява свободно, го прави удачен избор за изграждане на Web-сървър базиран изцяло на свободни продукти - GNU/Linux, Apache, MySQL/PostgreSQL и др.

При поискване, кодът, който е написан на PHP се интерпретира от уеб сървъра на който е качен, и резултатът се връща на уеб браузърът. Потребителят не може да види чистият PHP код без да има достъп до самият файл в който той е записан. По този начин е помислено за сигурността.

Самият език е преносим на много изчислителни архитектури и операционни системи като GNU/Linux, UNIX, Mac OS X, Windows.

Съществуват множество модули (разширения) за PHP, които добавят различни функционалности и позволяват много по-бързо и ефективно разработване. Такива допълнителни функционалности към езика са:

- функции за обработка (създаване, редактиране ...) на изображения
- функции за работа с низове и регулярни изрази
- функции за работа с XML съдържание
- функции за работа със сокети (гнезда)
- функции за дата и час
- математически функции
- функции за управление на сесии и работа с бисквитки (cookies)
- функции за компресия и шифриране/дешифриране
- функции за COM и .NET за (Windows)
- функции за SOAP
- функции за работа с различни бази данни
- функции за работа с принтер
- функции за създаване на приложения с графичен потребителски интерфейс, базирани на библиотеката GTK
- функции за изпращане на e-mail съобщения
- хранилище за разширения и приложения на PHP: PEAR

PHP може да работи с повечето модерни бази данни - MySQL, PostgreSQL, Microsoft SQL Server, Oracle, SQLite и д.р.

На официалния сайт на PHP се намира обширна библиотека с информация за езика и модулите му, която може да се използва както за основно запознаване с езика, така и като справочник по време на работата с него.

Поради отворения характер на езика съществуват множество потребителски групи в България и по света посветени на програмирането с PHP, където всеки може да получи помощ в работата си с този език.

2. Кратка история на PHP

Началото на PHP е поставено от Размус Лердорф през 1994 г., въпреки че самият Лердорф смята 1995-та за рождената година на PHP, защото тогава "отваря" кода на новия език за програмиране. В началото Размус създава Perl скриптове за следене на посещенията на резюмето на личната му страница, които впоследствие пренаписва и така се създава първата версия на **PHP** (тогава означаващо **P**ersonal **H**ome **P**age). Към средата на 1995 функционалността на езика е подобрена и той започва да се нарича PHP/FI (съкращение от Personal Home Page/Forms Interpreter), което всъщност е втората версия на езика (PHP 2).

С течение на времето все повече ентусиасти започват да се включват в проекта на Лердорф и езика набира все по-голяма популярност. Около средата на 1997 PHP е бил ползван в около 50 000 уеб сайта. По същото време двамата израелски студенти на Technion (израелски технологичен институт) Зеев Сураски (Zeev Suraski) и Анди Гутманс (Andi Gutmans) пренаписват целия език, констатирайки множеството бъгове и отчасти неправилни концепции за строеж на програмен език и по-късно известяват Размус с техните предложения. Той ги приема и така се създава основният екип от разработчици (PHP Core team), който по-късно се превръща в групата на PHP (PHP group). Зеев Сураски и Анди Гутманс заедно с Размус Лердорф добавят нови функционалности и API за допълнителни модули към езика и така се създава **PHP 3**. От този момент нататък PHP започва да се превръща в най-популярния език за динамични уеб разработки. Сураски и Гутманс пренаписват отново ядрото на езика и създават Zend engine през 1999 г. Пренаписването е направено основно от Сураски и Гутманс като курсов проект по време на тяхното обучение в Technion. Пренаписването е в следствие на факта, че кодът на PHP2 е бил труден за поддръжка и доста нестабилен.

През 2000 година се появява версия **PHP 4.** По това време PHP вече е бил използван в изграждането на около 5 милиона сайта. Версията **PHP 5** е пусната през 2004 г. , а **PHP 6** през 2006 г. През 2005 г. цифрата на сайтовете използващи PHP е близо 25 милиона.

Основната база данни с информация за езика в интернет е официалния сайт www.php.net

3. Как работи PHP? Първа програма на PHP

Преди да започнете да се занимавате с PHP е силно препоръчително да имате поне базови познания по HTML. PHP кода може да се пише директно в HTML документа, т.е. между html таговете. Също така html тагове могат да се включват в PHP кода. Това прави от езика изключително гъвкав инструмент за манипулиране на уеб документи.

За да напишете някакъв html документ не ви е нужно нищо друго освен текстов редактор и браузър. След като напишете кода браузъра веднага ще покаже html документа. Като кликнете на View -> Source от горното меню на браузъра ще можете веднага да видите html кода на страницата.

За да може да се изпълни един php скрипт обаче са необходими освен браузър още два компонента: сървър и специален софтуер - php интерпретатор. Това е така, тъй като за разлика от html, който се изпълнява в браузъра, PHP се изпълнява на сървъра. Затова ако разполагате само с браузър няма да можете да стартирате никакъв php скрипт.

Схемата на работа на php е следната: посетител отваря някаква php страница в интернет и браузъра изпраща заявка към сървъра. PHP скрипта се изпълнява на сървъра и връща резултата на браузъра. Като краен резултат посетителя вижда на монитора си изобразеното на php страницата. Това което вижда посетителя обаче е обикновен html документ. Ако отвори сорса на страницата в него посетителя няма да види нито един php елемент, нищо друго, освен html код. PHP кода, който е генерирал страницата в браузъра, остава напълно скрит от очите на посетителя. Ето един прост пример, чрез който ще покажем това нагледно.

По стара традиция първата програма на някакъв език извежда на екран съобщението "Hello, World!" (Здравей, свят!).

Един от най-важните елементи на PHP е командата **echo**, чрез която може да се покаже някакъв текст и други елементи на екрана на монитора, като текста който ще се показва трябва да е сложен в кавички и инструкцията да завършва с точка и запетая. Вътре в php кода може да се вмъкват html тагове - това е един от факторите, които правят php толкова гъвкав. Т.е., след като използвате командата echo, за да покажете текста на екран, вие можете допълнително да зададете ефекти на този текст чрез html тагове, вложени директно в съдържанието на конструкцията echo.

Както знаете, HTML документите имат начален таг <html> и краен таг </html>, като цялото съдържание на html страницата се разполага между тези два тага. По същия начиин и PHP скриптовете имат начален и краен таг, между които се помества съдържанието на скрипта.

Началния (отварящ) таг е **<?php**
Крайния (затварящ) таг е **?>**

Да направим така, че на екрана да се появи наклонен текст "Здравей свят!". Отворете някакъв текстов редактор, например Notepad, и напишете в него следния код:

```
<html>
<head>
<title>Тест</title>
</head>
<body>
<?php
echo "<i>Здравей, свят!</i>";
?>
</body>
</html>
```

след което съхранете файла с разширение php, например test.php, а след това качете страницата някъде в интернет на хостинг, който поддържа PHP. Сега можете да отворите сорса на страницата като ползвате бутоните View -> Source от горното меню на браузера.

В сорса на страницата test.php няма да се вижда нищо друго освен следния html код:

```
<html>
<head>
<title>Тест</title>
</head>
<body>
<i>Здравей, свят!</i>
</body>
</html>
```

Съответно това, което ще получите като резултат на екрана, ще бъде: *Здравей, свят!*

Вижда се, че ако желаете просто да покажете на екран някакво съобщение, е напълно излишно да ползвате PHP скрипт, тъй като това много по-лесно може да направите с добре познатите HTML тагове.

Урок 2

Инсталиране и конфигуриране на сървър (Apache).
Инсталатори на AMP. Тест на сървъра

1. Какъв софтуер е нужен, за да работи PHP?

Както вече споменахме, за да стартирате един PHP скрипт ви е нужен сървър и софтуера на езика - PHP интерпретатор.

PHP интерпретатора се използва със сървъра Apache, като обикновено към тях е присъединена и системата за управление на бази данни MySQL. "Троицата" Apache/PHP/MySQL е най-широко използваната в интернет конфигурация за разработване на динамични уеб страници.

Всички хостинг планове на Host.bg включват PHP и MySQL, както и phpMyAdmin - софтуер за работа с MySQL бази данни.

Ако желаете да работите с тези програми на своя личен компютър е необходимо да си изтеглите и да инсталирате сървъра Apache, PHP интерпретатора и MySQL базата данни. Това може да направите от съответните сайтове:

- www.apache.org
- www.php.net
- www.mysql.com
- www.phpmyadmin.net

Но вместо да теглите тези продукти един по един, да ги настройвате и да ги конфигурирате за съвместна работа, може да си изтеглите пакета XAMPP, дистрибуция на Apache, който съдържа в себе си всичко необходимо - Apache, MySQL, PHP, phpMyAdmin, FTP и др. елементи и е настроен за лесна инсталация. XAMPP има версии за работа под Linux, Windows и други ОС.

Повече информация за XAMPP и връзки за сваляне на софтуера ще намерите на следния адрес:

http://www.apachefriends.org/en/xampp.html

2. Инсталиране и конфигуриране на сървър (Apache)

След сваляне на XAMPP започнете инсталацията, като следвате указанията дадени на всяка стъпка.

След като изтеглите и инсталирате софтуера на компютъра си той ще се намира в папка **xampp** и пътя до него на компютъра ви ще бъде например C:\Program Files\xampp\ (ако xampp е разположен в папка Program Files на диск C:).

В папка xampp ще видите папка **htdocs** - това е вашата основна директория, в която трябва да помествате файловете и папките си. След като даден файл, например myscript.php, е създаден и съхранен в папка htdocs ще можете да го достъпвате и виждате в браузъра през адрес localhost, т.е. адреса му в браузъра ще бъде

http://localhost/myscript.php

а пътя до него (до сорса на файла) на машината ще бъде

C:\Program Files\xampp\htdocs\myscript.php

Ако в папка htdocs направите например папка dir1 и в нея сложите файл test1.php, адреса за достъп до файла в браузъра ще бъде

http://localhost/dir1/test1.php

а пътя до сорса на файла ще е

C:\Program Files\xampp\htdocs\dir1\test1.php

и т.н.

3. Тест на сървъра

Урок 3

PHP вграждане в HTML, тагове, стилове, интервал и коментари

1. PHP – Синтаксис

Синтаксис - правилата, които трябва да се спазват, за да се напише правилно структурирн код.

Някои основни положения от синтаксиса на PHP бяха споменати в по-горните материали, а именно:

- PHP скриптовете имат начален таг **<?php** и краен таг **?>**
- текст написан след командата **echo** и затворен в кавички се показва на екрана
- всяка конструкция в PHP завършва с точка и запетая
- html таговете могат да се вмъкват в php кода

- **начален таг <?php и краен таг ?>**

Синтаксиса и семантиката на PHP наподобяват тези на повечето езици за програмиране (C, Java, Perl) с допълнението, че всеки PHP код се съдържа в етикет. Всеки PHP код трябва да е разположен в следното...

PHP код:

```
<?php
?>
```

или стенографичния PHP таг, който изисква поддръжката на стенографичен запис да е активирана на вашия сървър...

```
<?
?>
```

Ако пишете PHP скриптове и планирате да ги разпространявате, предполага се, че ще използвате стандартната форма (която включва ?php) вместо стенографичната. Така ще сте сигурни, че скрипта ви ще работи, дори стартиран на друг сървър с различни настройки.

- **точка и запетая ";"**

Този символ обозначава края на PHP конструкция и никога не трябва да се забравя. Примерно, ако повторим кода за "Hello World!" няколко пъти ще трябва да поставим точка и запетая след всяко повторение.

PHP и HTML код:

```
<html>
<head>
<title>My First PHP Page</title>
</head>
```

```
<body>
<?php
echo "Hello World! ";
echo "Hello World! ";
echo "Hello World! ";
echo "Hello World! ";
echo "Hello World! ";
?>
</body>
</html>
```

Показва:
Hello World! Hello World! Hello World! Hello World! Hello World!

- **празно пространство (интервал)**

Както е в HTML, интервалите се игнорират в PHP конструкциите. Това означава, че е ОК да имате един ред PHP код, след него 20 реда празно пространство преди следващия ред от PHP кода. Може също да използвате табулация, за да направите по-прегледни големи файлове с много код. PHP интерпретатора ще игнорира и тях като празни пространства.

PHP и HTML код:

```
<html>
<head>
<title>My First PHP Page</title>
</head>
<body>
<?php
echo "Hello World!";
echo "Hello World!";
?>
</body>
</html>
```

Показва :
Hello World!Hello World!

Това е абсолютно изряден PHP код.

- **поставяне на кавички**

По отношение на кавичките в PHP съществува следната особеност - израза след командата echo, който искаме да се покаже на екрана, трябва да е затворен в кавички, но ако израза е ограден в двойни кавички и в същото време решим да сложим кавички и на някоя дума вътре в израза, PHP ще ни върне съобщение за грешка. Например ако напишем

```
<?php
echo "Тези думи ще "изскочат" на екран";
?>
```

софтуера няма да разчете израза, тъй като ще приеме кавичката пред думата "изскочат" за край на инструкцията и ще търси завършващите точка и запетая, каквито разбира се там няма, тъй като това не е края на израза. За да е възможно на екрана да се изобразят кавичките, употребени вътре в инструкцията, е необходимо пред тях да се сложи наклонена наляво черта, т.е. правилния код ще бъде:

```
<?php
echo "Тези думи ще \"изскочат\" на екран";
?>
```

С други думи, кавичките вътре в php инструкциите могат да се покажат ако са във формат \"

Същото правило важи и за стойностите на атрибутите в xhtml тагове, където според изискванията на XHTML трябва да се поставят кавички. Например ако искаме да изобразим центрирано заглавие чрез тага h1 и неговия атрибут align, който да е със стойност center правилния код ще бъде:

```
<?php
echo "<h1 align=\"center\">Това заглавие е центрирано</h1>";
?>
```

докато кода с обикновени кавички ще върне грешка, защото ще бъде неправилен:

```
<?php
echo "<h2 align="center">Това заглавие е центрирано</h2>";
?>
```

2. Коментари

Коментарите в PHP са подобни на коментарите, използвани в HTML. Коментарите в PHP винаги започва с последователност от специални символи и целия текст, който попада между началото и края на коментара ще бъде игнориран от браузъра.

В HTML главната функция на коментарите е да служат като бележки, за разработчика или за други, които искат да разгледат изходния код на страницата ви. Но коментарите в PHP се различават то тях по това, че те не се показват на потребителите или посетителите на сайта ви. Единствения начин да видите PHP коментарите е да отворите PHP файла за редакция. Това прави PHP коментарите полезни само за PHP програмистите. В случай, че сте забравили как изглеждат HTML коментарите, вижте примерапо-долу.

HTML код:

```
<!-- This is an HTML Comment -->
```

- **Синтаксис на коментарите в PHP: коментар на един ред**

Докато в HTML има само един вид коментри, PHP има два. Първия, който ще разгледаме е коментар на единичен ред. Той казва на PHP интерпретатора да игнорира всичко, което е написано на въпросния ред до десния край на реда. За да коментирате един ред се използва "//" и останалата част от текста се игнорира от PHP интерпретатора.

PHP код:

```
<?php
echo "Hello World!"; //This will print out Hello World!
echo "<br>Psst...You can't see my PHP comments!";
//echo!"nothing";
// echo "My name is Humperdinkle!";
?>
```

Показва:

Hello World!
Psst...You can't see my PHP comments!

Забележете, че две от нашите echo конструкции не се изпълниха, защото ги игнорирахме с коментар за един ред. Този вид коментиране често се използва за бързи бележки относно сложен и объркващ код или просто за временно игнориране на ред от PHP code.

- **Синтаксис на коментарите в PHP: многоредов коментар**

Подобно на HTML коментар, многоредовия PHP коментар се използва за игнориране на големи блокове от кода или за писане на по-големи бележки за поясняването му. Многоредовия PHP коментар започва с " /* " и завършва с " */ ".

PHP код:

```
<?php
/* This Echo statement will print out my message to the
the place in which I reside on.  In other words, the World.
*/
echo "Hello World!";
/* echo "My name is Humperdinkle!";
echo "No way! My name is Uber PHP Programmer!";
*/
?>
```

Показва: **Hello World!**

ПРИМЕР ЗА КОМЕНТАРИ В PHP

// Това е коментар на един ред

Това също е коментар на един ред

/* Това е коментар
разположен на
няколко реда */

Добри практики в коментирането: една от добрите практики, която може да се препоръча на начинаещи в PHP програмирането е.... ИСПОЛЗВАЙТЕ ГИ!!! Толкова много хора пишат сложен PHP код и са често твърде мързеливи, за да напишат хубави коментари или пък вярват, че коментари и пояснения не са нужни. Обаче наистина ли вярвате, че ще си спомните точно какво сте си мислили пишейки това, когато погледнете кода си след година или повече?Ако кодът ви е богат на коментари ще сте си благодарни и щастливи PHP програмисти в бъдеще. Използвайте коментари за един ред само за бързи бележки, за тънки и хитри места в кода си, а коментари на няколко реда, за да опишете нещо много по-задълбочено отколкото с кратка бележка.

3. Как да запазвате PHP страници?

Ако имате вкаран PHP код в HTML страница, която сте създали и искате Web браузъра да я интерпретира правилно, тогава трябва да запазите файла с **.php** разширение, вместо стандартното .html разширение. Уверете се че сте запазили файла си пправилно. Вместо index.html той трябва да бъде **index.php** ако в него има PHP код.

Пример : проста HTML & PHP страница.

По-долу е даден пример за най-лесната PHP и HTML страничка, която може да създадете и която следва всички Web стандарти.

PHP и HTML код:

```
<html>
<head>
<title>My First PHP Page</title>
</head>
<body>
<?php
echo "Hello World!";
?>
</body>
</html>
```

Показва:
Hello World!

Ако запазите този файл и го сложите на сървър, който поддържа PHP и го заредите в web браузъра си, трябва да видите да се показва "Hello World!". Ако не се покаже, моля проверете дали сте написали примера правилно.

Използвахме PHP функцията **echo**, за да изпишем "Hello World!" и ще я разгледаме много по-задълбочено в PHP функциите както и много други като нея по-късно в това ръководство.

Урок 4

Типове данни

PHP поддържа няколко различни типа променливи. Типа на дадена променлива се определя от вида на нейната стойност, която може да бъде:

- **integer** - цяло число (например 50)
- **floating-point** - число с плаваща запетая (например 3.14)
- **boolean** - логическа (булева) стойност - вярно (истина) и невярно (неистина)
- **string** - поредица от символи (низ)

A. Типа на променлива **integer** може да бъде каквото и да е цяло число, без значение дали е с положителен или отрицателен знак. Примери за такива променливи са:

```
<?php
$celochislena_promenliva1 = 102;
$celochislena_promenliva2 = 12895;
$celochislena_promenliva3  = -37;
$celochislena_promenliva4  = 1;
?>
```

B. Типа на променлива **floating-point** е число с плаваща (десетична) запетая, без значение колко са цифрите преди и след запетаята. В PHP кода вместо запетая се използва точка.

Примери за такива променливи са:

```
<?php
$chislo_s_plavashta_zapetaia1 = 3.14;
$chislo_s_plavashta_zapetaia2 = 178.54098361;
?>
```

C. Типа на променлива **boolean** може да приема само две стойности - true (вярно, истина) и false (невярно, неистина), например:

```
<?php
$logicheska_stoinost1 = true;
$logicheska_stoinost2 = false;
?>
```

D. Типа на променлива **string** може да съдържа поредица символи, например някаква дума или няколко думи и цифри. В даден низ (string) може да бъдат включени и имена на други променливи. Низовете трябва да се затварят в двойни (") или единични кавички ('). Стойността на стринг, затворена в двойни кавички, автоматично бива проверяван дали не съдържа имена на променливи и ако в стринга бъде открито име на променлива, то бива заместено със стойността на тази променлива:

```
<?php
$malko_ime = "Иван";
$familia = "Иванов";
$trite_imena = "$malko_ime Петров $familia";
echo "$trite_imena";
?>
```

Горния код ще изведен на екран трите имена Иван Петров Иванов.

Освен това на една променлива може да се зададе изобщо да няма никаква стойност. Това става като и се присвои стойността null:

```
<?php
$promenliva_bez_stoinost = null;
echo "Няма никаква стойност: $promenliva_bez_stoinost";
?>
```

На екрана ще се вижда само Няма никаква стойност:

При PHP има една важна особеност във връзка с типовете на променливите. В много програмни езици преди да бъде употребена една променлива е необходимо изрично да се укаже от какъв тип е тя. В PHP няма нужда предварителни да се декларира типа на променливата, тъй като софтуера на PHP автоматично го разпознава в момента на задаването на стойността на дадена променлива.

E. тип променливи - **array** (масив). Докато на описаните по-горе типове променливи се задават единични стойности, типа array представлява променлива, на която могат да се зададат едновременно множество стойности. Тя се ползва обикновено когато е нужно да се съхранят няколко близки по значение стойности, обединени под общо название. Например масив с название $avtomobili може да съдържа стойностите "Мерцедес", "BMW", "Форд" и т.н.. Този тип ще разгледаме в подробности в урока за масивите.

Създаването на масив става чрез конструкцията array(), като отделните стойности се отделят чрез запетаи. Синтаксиса е следния:

```
$promenliva = array("стойност1", "стойност2", "стойност3");
```

Тъй като на една и съща променлива са присвоени едновременно няколко стойности, всяка от тези стойности автоматично получава пореден номер. Съответно всяка от стойностите на променливата е достъпна (например за извеждане на екрана) чрез името на променливата, съчетано с номера на стойността. Затова при работа със стойностите на масива индентификационните номера за съответните стойности трябва да се изписват, като се поставят непосредствено след името на променливата и трябва да бъдат затворени в големи квадратни скоби - []. Това е нужно, за да може PHP да "разбере" която точно от многото стойности на масива имаме предвид. Началния номер започва от нула. Т.е. ако искаме да изведем на екран първата стойност на нашата примерна променлива $avtomobili трябва да напишем следния код:

```
<?php
$avtomobili = array("Мерцедес", "BMW", "Форд");
echo "$avtomobili[0]";
?>
```

В случая първата стойност на масива е "Мерцедес", затова тя автоматично получава индентификационен номер нула - [0]. Резултата от горния код ще бъде показане на екран на стойността Мерцедес. За да работим със следващата стойност трябва да напишем $avtomobili[1] и т.н.

Друг начин на дефиниране на променлива от тип array е като всяка стойност се зададе поотделно на променливата. В този случай при всяко присвояване на стойност на променливата може за прегледност да се укаже и индентификационния номер. Синтаксисът на такъв тип дефиниране на array променлива е следния:

```
$promenliva[0] = "стойност1";
$promenliva[1] = "стойност2";
$promenliva[2] = "стойност3";
```

Дори ако не бъдат указани изрично номера на стойностите те ще им бъдат присвоени по подразбиране, т.е. ако напишем

```
<?php
$avtomobili[] = "Мерцедес";
$avtomobili[] = "BMW";
$avtomobili[] = "Форд";
echo "$avtomobili[0]";
?>
```

ще получим като резултат извеждане на думата "Мерцедес", тъй като на първата стойност на масива автоматично е бил присвоен индентификационен номер 0.

Освен чрез цифри стойностите на масивите могат да бъдат идентифицирани и чрез буквени наименования, например думи или съкращения. Всяка от думите с които са идентифицирани различните стойности от масива се превръща в "ключ" за съответната стойност. Обръщането към дадена стойност става като се изпише името на масива, следвано от ключа за стойността, който също трябва да е затворен в квадратни скоби, например:

```
<?php
$podpis[ime] = "Иван";
$podpis[prezime] = "Петров";
$podpis[familia] = "Иванов";
echo "$podpis[prezime]";
?>
```

Горния код ще изведе на екран стойността на масива $podpis която е с ключ "prezime", т.е. "Петров". По-подробна информация за работа с масивите е дадена в материалите за PHP масиви.

Урок 5

Променливи и област на видимост. Променлива от тип променлива

1. Какво е променлива? Имена и синтаксис на променливи

Променливата е метод за съхрнение на величина, като текстов низ "Hello World!" или числовата стойност 4. Променливата може да бъде преизползвана в целия ви код на местата, където трябва да пишете точната стойност отново и отново. В PHP променливите се дефинират в следната форма:

$promenliva_ime = Stojnost;

Ако пропуснете символа за долар в началото - няма да работи. Това е често срещана грешка на новаците в PHP програмирането!

Бърз пример за променлива: Да речем, че искаме да съхраним стойностите, за които говорихме в горния параграф. Как да направим това? Първо трябва да направим променлива с някакво име и после да и зададем стойността, която искаме. Примерът по-долу показва правилния начин да направим това.

PHP код:

```
<?php
$hello = "Hello World!";
$a_number = 4;
$anotherNumber = 8;
?>
```

Бележка за програмистите: PHP не изисква променливите да се декларират преди да се инициализират.

Има няколко правила, които трябва да спазвате, когато избирате имена за вашите PHP променливи а именно:

- PHP променливите трябва да започват с буква или подтире "_".
- PHP променливите могат да се наименоват само със символите от a-z, A-Z, 0-9, или _ .
- Променливи, в които има повече от една дума, думите могат да се разделят с подтирета. $my_variable
- Променливи с повече от една дума могат да се разграничат и с главна буква. $myVariable

Променливите са едни от най-важните програмни конструкции както в PHP така и в останалите езици. Те представляват нещо като "контейнери" или "сандъци" в които се съхраняват различни стойности - цифри и думи.

За да бъде създадена една променлива на нея първо трябва да и се измисли някакво название, например някаква дума или съкращение и т.н. След това на променливата може да бъде зададена (присвоена) някаква стойност.

На променливите може да задавате имена (названия) каквито вие си изберете, т.е. вие ще решавате какво да бъде името на дадена променлива. При избора на име

обаче трябва да имате предвид, че в PHP има известно количество запазени думи, които не могат да бъдат използвани като имена на променливи (а също и като имена на функции). Списък със запазените думи на езика може да видите на следния адрес:

http://www.php.net/manual/en/reserved.php

Променливите се отличават от останалите конструкции на езика по това, че винаги започват със символа на долара: **$**. Следователно обичайния вид на променливите е: **$promenliva**

Имената на променливите може да съдържат големи и малки латински букви, арабски цифри и долна черта, като може да започват само с буква или долна черта (след символа на долара, който винаги е най-отпред). Т.е. името на променливата може да съдържа цифри, но не може да започва с цифра. Например $ime123, $ime_123 и $_123ime са валидни имена на променливи, докато $123ime е невалидно название.

Има значение дали изписвате имената на променливите с големи или малки букви. Т.е. една и съща дума написана само с малки букви, с начална главна буква или само с главни букви, се възприема като 3 различни имена на променливи, например $ime, $Ime и $IME са 3 различни променливи.

2. Задаване (присвояване) на стойност на променлива

След като сте измислили название на променливата е време да й зададете някаква стойност. На променливите могат да се задават както цифрови, така и буквени стойности (например някакъв текст). След задаването на някаква стойност може да си представяте променливата като склад или кутия, където зададената стойност се пази до момента, когато стане необходимо да бъде използвана.

Както подсказва названието им - променливи - "складираното" в дадена променлива съдържание може да се променя по време на изпълнение на скрипта. С променливите може да се извършват различни операции - например те могат да бъдат сравнявани помежду си, за да се установи дали стойностите им са еднакви или различни, като на основа на резултата от извършеното сравнение в скрипта се задава да бъде изпълнено определено действие; променливите могат също да бъдат комбинирани, като в един израз се наредят няколко променливи или с тях да се извършват различни аритметични действия (сбор, умножение и т.н.). Резултата от извършеното действие може да бъде присвояван като стойност на друга променлива.

Стойностите на променливите, независимо дали са цифрови или текстови, им се присвояват чрез оператора за присвояване на стойност, който е "равенство" (=). В някои случаи стойността която се присвоява на променливата трябва да е затворена в кавички, а цялата инструкция винаги трябва да завършва с точка и запетая. Задаването на стойност на една променлива изглежда така:

$promenliva = "стойност";

По отношение на кавичките, в които се затварят стойностите на променливите, има някои особености, които ще бъдат изяснени по-нататък.

При разчитането на подобен код не трябва да се казва "$promenliva е равно на "стойност". Правилния израз ще бъде "На променливата $promenliva е присвоена стойността "стойност". Например ако имаме кода **$a = 30;** това не означава, че a е равно на 30. При разчитането на този код трябва да кажем "На променливата a е присвоена стойността 30". Може да си представяте променливата **$a** като сандък, върху който е изписана буквата **a** и в който са изсипани за съхранение 30 единици стойност.

Вече знаете, че командата echo извежда на екран израза, който следва след нея (затворен в кавички и завършващ с точка и запетая). Чрез echo може да се покаже на екран и стойността на дадена променлива, ако преди това променливата е декларирана (създадена)

чрез задаване на нейната стойност, а след това името на променливата е поставено в израза, който echo извежда на екрана.

Например може да се напише следния код:

```
<html>
<head>
<title>Пример за употреба на променливи</title>
</head>
<body>

<?php
// Присвояване стойност на променлива
$moeto_ime = "Иван";
// Извеждане на стойността на променливата на екран
echo "Аз се казвам <b>$moeto_ime</b>.";
?>
</body>
</html>
```

Резултата от горния код в браузъра ще бъде: Аз се казвам **Иван**. В случая създадохме променливата $moeto_ime, на която присвоихме стойност "Иван".

Понякога се налага на няколко променливи да бъде присвоена една и съща стойност. Това може да стане с един единствен израз по следния начин:

```
<?php
$promenliva1 = $promenliva2 = $promenliva3 = "стойност";
?>
```

По този начин и на трите променливи е зададена еднаква стойност.

3. Прости операции с променливи

Комбинация от променливи може да бъде зададена като стойност на трета променлива, например така:

```
<?php
$malko_ime = "Иван";
$familia = "Иванов";
$cialo_ime = "$malko_ime $familia";
echo "Казвам се $cialo_ime";
?>
```

Горния код ще изведе на екрана изречението Казвам се Иван Иванов.

Същия резултат ще получите и от следния код:

```
<?php
$malko_ime = "Иван";
$familia = "Иванов";
```

```
echo "Казвам се $malko_ime $familia";
?>
```

Досега задавахме буквени стойности на променливите. Нека да зададем цифрови стойности на две променливи и след това да извършим с тях някаква аритметична операция.

Например:

```
<?php
$parva_cifra = 2;

$vtora_cifra = 3;

$sbor = $parva_cifra + $vtora_cifra;

echo "$sbor";

?>
```

В случая зададохме стойностите 2 и 3 съответно на променливите $parva_cifra и $vtora_cifra, след което извършихме сбор на стойностите чрез оператора за събиране, който е "плюс" (+), като присвоихме новата стойност на променливата $sbor. Накрая изведохме на екран резултата, който е 5.

Забележете, че в инструкцията **$sbor = $parva_cifra + $vtora_cifra;** не са използвани кавички. Слагането на кавички определя възприемането на израза именно като израз знак по знак (като поредица от символи - низ), а не като аритметичното действие, което искаме да извършим. Ако бяхме поставили кавички (**$sbor = "$parva_cifra + $vtora_cifra";**) резултата щеше да е не сбора от 2 и 3, който е 5, а самия израз "2 + 3". Повече по този въпрос ще бъде казано в материалите за типовете на променливите и операторите за сравнение.

Можем да извършваме сбор не само на стойностите на две променливи, но също на стойността на дадена променлива с дадена цифра, например така:

```
<?php

$parva_cifra = 2;

$sbor = $parva_cifra + 3;

echo "$sbor";

?>
```

Резултата от горния код отново ще бъде **5**.

4. Предварително дефинирани променливи: Променливи на средата

В PHP съществуват и предварително дефинирани (вградени в езика) променливи, които могат да се използват в скриптовете без да е необходимо преди това да бъдат декларирани. Сред предварително дефинираните променливи има няколко, които са наречени "променливи на средата" или "променливи на обкръжението" (environment variables). Чрез тези променливи може да се извлече всевъзможна информация за посетителя на една страница - от кой уеб адрес е дошъл, какъв е IP адреса му, каква операционна система и браузър ползва и т.н.

Най-използваните такива променливи са следните:

$HTTP_USER_AGENT - дава информация за браузъра и ОС
$REMOTE_ADDR - дава информация за IP адреса
$SERVER_SOFTWARE - дава информация за сървъра
$HTTP_REFERER - дава информация за URL откъдето идва юзера

Можете да впечатлите някои от посетителите на страницата си като сложите в нея код подобен на следния:

```
<?php
echo "Вашия IP адрес е <b>$REMOTE_ADDR</b>, сървъра ви е
<b>$SERVER_SOFTWARE</b>, браузъра ви и операционната система са
<b>$HTTP_USER_AGENT</b>, идвате от <b>$HTTP_REFERER</b>";
?>
```

При отваряне на страницата ви посетителя ще види данните на своята машина и местоположение, например нещо от типа:

Вашия IP адрес е **84.128.68.0**, сървъра ви е **Apache/1.3.33 (Unix) PHP/4.3.10 mod_ssl/2.8.22 OpenSSL/0.9.7d**, браузъра ви и операционната система са **Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)**, идвате от **http://www.domain.com**

Резултат от последната променлива $HTTP_REFERER ще има само ако посетителя е отворил страницата ви чрез кликване върху връзка към нея, поставена в някакъв друг уеб сайт.

Разбира се освен за впечатляване на посетителите тези променливи се използват и за събиране на важна статистическа информация. Пълен списък с променливите на средата може да видите като използвате вградената функция phpinfo(). Направете си php страница в която сложете следния код

```
<?php
phpinfo();
?>
```

В изведената информация ще видите списък на променливите на средата със съответните им стойности, показването на характеристиките на сървъра и PHP софтуера, с който работи вашия компютър.

5. Променлива от тип променлива

Понякога е удобно да можем да имаме променливи имена на променливи. Това означава - име на променлива, което може да бъде установявано и използвано динамично. Обикновената променлива се установява посредством израз като:

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

Променливата променлива взима стойността на променлива и разглежда тази стойност като име на променлива. В горния пример *hello* може да бъде използвано за име на променлива с помощта на два доларови знака. Т.е.

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

В този момент са дефинирани и съхранени две променливи в символното дърво на PHP: *$a* със съдържание "hello" и *$hello* със съдържание "world". Следователно, следният израз:

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

ще изведе абсолютно същото като:

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

т.е. и двата извеждат: hello world.

За да използвате променливи променливи с масиви, трябва да разрешите една двусмисленост. А именно, когато напишете *$$a[1]*, синтактичният анализатор трябва да знае дали сте искали да използвате *$a[1]* като променлива, или сте искали *$$a* като променлива и [1] като индекс от тази променлива. Синтаксисът за разрешаване на тази двусмисленост е: *${$a[1]}* за първия случай и *${$a}[1]* за втория.

Предупреждение: Моля забележете, че променливите променливи не могат да бъдат използвани със свръхглобалните масиви на PHP вътре във функции или методи на клас. Освен това променливата *$this* е специална променлива, която не може да бъде достъпвана динамично.

Урок 6

Какво е константа? Дефиниране на константа - синтаксис

1 Какво е константа?

Константите, подобно на променливите, представляват конструкции на езика в които може да се съхраняват стойности от различни типове. Основната разлика между константи и променливи е, че, както подсказва самото им название, константите не могат да бъдат променяни след като веднъж са били дефинирани и им е била присвоена някаква стойност. Тази стойност не може да бъде нито увеличавана, нито намалявана, нито заменяна, нито анулирана.

2 Дефиниране на константи

Константите се дефинират и им се присвоява стойност чрез функцията **define()**. Синтаксиса на функцията е следния:

define(име на константа, стойност на константа)

Т.е. в кръглите скоби на функцията се записват последователно името на константата и нейната стойност, отделени със запетая.

Пред имената на константите не се поставя символ за долар или някакъв друг символ.

Имената на константите могат да бъдат каквито пожелаете, т.е. името го избирате вие. При избор на името трябва да спазвате същите правила както при избор на име на променлива - названието на константа може да съдържа латински букви, арабски цифри и символа долно тире (_), като може да започва с буква или с тире, но не и с цифра.

По подразбиране има значение дали названията на константите са изписани с големи или малки букви, т.е. konstanta, Konastanta и KONSTANTA са три различни имена на константи. Това положение може да бъде променено с използване на ключовата дума true, както е обяснено в примерите по-долу.

Не е задължително, но е прието имената на константите да се пишат с главни букви.
Стойностите на константите могат да бъдат само от следните типове:

- целочислени стойности

- стойности с плаваща запетая
- булеви стойности (вярно / невярно)
- низове (поредици от символи)

Друга разлика между константите и променливите е, че константите по подразбиране имат глобален обхват на действие, т.е. те са достъпни на всяко място без да е необходимо да бъдат декларирани като глобални чрез ключовата дума global.

Пример за дефиниране на константа:

```
<?php
define(KONST1, "Така се дефинират константи");
echo KONST1;
?>
```

Горния код ще изведе съобщението "Така се дефинират константи", тъй като това е стойността, която сме задали на константата KONST1. Понеже става дума за стойност от тип string (низ), тя е затворена в кавички.

Както беше споменато по-горе, по подразбиране има значение дали названието на константата е с малки или големи букви. Например ако напишем кода

```
<?php
define(KONST1, "Така се дефинират константи");
echo konst1;
?>
```

върнатия резултат ще бъде konst1.

Има начин названията на константите да станат нечувствителни към употребата на малки и големи букви. За целта е необходимо след стойността на константата да добавим ключовата дума true. В такъв случай кода

```
<?php
define(KONST1, "Така се дефинират константи", true);
echo konst1;
?>
```

ще ни върне като резултат съобщението "Така се дефинират константи", тъй като вече няма значение дали името на константата се изписва с малки или с големи букви.

Урок 7

Предварително дефинирани константи. Магически константи

Ще споменем накратко още някои особености на константите в PHP. Подобно на предварително дефинираните функции и променливи, PHP поддържа и широк набор от предварително дефинирани константи, т.е. константи които няма нужда да бъдат декларирани чрез функцията define(), а могат направо да се използват в скриптовете. Списък с тези константи може да видите в официалния сайт www.php.net

Съществуват и 5 специални предварително дефинирани константи, наречени "магически константи" (magical constants), които не са константи в пълния смисъл на думата, тъй като могат да променят своята стойност в зависимост от констекста в който са използвани. "Магическите константи" са следните:

__FILE__ указва пълния път до текущия файл
__LINE__ указва текущия ред във файла
__FUNCTION__ указва име на функция
__CLASS__ указва име на клас
__METHOD__ указва име на метод

Например константата __FILE__ съдържа като стойност името и пълния път до текущия файл и ще има различна стойност в зависимост от това с какъв файл се работи в момента и къде се намира той. Ако примерно работите с файл на своята машина, който се нарича test.php и се намира в папка htdocs, която се намира в папка xampp, която е в папка Program Files на диск C:, тогава кода

```
<?php
echo __FILE__;
?>
```

ще ни върне като резултат пълния път до файла, а именно

C:\Program Files\xampp\htdocs\test.php

Урок 8

Какво е оператор? Видове оператори. Приоритет на оператори

1. Какво е оператор?

Операторите представляват конструкции на езика, чрез които се извършват различни операции с променливите като аритметични операции (събиране, изваждане, умножение, деление), присвояване на стойности на променливи, сравняване на променливи и пр. В материалите за променливите ние вече употребихме оператора за присвояване на стойност, който е равенство (=) и оператора за събиране, който е плюс (+).

2. Видове оператори

а) Аритметични оператори

+ събиране

- изваждане

***** умножение

/ деление

. съединяване на низове /нарича се "конкатенация на низове"/

% деление по модул - резултата, който се връща, е целочисления остатък от делението на двете променливи

Ясно е, че с първите четири оператора могат да се събират, изваждат и т.н. стойностите на дадени променливи. Например да зададем на променлива $prom1 стойност 100 и на $prom2 стойност 50 и да извършим някаква операция с тях:

```
<?php
// Пример за събиране
$prom1 = 100;
$prom2 = 50;
$prom1 = $prom1 + $prom2;
echo "$prom1";
?>
```

Горния код ще върне резултат 150 (100 + 50 = 150), тъй като увеличихме стойността на $prom1 с 50 единици като я събрахме със стойността на $prom2.

Същия резултат ще получим и от операцията

```
$prom1 = $prom1 + 50;
```

при

```
$prom1 = $prom1 * $prom2;
```

и при

```
$prom1 = $prom1 * 50;
```

резултата ще е 5000 (умножение 100 по 50) и т.н.

По-особен е оператора за съединяване (конкатениране) на низове, който е точка (.) Той не извършва пресмятане, а просто съединява двете стойности, така че ако напишем

```
$prom1 = $prom1 . $prom2;
```

резултата ще бъде 10050, т.е. "снаждане" на 100 с 50.
Кода

```
$prom1 = $prom1 % $prom2;
```

ще върне резултат нула, тъй като това е целочисления остатък при делене 100 на 50.

b) Оператори за присвояване

= присвояване на стойност
+= едновременно сбор и присвояване на стойност
-= едновременно изваждане и присвояване на стойност
*= едновременно умножение и присвояване на стойност
/= едновременно деление и присвояване на стойност
.= едновременно конкатениране и присвояване на стойност
%= едновременно деление по модул и присвояване на стойност

Вече познавате оператора за присвояване на стойност (=). Той има няколко разновидности, които позволяват да се извърши някаква аритметична операция и едновременно с това новата стойност да се присвои на дадена променлива. Нека да зададем отново на променливата $prom1 стойност 100 и да извършим следната операция:

```
<?php
// Пример за едновременно събиране и присвояване на стойност
$prom1 = 100;
$prom1 += 50;
```

```
echo "$prom1";
?>
```

Резултата от горния код ще бъде 150, тъй като с операцията **$prom1 += 50;** увеличаваме стойността на променливата $prom1 с 50 единици, така че тя става 150 и едновременно с това присвояваме новата стойност на $prom1. Т.е. този код е напълно аналогичен на кода

$prom1 = $prom1 + $prom2;

където $prom2 е със стойност 50 или на операцията

$prom1 = $prom1 + 50;

Съответно операцията

$prom1 *= 50;

ще ни даде резултат 5000 и е напълно аналогична с

$prom1 = $prom1 * $prom2; // Където $prom2 е 50

или

$prom1 = $prom1 * 50;

и т.н.

c) Оператори за сравнение

Следващия тип оператори в PHP са операторите за сравнение. Както подсказва наименованието им, те се използват да сравняват стойностите на дадени променливи - дали тези стойности са равни, дали едната е по-голяма от другата и пр. и резултата който връщат може да приема само две стойности - вярно (истина) или невярно (неистина). Можете да си представяте тези оператори във вид на въпроси с които питате: Едната стойност (отляво) по-голяма ли е от другата стойност (отдясно)? Едната стойност равна ли е на другата? Едната стойност различна ли е от другата? и т.н.

Ако резултатите се изведат на екран, тогава, ако резултата е "вярно", той се изобразява чрез единица (1), а ако е "невярно" се изобразява чрез нула (празна стойност).

== равно ли е на - връща 1 (вярно) ако стойностите са равни
!= различно ли е от - връща 1 (вярно) ако стойностите са различни
<> различно ли е от - подобно на горния оператор !=
< по-малко ли е от - връща 1 (вярно) ако стойността отляво е по-малка
> по-голямо ли е от - връща 1 (вярно) ако стойността отляво е по-голяма
<= по-голямо или равно ли е - връща 1 (вярно) ако стойността отляво е по-малка или равна на дясната
>= по-малко или равно ли е - връща 1 (вярно) ако стойността отляво е по-голяма или равна на дясната
=== и стойността и типа на променливите ли са равни - връща 1 (вярно) ако не само стойностите, но и типовете на дадени променливи са еднакви

Ще изясним действието на операторите за сравнение с няколко прости примера. Нека да имаме променливите $prom1 и $prom2 и да им зададем съответно стойности 100 и 50, а след това да сравним тези стойности с някой от операторите за сравнение, например с оператора "по-голямо ли е от" (>):

```
<?php
$prom1 = 100;
```

```
$prom2 = 50;
$sravnenie = ($prom1 > $prom2);
// Питаме: $prom1 по-голямо ли е $prom2?
// Отговорът е: да, тъй като 100 е по-голямо от 50
echo "<b>Резултата от сравнението е $sravnenie</b>";
?>
```

След като се изпълни горния код на екрана ще получим: **Резултата от сравнението е 1**. Тъй като използваме за сравнение оператора "по-голямо ли е от" (>) с този код ние все едно питаме "100 по-голямо ли е от 50?" и тъй като 100 е по-голямо от 50, то резултата от сравнението е "истина" и се изобразява на екрана като единица. Ако напишем:

```
$sravnenie = ($prom1 == $prom2);
```

то резултата ще бъде празна стойност (неистина), тъй като използваме за извършване на сравнението оператора "равно ли е на" с което все едно питаме "100 равно ли е на 50?" и понеже 100 не е равно на 50 получаваме резултат "невярно".

По същия начин действат и останалите оператори за сравнение. Обърнете внимание, че инструкцията за сравнение се затваря в кръгли скоби - ().

По-особено е единствено действието на последния оператор от списъка, който се изобразява чрез 3 равенства (===). Това е оператора, който сравнява не само стойностите на променливите, но и типа им и връща верен резултат само ако и стойностите и типовете съвпадат. В случая става дума за следната особеност на PHP: вече беше обяснено, че в PHP, за разлика от някои други езици, не е необходимо предварително да се декларира от какъв тип е дадена променлива, тъй като софтуера на езика автоматично определя типа на променливите според начина по който са зададени техните стойности. Освен това беше указано, че когато стойността на дадена променлива е поредица от символи, т.е низ (тип string), то е задължително низа да бъде ограден с единични или двойни кавички. Това означава, че в зависимост от това дали дадена стойност на променлива е оградена в кавички или не софтуера на PHP може да я разпознае като два различни типа променлива. Например нека да зададем на променливата $prom1 стойност 100: **$prom1 = 100;**

Зададена по този начин променливата има тип integer, т.е. цяло число, понеже 100 е цяло число. Ако обаче оградим цифрата 100 в кавички стойността на променливата ще си остане 100, но типа й вече ще бъде string (низ). В такъв случай кода

```
<?php
$prom1 = 100;
$prom2 = "100";
$sravnenie = ($prom1 === $prom2);
echo "Резултата от сравнението е $sravnenie";
?>
```

ще ни върне като резултат "невярно" (празна стойност), понеже стойностите на променливите са еднакви, но типовете им са различни.

d) Логически оператори

Логическите оператори са подобни на операторите за сравнение по това, че също се използват за сравняване на един компонент с друг и също връщат като резултат някоя от двете булеви стойности, т.е. вярно или невярно. Логическите оператори обаче сравняват само цели изрази, в които са използвани операторите за сравнение и са върнали като резултат true или false. Т.е. логическите оператори сравняват не отделни променливи, а само изрази и то притежаващи булева стойност (истина или неистина).

Списък на логическите оператори:

&& логическо И
and логическо И (подобно на &&)

|| логическо ИЛИ
or логическо ИЛИ (подобно на ||)

! логическо отрицание (логическо НЕ)

xor логическо изключващо ИЛИ

Когато работим с логическия оператор "логическо И" (&&) ние все едно питаме: "И левия И десния израз ли са верни?". Т.е. този оператор връща 1 (вярно) ако и двата израза от двете му страни връщат резултат true, например

```
<?php
$prom1 = 100;
$prom2 = 50;
$log_sravnenie = (($prom1 >$prom2) && ($prom1 != $prom2));
echo "Резултата от логическото сравнение е $log_sravnenie";
?>
```

Върнатия резултат от горния код ще бъде 1 (вярно). Това е така, защото първо имаме две сравнения с операторите за сравнение:

($prom1 > $prom2) с което питаме "$prom1 по-голямо ли е от $prom2?"

и

($prom1 != $prom2) с което питаме "$prom1 различно ли е от $prom2?"

И двете сравнения с операторите за сравнение ще ни върнат резултат true, тъй като 100 е по-голямо от 50 и 100 е различно от 50. Следователно и в левия и в десния израз ще имаме резултат "вярно".

След това сравняваме тези два израза като питаме "И двата израза ли са верни?". Понеже в случая и двата израза са верни, то като изведем на екран стойността на променливата $log_sravnenie ще получим резултат 1 (вярно).

Забележете, че сравненията с операторите за сравнение са затворени в кръгли скоби, както е затворен в такива скоби и целия израз - сравнението с логическия оператор.

По подобен начин работят и останалите оператори, а именно:

Оператора "логическо ИЛИ" (||) все едно пита: "Верен ли е поне левия ИЛИ поне десния израз?" и връща резултат 1 (вярно) ако поне един от изразите е верен.

"Логически изключващото ИЛИ" (xor) връща резултат "вярно" ако само единия от двата израза е верен, но не и когато и двата са верни (тогава връща неистина - false). Т.е. с този оператор все едно питаме: "Само едния от двата израза е верен, нали?" и ако отговора е "Да", тогава резултата е "истина".

Особеност има само при "логическото НЕ" (!). Този оператор се използва не с два, а с един израз, като се поставя пред израза. Връща резултат "вярно" ако изследвания израз е неверен и обратно, т.е. с логическото отрицание все едно питаме: "Този израз НЕ е верен, нали?" и ако отговора е положителен, т.е. "Да, не е верен", тогава резултата е "истина":

```
<?php
$prom1 = 100;
```

```
$log_sravnenie = !($prom1 > 500);
echo "Резултата от логическото сравнение е $log_sravnenie";
?>
```

Горния код ще върне резулат 1 (вярно), тъй като в израза все едно питаме "Не е вярно, че 100 е по-голямо от 500, нали?". И понеже 100 наистина не е по-голямо от 500, то резултата от сравнението с оператора за логическо отрицание е true.

3. Приоритет на оператори

Тук трябва да спомена за една важна особеност на операторите - всички оператори имат предварително определен приоритет на изпълнение. Това означава, че независимо в какъв ред подредите няколко оператора, те ще се изпълнят не според това кой от тях сте написали по-напред (по-вляво в кода), а според приоритета си. Например операторите за умножение и деление имат по-висок приоритет на изпълнение от операторите за събиране и изваждане. Нека да напишем следния код:

```
<?php
$izchislenie = 100 - 50 * 20;
echo "Резултата от изчислението е $izchislenie";
?>
```

При този случай по-напред (по-вляво) е зададен оператора за изваждане (-), а след него е оператора за умножение (*). Ако изпълнението на кода следваше тази логика, тогава резултата би трябвало да бъде 1000, тъй като 100 минус 50 е 50, умножено по 20 е 1000. Резултата от този код обаче ще бъде -900, защото пръв по приоритет е оператора за умножение, т.е. първото действие е 50 по 20, което е 1000, а след това е изваждането 100 минус 1000, което е -900. Ако желаете да получите резултата 1000 трябва да напишете кода в следния вид:

```
<?php
$izchislenie = (100 - 50) * 20;
echo "Резултата от изчислението е $izchislenie";
?>
```

Т.е. трябва да използвате кръглите скоби по начина по който те се ползват за решаване на задачи със скоби. Ако два или повече оператора имат един и същ приоритет, едва в такъв случай те се изпълняват отляво надясно според мястото си в кода.

Урок 9

Оператори за инкрементиране и декрементиране

С понятието **"инкрементиране"** се обозначава увеличаването на стойността на дадена променлива с една единица. Оператора за инкрементиране се записва чрез два плюса: ++

С понятието **"декрементиране"** се обозначава намаляването на стойността на дадена променлива с една единица. Оператора за декрементиране се записва чрез два минуса: --

Съществуват два вида оператори за инкрементиране и декрементиране - префиксни и постфиксни (за краткост пре- и пост-):

++$promenliva префиксно инкрементиране

--$promenliva префиксно декрементиране

$promenliva++ постфиксно инкрементиране

$promenliva-- постфиксно декрементиране

Т.е. пре- операторите се различават от пост- операторите по мястото, където се поставят знаците ++ и --.

Действието и на двата оператора за инкрементиране (пре- и пост-) се изразява в добавяне на единица към стойността на променливата, към която са прикрепени.

Действието и на двата оператора за декрементиране (пре- и пост-) се изразява в отнемане на една единица от стойността на променливата, към която са прикрепени.

Разликата между пре- и пост- операторите е, че при пре- операторите първо се извършва аритметичната операция (увеличаване или намаляване на стойността на променливата с единица) и след това се връща резултата, а при пост- операторите първо се връща резултат (стойността на променливата), а едва след това стойността се увеличава или намалява с единица.

Пример за пре-инкрементиране:

```
<?php
$prom1 = 100;
$prom1 = ++$prom1;
echo "Резултата от пре-инкрементирането е $prom1";
// Връща резултат 101
?>
```

Горния код ще върне резултат 101, тъй като чрез оператора за префиксно инкрементиране добавихме единица към стойността на променливата $prom1.

Същия резултат ще получим и от кода:

```
<?php
$prom1 = 100;
++$prom1;
echo "Резултата от пре-инкрементирането е $prom1";
// Връща резултат 101
?>
```

Абсолютно същия резултат като от горния код ще получим ако използваме в същия контекст и оператора за пост-инкрементиране:

```
<?php
$prom1 = 100;
$prom1++;
echo "Резултата от пост-инкрементирането е $prom1";
// Връща резултат 101
?>
```

Ако обаче заместим пре-инкрементирането от първия пример с пост-инкрементирането от горния пример, тогава резултата няма да е 101, а 100:

```
<?php
$prom1 = 100;
$prom1 = $prom1++;
echo "Резултата от пост-инкрементирането е $prom1";
// Връща резултат 100
?>
```

Това е така, защото при пре-инкрементиране първо се увеличава стойността на променливата с единица и след това се връща резултата, а при пост-инкрементиране първо се връща резултат (стойността на променливата), а едва след това стойността се увеличава с единица. За да може да получим резултата 101 в горния контекст трябва да напишем:

```
<?php
$prom1 = 100;
$prom1++;
$prom1 = $prom1++;
echo "Резултата от пост-инкрементирането е $prom1";
// Връща резултат 101
?>
```

Операторите за декрементиране работят по аналогичен начин.

- Операторите за инкрементиране и декрементиране не оказват влияние върху булеви стойности (истина и неистина) и не извеждат никакъв резултат.
- Стойностите на променливи съставени от букви могат да бъдат инкрементирани, но не могат да бъдат декрементирани.
- Стойността null не може да бъде декрементирана, но може да бъде инкрементирана, при което се връща резултат 1.

Урок 10

Условни (контролни) конструкции if , if-else и switch. Условен (троен) оператор (?)

В PHP съществуват конструкции, чрез които не само да определите дали даден израз е верен или не (дали връща като резултат "истина" или "неистина"), но и да зададете на скрипта да извърши определени операции в зависимост от върнатия резултат.

1. Условна конструкция if

Условната конструкция if се използва за проверка на истинността на даден израз и задаване на последващо действие, като синтаксисът й е:

if (израз с оператор за сравнение) {направи това}

При тази конструкция все едно казвате: "Ако е изпълнено даденото условие, тогава направи ето това".

Забележете, че израза, в който се задава условието с оператора за сравнение, се поставя в кръгли скоби - (), а операцията, която трябва да се задейства ако е изпълнено условието, е поставена в големи скоби - {}.

Такъв вид конструкция връща като резултат булева стойност от израза в кръглите скоби, чиято истинност се проверява, т.е. върнатия резултат е "вярно" или "невярно". Ако резултата е "вярно", тогава се изпълнява операцията, зададена в големите скоби {}.

Например нека да имаме следния код:

```
<?php
$prom1 = 100;
$prom2 = "100";
if ($prom1 == $prom2)
{
$rezultat = "Двете променливи са с еднакви стойности";
echo "$rezultat";
}
?>
```

Горния код ще изведе на екран съобщението "Двете променливи са с еднакви стойности", тъй като чрез скрипта ние все едно даваме следното указание: "Провери дали са равни стойностите на променливите $prom1 и $prom2 и АКО са равни изведи на екран съобщението "Двете променливи са с еднакви стойности".

Може и да не се използва променливата $rezultat, а директно да се зададе извеждане на надписа чрез echo:

```
if ($prom1 == $prom2)
{echo "Двете променливи са с еднакви стойности";}
```

В случая резултата от сравнението е "вярно", тъй като двете променливи имат еднакви стойности, макар че са различни по тип.

2. Условна конструкция if-else

Резонно възниква въпроса какво ще стане ако резултата от проверката не е "истина". Например да заменим оператора за равенство (==) с оператора за едновременно сравнение на стойност и на тип (===):

```
<?php
$prom1 = 100;
$prom2 = "100";
if ($prom1 === $prom2)
{
$rezultat = "Двете променливи са с еднакви стойности";
echo "$rezultat";
}
?>
```

Този код няма да ни върне никакъв резултат, защото условието за да се покаже на екрана въведения текст не е изпълнено - запитването е дали променливите са еднакви не само по стойност, но и по тип, а по тип те се различават. Освен това ние не сме указали какво да се прави в такъв случай.

Ясно е, че в такава ситуация е нужно някакво алтернативно решение - код, който да се изпълни ако поставеното условие не е истина.
За такива случаи е предвидено да се използва конструкцията if-else, чиито синтаксис е:

```
if (израз с оператор за сравнение) {направи това}
else {направи друго}
```

Т.е. чрез тази конструкция ние все едно казваме: "Ако поставеното условие е изпълнено (връща резултат "вярно") - направи това, но ако то не е изпълнено (връща резултат "невярно") - тогава направи ето това (друго действие)".

Ако резултата от проверката в израза след if е отрицателен, тогава се прескача кода заключен в големите скоби след if и се изпълнява кода в големите скоби след else. В такъв случай нашия примерен скрипт може да има вида:

```
<?php
$prom1 = 100;
$prom2 = "100";
if ($prom1 === $prom2)
{
$rezultat = "Двете променливи са с еднакви стойности";
echo "$rezultat";
}
```

```
else
{
echo "Променливите се различават или по стойност или по тип";
}
?>
```

Горния код ще върне като резултат надписа "Променливите се различават или по стойност или по тип", тъй като резултата от проверката е отрицателен, което означава, че се прескача показването на стойността на променливата $rezultat и директно се изпълнява кода, заключен в големите скоби след else.

Забележете, че след кръглите скоби, в които е затворен израза за проверка (веднага след if), не се поставят точка и запетая, тъй като това не е край на инструкция.

!!! elseif / else if

elseif, както подсказва и името й, е комбинация от *if* и *else*. Също както *else*, тя разширява конструкцията *if* така, че да изпълни различна инструкция, в случай че първоначалният израз *if* се е изчислил на **FALSE**. За разлика от *else* обаче тя ще изпълни този алтернативен израз само ако условният израз *elseif* се изчислява на **TRUE**. Например, следният код ще изведе a е по-голямо от b, a е равно на b или a е по-малко от b:

```
<?php
if ($a > $b) {
   echo "a е по-голямо от b";
} elseif ($a == $b) {
   echo "a е равно на b";
} else {
   echo "a е по-малко от b";
}
?>
```

В рамките на една конструкция *if* може да има много изрази *elseif*. Първият израз *elseif* (ако има такъв), който се изчисли на **TRUE** ще бъде изпълнен. В PHP можете също да напишете и 'else if' (с две думи) и поведението му ще бъде идентично на 'elseif' (с една дума). Синтактичното значение е малко по-различно (ако сте запознати със C, тук е налице същото поведение), но последният ред е този, който и в двата случая ще има абсолютно същото поведение. Конструкцията *elseif* се изпълнява само ако предшестващият израз *if* и всички предшестващи изрази *elseif* се изчислят на **FALSE** и текущият израз *elseif* се е изчислил на **TRUE**.

Забележка: Забележете, че *elseif* и *else if* ще бъдат разгледани като напълно еднакви, единствено когато се използват къдрави скоби, както в горния пример. Когато използвате двоеточие, за да дефинирате вашите условия *if/elseif*, не трябва да разделяте *else if* на две думи, в противен случай PHP ще се провали със синтактична грешка.

```
<?php
/* Неправилен метод: */
if($a > $b):
   echo $a." е по-голямо от ".$b;
else if($a == $b): // Няма да се компилира.
   echo "Горният ред предизвиква синтактична грешка.";
```

```
endif;
/* Правилен метод: */
if($a > $b):
   echo $a." е по-голямо от ".$b;
elseif($a == $b): // Забележете комбинацията от думи.
   echo $a." е равно на ".$b;
else:
   echo $a." не е по-голямо или равно на ".$b;
endif;
?>
```

Например:

```
<?php
$Num = 50;
if (($Num > 9 ) && ($Num < 100))
{
echo $Num, "е двуцифрено число";
}
elseif ($Num >= 100)
{
echo $Num, "Има три или повече цифри";
}
else
{
echo $Num, "е едноцифрено число";
}
?>
```

В горния пример на променливата $Num се присвоява стойност 50 и се проверяват различни условия. Ако стойността на $Num е между 9 и 100, се извежда съобщението "$Num, е двуцифрено число". Ако стойността е по-голяма от 100, се извежда съобщението $Num, има три или повече цифри". В противен случай съобщението е "$Num, е едноцифрено число". Както отбелязахме по-рано можете да имате само една клауза else в конструкцията if. Обаче можете да имате множество клаузи else if в една конструкция if. Това е показано в следващия пример>

```
<?php
$Num = 50;
if (($Num > 9 ) && ($Num < 100))
{
echo $Num, "е двуцифрено число";
}
elseif (($Num > 99) && ($Num < 1000 )
{
echo $Num, "е трицифрено число";
}
elseif (($Num > 9999) && ($Num < 10000))
{
echo $Num, "е четирицифрено число";
}
elseif ($Num >= 10000)
{
echo $Num, "Има пет или повече цифри";
}
else
{
echo $Num, "е едноцифрено число";
}
?>
```

3. Конструкция switch

Конструкцията switch може да спести многократното повторение на конструкция if-else. Switch има синтаксис, който позволява да се проверят множество условия едно след друго и в зависимост от върнатия резултат да се укаже изпълнение на някакво действие. Ключови думи и символи в конструкцията switch са:

- думата **case**, с която се задава израз за проверка
- **двоеточие :**, след което следва действие за изпълнение
- думата **break**, с която се прекъсва изпълнението

Switch дава възможност да се укаже и действие по подразбиране чрез думата default, което да се изпълни в случай, че нито едно от зададените условията не е изпълнено.

Например нека да имаме двете променливи от горния пример със стойности съответно 100 и "100". Ще зададем да бъде последователно проверено:

- дали тези променливи са равни и по стойност и по тип
- дали са равни поне по стойност

Кода ще бъде следния:

```
<?php
$prom1 = 100;
$prom2 = "100";
switch(true)
{
case $prom1 === $prom2: echo "Равни по стойност и тип";
break;
case $prom1 == $prom2: echo "Равни поне по стойност";
break;
default: echo "Не са равни нито по стойност нито по тип";
```

```
}
?>
```

На горния скрипт му е дадена следната задача:

- ако е вярно, че променливата $prom1 е равна на променливата $prom2 и по стойност и по тип, тогава изведи съобщението "Равни по стойност и тип" и прекъсни изпълнението (break). Ако това условие не е изпълнено продължи нататък.
- ако двете променливи не са равни и по стойност и по тип, но са равни поне по стойност, тогава изведи съобщението "Равни поне по стойност" и прекъсни изпълнението.
- ако двете променливи не са равни нито по стойност нито по тип, тогава изведи стандартно съобщение (default) за всички случаи когато условията не са изпълнени - "Не са равни нито по стойност нито по тип".

В този случай изведеното съобщение ще бъде "Равни поне по стойност".

Ако например зададем на $prom1 стойност 50, тогава ще се изпълни действието по подразбиране, указано след default, което е предвидено за всички случаи, когато нито едно условие не е върнало резултат "истина", т.е. тогава съобщението ще бъде "Не са равни нито по стойност нито по тип" и т.н.

4. Условен (троен) оператор

Измежду операторите има един, който е подходящо да бъде представен тук, тъй като той може да замести целия блок на конструкцията if-else. Това е един от най-ползваните оператори.

Оператора се нарича "троен оператор", тъй като едновременно извършва 3 действия:

1) Определя дали даден израз връща "истина" или "неистина"
2) Ако връща "истина" задава изпълнение на определено дейстие
3) Ако връща "неистина" задава изпълнение на алтернативно действие

От описанието става ясно, че тройния оператор е нещо като по-къс вариант на конструкцията if-else. Изписва се чрез въпросителния знак (?)

Синтаксиса на тройния оператор е следния:

израз ? ако е вярно направи това : ако е невярно направи друго

където:

- "израз" е израза с оператор за сравнение, който се проверява дали ще върне резултат "вярно" или "невярно"
- "ако е вярно направи това" е кода, който трябва да се изпълни ако резултата е "вярно"
- "ако е невярно направи друго" е кода, който трябва да се изпълни ако резултата е "невярно"

Ключовите моменти от синтаксиса на тройния оператор са въпросителната (знака на самия оператор), която се поставя след израза, чиято истинност ще се проверява и двоеточието, което се поставя между указанията за операцията в случай, че резултата е бил "вярно" и указанията за операцията в случай, че резултата е бил "невярно".

За пример ще използваме същия код от примера за конструкцията if-else:

```
<?php
$prom1 = 100;
$prom2 = "100";
```

```
if ($prom1 === $prom2)
{
$rezultat = "Двете променливи са с еднакви стойности";
echo "$rezultat";
}
else
{
echo "Променливите се различават или по стойност или по тип";
}
?>
```

Ако трябва да получим същия резултат, но чрез използване на тройния оператор, тогава трябва да напишем кода:

```
<?php
$prom1 = 100;
$prom2 = "100";
$rezultat = $prom1 === $prom2 ?
"Двете променливи са с еднакви стойности" :
"Променливите се различават или по стойност или по тип";
echo "$rezultat";
?>
```

В случая извършваме проверка за истинност на израза $prom1 === $prom2, т.е. дали е вярно, че двете променливи са еднакви и по стойност и по тип, задаваме две възможности - при резултат "вярно" и при резултат "невярно" и едновременно с това присвояваме тези възможности на променливата $rezultat. По този начин променливата $rezultat приема някоя от двете стойности в зависимост от това дали резултата е true или false. Накрая чрез echo показваме резултата.

Аналогичен резултат ще ни даде и кода:

```
<?php
$prom1 = 100;
$prom2 = "100";
echo $prom1 === $prom2 ?
"Двете променливи са с еднакви стойности" :
"Променливите се различават или по стойност или по тип";
?>
```

Тук имаме само сравнение между $prom1 и $prom2, задаване на съответните операции при резултат "истина" и "неистина" и извеждане на резултата на екран чрез echo, без стойностите да се присвояват на трета променлива.

Урок 11

Какво е цикличен оператор? Конструкции while, do-while и for

Цикъла е конструкция, която ви позволява да повтаряте отново и отново даден набор от инструкции дотогава, докато зададеното в цикъла условие връща резултат "истина". Вие може да определите точно числото на повторенията или броя повторения може да зависи от определени едно или повече условия.

1. Конструкция while

Конструкцията с ключова дума while има следния синтаксис:

while (израз за проверка) {прави това}

Т.е. с тази конструкция все едно казваме: "Докато е вярно това трябва да правиш ето това". Цикъла while борави с булеви резултати (истина и неистина). Докато израза в кръглите скоби, чиято истинност се проверява, връща резултат true (истина), дотогава ще бъде изпълняван кода, указан в големите скоби. В момента в който резултата от условието за проверка стане false (неистина) изпълнението на цикъла се прекъсва.

Да разгледаме следния пример:

```
<?php
$prom1 = 100;
$prom2 = 50;
while ($prom2 < $prom1)
{
$prom2++;
echo "$prom2<br />";
}
?>
```

В горния код е дадено за изпълнение следното:

- Присвоени са стойностите 100 и 50 съответно на променливите $prom1 и $prom2
- Зададено е условието - докато стойността на $prom2 е по-малка от стойността на $prom1 прибавяй по една единица към стойността на $prom2
- Резултата от промяната на променливата $prom2 е изведен на екран.

Визуалния резултат, генериран от горния код, ще бъде поредицата числа от 51 до 100, като за по-голяма прегледност чрез
 е зададено всяко отделно число да е на отделен ред. Т.е. при всяко изпълнение на цикъла към стойността на променливата $prom2 е прибавяна по единица, докато стойностите на двете променливи не се изравнят.

2. Конструкция do-while

Синатсисът на конструкцията while е такъв, че ако зададеното в кръглите скоби условие върне резултат "неистина" още при първата проверка за истинност, тогава кода в големите скоби изобщо няма да се изпълни. Понякога обаче е необходимо поне едно изпълнение на цикъла. В такъв случай се ползва конструкцията do-while. Тя има следния синтаксис:

do {прави това} while (израз за проверка на истинност)

Т.е. дори още първата проверка на условието да върне резултат "неистина", все пак ще получим еднократно изпълнение на цикъла, тъй като израза за проверка истинността на условието е след нареждането за първоначално изпълнение.

Пример:

```
<?
$prom1 = 100;
$prom2 = 50;
do
{
echo "Това съобщение ще се покаже поне веднъж.<br />";
}
while(++$prom1 < $prom2)
?>
```

В горния код стойността на $prom1 (100) е по-голяма от стойността на $prom2 (50), а зададеното условие е да се показва изречението "Това съобщение ще се покаже поне веднъж" докато $prom1 е по-малко от $prom2, т.е. докато 100 е по-малко от 50. Този цикъл ще върне "невярно" още при първото си изпълнение, понеже 100 не е по-малко от 50, но все пак ще имаме едно изпълнение на цикъла, т.е. едно показване на екран на изречението "Това съобщение ще се покаже поне веднъж". Ако разменим стойностите на двете променливи в същия код, тогава изречението ще бъде изведено на екрана 50 пъти.

3. Конструкция for

Често използвана е също конструкцията for. С нейна помощ се постига повече контрол над изпълнявания цикъл. Синтаксиса на for е следния:

for (променлива-брояч с начална стойност;условие;нова стойност на променливата-брояч) {направи това}

В кръглите скоби след ключовата дума for има 3 елемента:

- Задава се някаква начална стойност на променлива, от която начална стойност да започва изпълнението на цикъла. Тази променлива се явява брояч на това колко пъти ще се изпълни цикъла. Често в скриптовете тази променлива носи названието $i (от "инициализираща". т.е. начална стойност).
- Поставя се условието, като цикъла се изпълнява докато поставеното условие връща резултат "истина".
- Задава се увеличение на началната стойност на променливата.

Трите елемента се отделят един от друг чрез точка и запетая.

Преди всяко изпълнение на цикъла условието се проверява за истинност и ако резултата е true цикъла се изпълнява отново, като при това се увеличава стойността на променливата-брояч. Цялата конструкция дава възможност да се настрои точно определен брой пъти на изпълнението на цикъла.

Един най-прост пример с цикъла for е следния код:

```
<?php
for ($i = 1; $i <= 5; $i++)
{
echo $i;
}
?>
```

Горния код ще ни даде като резултат поредицата цифри от 1 до 5, тъй като е зададено началната стойност при изпълнение на цикъла да бъде 1 ($i = 1), при всяко следващо повторение на цикъла стойността на $i да се увеличава с единица ($i++), а в условието е зададено цикъла да продължи да се изпълнява докато стойността на променливата $i е по-малка или равна на 5 ($i <= 5). Следователно цикъла ще се изпълнява докато началната стойност 1 не нарастне до 5, т.е. ще имаме 5 повторения на цикъла.

Можем да усложним малко този пример:

```
<?php
for ($broi = 5; $broi < 20; $broi++)
{
$rezultat = $broi + 100;
echo "$broi + 100 = $rezultat<br />";
}
?>
```

Горния код извършва следното:

- на променливата-брояч $broi е зададена начална стойност 5 ($broi = 5;)
- поставеното условие е: скрипта се изпълнява докато стойността на брояча е по-малка от 20 ($broi < 20;)
- зададено е началната стойност на променливата-брояч да се инкрементира при всяко изпълнение на цикъла ($broi++), т.е. да се увеличава с единица при всяко следващо изпълнение на цикъла.

След това започва изпълнението на зададеното в конструкцията {$rezultat = $broi + 100;}. При първото изпълнение началната стойност на променливата-брояч е събрана със сто. Понеже началната стойност е 5, операцията е сбор на 5 със 100, което е 105. Резултата се присвоява на променливата $rezultat, показва се на екрана чрез echo и цикъла започва следващото си изпълнение. Тъй като е зададено променливата-брояч да се инкрементира, т.е. да увеличава стойността си с единица при всяко следващо изпълнение на цикъла, то при следващото изпълнение на цикъла нейната стойност ще бъде 6, съответно сбора вече ще бъде 6 + 100 и резултата ще е 106. При третото изпълнение на цикъла резултата ще е 107 и т.н.

По този начин цикъла ще се изпълнява докато израза за проверка не върне резултат "неистина". Според зададеното условие цикъла ще се изпълнява докато началната стойност 5 е по-малка от 20. Тъй като на всяко изпълнение началната стойност се увеличава с единици, то цикъла ще се изпълни 15 пъти (докато стойността на променливата не нарастне от 5 до 19).

Урок 12

Файлове и файлови функции.
Отваряне на файл. Запис във файл. Затваряне на файл

1. Файлове и файлови функции

Ако преди това сте работили с други езици за програмиране, като C или C++, може би си спомняте, че работата с файлове може да се раздели на следните етапи:

- **Отваряне на файл.** Зададения файл се отваря. Ако файла не съществува той се създава.
- **Обработка на файла.** След като файла е бил отворен, той е готов за обработка. Може да четете данните от файла, да записвате в него, като започнете отначало, или да добавите новите данни към вече съществуващите.
- **Затваряне на файл.** След като сте свършили работата си с файла, трябва да го затворите

PHP предлага множество функции за работа с файлове, които можете да използвате, за да запишете данни от Web страница във файл, да извлечете данните от файл в Web страница или просто да промените или да добавите данни от вашата Web страница във файл. Някой от най често използваните файлови операции са:

- Проверка за съществуването на файл
- Отваряне на файл
- Четене от файл
- Запис във файл

2. Проверка за съществуването на файл

Преди да извършите операция с даден файл, често ще ви се налага да проверите дали файлът съществува или не. За целта можете да използвате функцията file_exists(). Функцията file_exists() връща стойност True ако зададения файл съществува, или стойност False ако не съществува. Функцията file_exists() е със следния синтаксис:

bool file_exists(string име_на_файл);

Както се вижда от горния синтаксис, функцията file_exists() приема само един аргумент - име_на_файл, който задава името на файла, за чието съществуване се проверява.

Разгледайте следния код за да разберете как се използва функцията file_exists().

```
<?php
if (!(file_exists("data.dat"))) {
echo "Файлът съществува"
}
else {
echo "Файлът не съществува"
}
?>
```

В горния код наличието на файла data.dat се проверява с реда if (!(file_exists("data.dat"))). Както си спомняте от предишните уроци, операторът ! обръща резултатът върнат от извикването file_exists("data.dat").

3. Отваряне на файл

За да отворите файл в PHP трябва да използвате функцията fopen(). Тази функция може да отвори или файл или URL адрес. Оттук следва, че файлът може да бъде както локален, така и отдалечен. Функцията fopen() извиква поне два аргумента. Синтаксисът на функцията fopen() е следният:

int fopen(string име_на_файл, string режим, int път);

В горния синтаксис аргумента име_на_файл задава името на файла, който трябва да се отвори. Ако стойността на аргумента име_на_файл започва с http://, тогава означава, че файлът се намира на отдалечен Web сървар и за да се отвори файлът трябва да се установи HTTP 1.0 сесия със зададения сървър. Аналогично, ако аргументът име_на_файл започва с FTP:// , зададения файл трябва да се свали от FTP сървър след постановяването на FTP сесия със зададения сървър. Ако стойността на аргумента име_на_файл започва с php://stdin, php://stdout или с php://stderr, се отваря съответния стандартен входно изходен поток. Обаче, ако стойността на аргумента не започва с никой от тези префикси, се приема, че зададения файл е локален, и PHP го търси във файловата система на текущия компютър.

Вторият аргумент mode(режим) задава, дали файлът се отваря за четене , за запис или за добавяне.

Аргументът режим, може да приема една от следните стойности:

r Файлът се отваря само за четене

r+ Файлът се отваря за четене и запис, текущата позиция е началото на файла

w Файлът се отваря само за запис. Ако в него е имало някакви данни, те ще бъдат загубени. Ако файлът не съществува, се създава нов файл.

w+ Файлът се птваря за четене и запис. Ако в него е имало някакви данни, те ще бъдат загубени. Ако файлът не съществува, се създава нов файл.

a Файлът се отваря за добавяне на данни към към съществуващите в него. Ако файлът не съществува се създава нов файл.

a+ Файлът се отваря за четене и за добавяне на данни. Ако файлът съдържа данни, новите данни ще се запишат в края му. Ако файлът не съществува, се създава нов файл.

Третият аргумент include_pat (път) не е задължителен и се използва ако искате да търсите файла в пътя, зададен от този аргумент.

Следващият код демонстрира използването на функцията fopen().

```
<?php

if (!(file_exists("data.dat"))) {
$fp = fopen("data.dat","w+");
}
else {
// Ако файлът съществува, данните ще се добавят
$fp = fopen("data.dat","a");
}

?>
```

В горния код първо се прави проверка за съществуването на файла data.dat Ако файлът не съществува, се създава нов файл с име data.dat чрез режима w+. Това означава, че файлът е готов, както за четене, така и за запис. От друга страна, ако изразът if (file_exist("data.dat")) върне стойност True (тоест ако файлът data.dat съществува), файлът се отваря за добавяне на

данни към него. Можете да използвате знака @ пред функцията fopen(), както и пред всяка друга системна функция в PHP - например, @fopen("data.dat"),"w+");. Използването на символа @ в началото на функцията ще ви помогне да избегнете стандартните предупреждения на компилатора. Обаче трябва да се погрижите за предоставянето на собствени (при изпълнението на програмиста) съобщения за грешки, в случай, че функцията не успее да свърши задачата си.

4. Запис във файл

PHP предоставя функцията fwrite() за запис на данни във външен файл. Тази функция приема три аргумента. Синтаксисът на функцията ж е следния:

int fwrite(int файлов_манипулатор string низ, int дължина);

Първият аргумент е файлов манипулатор, в който искате да запишете данните. Вторият аргумент string (низ), е информацията, която искате да запишете. Последният аргумент, дължина, не е задължителен. Той задава броят байтове, които да се запишат във файла.

За да разберете по-добре действието на функцията fwrite(), разгледайте следния програмен фрагмент:

```
dataformat = "A20A50A200A4A2A10A6";
$line =
pack ($dataformat, $name, $addres, $email, $gander, $birth_mounth,
$comedy, $hobies[0]);
echo $line,"<br>",strlen($line);
$fp = @fopen("data.dat","w+")
or die ("Не може да се отвори файла");

// Запис във файла
$result = @fwrite($fp, $line)
or die ("Не може да се запишат данните във файла");

fclose($fp);
```

В горния код файлът data.dat се отваря в режим w+, което позволява операции за четене и запис във файла. След като файлът е отворен успешно, низът, съхранен в променливата $line, се записва във файла. Накрая, отворения файл се се затваря със помощта на функцията fclose().

5. Затваряне на файл

За да затворите файл, трябва да използвате функцията fclose(). Ако функцията fclose() затвори файла успешно, тя връща стойност True. Ако се провали затварянето на файла, функцията връща стойност False. Синтаксисът на функцията fclose() е:

bool fclose(int файлов_манипулатор);

Функцията fclose() изисква като аргумент файловият манипулатор на файла, който трябва да се затвори. За да се изпълни функцията успешно, стойността на файловият манипулатор трябва да е валиден и да съответства на файла, който преди това е отворен с функцията fopen().

Урок 13

Четене от файл

1. Четене от файл

За да прочетете данни от външен файл трябва да използвате функцията **fread()**. Синтаксисът на функцията fread() е:

string fread(int файлов_манипулатор, int дължина);

Както можете да видите, функцията fread() приема два аргумента: файлов_манипулатор и дължина. Аргументът файлов_манипулатор задава мястото във файла, от където да се чете. Аргументът дължина задава броя на символите, които да се прочетат от това място. Операцията по четене продължава докато се прочетат толкова символи, колкото са зададени от аргумента дължина. Ако се достигне до края на файла преди да се прочетат необходимия брой символи, ще се върне низ, съдържаш прочетените до момента символи.

Разгледайте следния файл:

```
$fo = @fopen("C:PHPMyfilesdata1.dat", r)
or die ("Липсва файла!");
//-- Прочитат се първите 124 символа от файла data1.dat
$fr = fread($fo, 124);
```

В горния код файла data1.dat се отваря за четене с помощта на функцията fread(). Изразът or die ("Липсва файла!"); се използва за отпечатване на съобщение, ако функцията fopen() не успее да отвори файла. Резултатът от функцията (1 или 0) се съхранява в променливата $fo. След това тази променлива се предава на функцията fread() като файлов манипулатор. Стойността 124, предадена на fread(), задава, че трябва да се прочетат първите 124 символа от файла data1.dat

Четенето на файл с fread е побитово и можем да контролираме, колко бита ще прочетем, използвайки вторият параметър $length, оказващ броя на битове, които искаме да прочетем. Ако искаме да прочетем целия файл, което е в повечето случай, ще изпозваме функцията int filesize (string $filename), която връща броя на битовете в един файл:

```
<?php
// get contents of a file into a string
$filename = "something.txt";
$handle = fopen($filename, "r");
$contents = fread($handle, filesize($filename));
fclose($handle);
?>
```

След изпълнението на файла ще получим съдържанието на файла в променливата $contents.**file_get_contents и file_put_contents.**

Както може би сте забелязали вече, операциите по писане и четене от файл са еднотипни и доста сложни за запомняне. Затова php съдържа и други функции за четене и писане на файлове. Ще разгледаме две специфични:

file_get_contents прочита цялото съдържание на файла по дадено име, което прави операцията по четене доста по-лесна и поради тази причина file_get_contents е доста популярна и често използвана функция:

```
<?php
// get contents of a file into a string
$filename = "something.txt";
$contents = file_get_contents($filename);
?>
```

Това е същият скрипт за четене на файл, но използвайки file_get_contents. Сами виждате колко по-лесно става четенето от файл.

file_put_contents – записва съдържание в текстов файл. Ето и пример:

```
<?php
$file = 'people.txt';
$person = "John Smith\n";
file_put_contents($file, $person);
?>
```

Указаният скрипт записва във файла people.txt променливата $person. Извикана по този начин, функцията ще замести старото съдържание на файла с новото. Ако се налага да добавяме съдържание, ще трябва да използваме и трети параметър. Ето пример:

```
<?php
$file = 'people.txt';
$person = "John Smith\n";
file_put_contents($file, $person, FILE_APPEND);
?>
```

Това добавява в файла съдържанието на променливата $person.

Урок 14

Прочитане на символ. Четене на произволна дължина. Проверка на файл. Определяне на размер

1. Прочитане на символ. Четене на произволна дължина.

Функциите fgetc() и fgets()

Функцията fread() връща низ с определена дължина от текущата позиция във файла. Но как можете да прочетете един символ? Можете лесно да накарате функцията fread() да прочете един символ, като зададете желаната позиция и дължина 1. Но PHP ви предоставя по-лесен начин да направите това! За целта използвайте функцията fgetc(). Тази функция взема един символ от текущата позиция във файла. Синтаксисът на функцията fgetc() е:

string fgetc(int файлов_манипулатор);

Функцията fgets() връща ред (низ) от текущата позиция във файла. За разлика от функцията fread(), дължината на низа върнат от функцията fgets(), е с единица по-малко от зададената като аргумент. Синтаксисът на функцията fgets() е:

string fgets(int файлов_манипулатор int дължина)

Операцията по четене се прекратява при едно от следващите условия:

- Прочетени са дължината -1 байт.
- Достигнат е символа за нов ред – n

- Достигнат е края на файла

Разгледайте следния код, който демонстрира използването на функциите:

```
<?php

$file = "data1.dat";
// Отваряне на файла data1.dat

$fo = @fopen($file, "r")
or die ("Не може да се намери файла!");

// Намиране размера на файла - data1.dat
$file_length = filesize($file);
echo "Дължината на файла е : $file_length", "n";

// Прочитане на файла ред по ред. При всяко срещане на края на ред,
// променливата $total_rows се увеличава с 1
while (!(feof($fo)))
{
$tr = fgets($fo, $file_length);
$total_rows = $total_rows + 1;
}
echo "Броят на редовете във файла е: $total_rows ", "n";

// Затваряне на файла data1.dat за да може да се извърши следващия цикъл
fcolse($fo);

// Повторно отваряне на файла data1.dat и текущата позиция е в началото на файла.
$fo1 = @fopen($file, "r");

// Прочитане на файла символ по символ. При всяко срещане на символ,
// променливата $total_rows се увеличава с 1
while (!(feof($fo)))
{
$tr = fgetc($fo1);
$total_rows = $total_rows + 1;
}
echo "Броят на символите във файла е: $total_rows ", "n";

?>
```

2. Проверка на файл - функцията feof()

Можете да използвате функцията feof(), за да определите дали цялото съдържание на зададения файл е било обработено и дали текущата позиция е в края на файла. Функцията връща стойност True, ако файловият манипулатор в момента сочи в края на файла, и стойност False, ако не е достигнат края на файла.

Функцията feof() приема като аргумент манипулатор на файл, както е показано в следващия синтаксис

```
int feof(int файлов_манипулатор);
```

3. Определяне на размер - функцията filesie()

Функцията filesie() връща размера на зададения файл. Ако зададения файл не съществува, се връща стойност False (или 0). Функцията има следния синтаксис:

int filesie(string име_на_файл);

Както се вижда от горния синтаксис, функцията filesie() приема един аргумент - името на файла, чиято дължина трябва да се определи.

4. Текущата позиция във файл - функцията fseek()

Можете да използвате функцията fseek(), за да зададете текущата позиция във файл. Синтаксисът на функцията е следният:

int fseek(int файлов_манипулатор, int отместване, int база);

В горният синтаксис първият аргумент е файлов манипулатор, който задава файла върху който ще се изпълни функцията. Следващият аргумент отместване, е цяло число, което задава новата текуща позиция във файла. Третия аргумент база, задава как да се извърши изчислението на новата позиция. Стойностите на двата аргумента отместване и база представляват брой байтове.

Аргументът база има следните три стандартни стойности:

- **SEEK_SET** Текущата позиция се премества от началото на файла на зададения брой байтове от параметъра отместване.
- **SEEK_CUR** Текущата позиция се изчислява, като към текущата позиция във файла се добавят броят байтове, зададени в параметъра отместване.
- **SEEK_END** Текущата позиция се изчислява като сума от дължината на файла и броя байтове, зададени в параметъра отместване.

За да разберете по-добре как действа функцията fseek(), разгледайте следния пример:

```
<?php

$file = "data1.dat";

// Отваряне на файла data1.dat
$fo = $fopen($file, "r")
or die ("Не може да се намери файла");

// Определяне на дължината на data1.dat
$file_length = filesize($file);

echo "n";
echo "Размерът на файла: $file_length.", "n";

// Позициониране на 22 байт във файла
$fs = fseek($fo, 22, SEEK_CUR);

// Прочитане на 6 символа от текущата позиция във файла
$current_char = fread($fo, 6);

echo "В момента текущата позиция е върху символа $current_char";

?>
```

Урок 15

Изтриване на файл. Навигация във файл. Заключване на файлове

1. Изтриване на файл – дункцията unlink()

- описание **bool unlink (string $filename [, resource $context])**

  Изтрива файл указан с *filename* .
- параметри – *filename* (път до файла), *context(...)*
- връщани стойности - връща TRUE при успех или FALSE при неуспех.

Примери: Използване на unlink()

```
<?php
$fh = fopen('test.html', 'a');
fwrite($fh, '<h1>Hello world!</h1>');
fclose($fh);

mkdir('testdir', 0777);

unlink('test.html');
unlink('testdir');
?>
```

2. Навигация във файл

3. Заключване на файл

Урок 16

Какво е масив? Деклариране на масиви

Създаване, запълване, добавяне и изтриване на елементи от масив

В материала за типовете на променливите вече беше споменато за един особен тип променливи - array (масив). Масивите са променливи, които, за разлика от другите променливи, могат да съдържат повече от една стойност. Декларирането им става с функцията array(), като в кръглите скоби се записват стойностите на масива разделени със запетаи. Всяка стойност на масива автоматично получава идентификационен номер, като началния номер по подразбиране е 0. При извикване на някоя стойност на масива тя трябва да се укаже чрез идентификационния си номер записан в квадратни скоби, например:

```
<?php
$promenliva = array("стойност1", "стойност2", "стойност3");
echo "$promenliva[0]";
?>
```

Резултата от горния код ще бъде извеждане на екран на стойност1, тъй като това е стойността с идентификационен номер 0.

Стойностите на масива могат да се зададат и поотделно на променливата, като в този случай пред името на променливата трябва да се постави квадратна скоба, която може да съдържа идентификационен номер. Ако не се укаже изрично идентификационен номер, тогава такъв се присвоява автоматично, като отново началния номер по подразбиране е 0, например:

```
<?php
$promenliva[] = "стойност1";
$promenliva[] = "стойност2";
$promenliva[] = "стойност3";
echo "$promenliva[0]";
?>
```

Резултата от този код отново ще е стойност1.

Идентификационния номер може да бъде и изрично указан. В такъв случай не е задължително началния номер да бъде 0. Можете да използвате всякакви номера каквито сметнете за необходими, например:

```
<?php
$hrana[5] = "ябълки";
$hrana[8] = "домати";
$hrana[25] = "хляб";
$hrana[17] = "пилешко";
echo "$hrana[25]";
?>
```

Резултата от горния код ще бъде извеждане на екрана на думата "хляб", тъй като 25 е номера на стойността "хляб" от масива $hrana.

Вместо номера за идентифициране на съответната стойност на масива може да се ползват и думи, например:

```
<?php
$hrana[plodove] = "ябълки";
$hrana[zelenchuci] = "домати";
$hrana[testeni] = "хляб";
$hrana[meso] = "пилешко";

echo "$hrana[meso]";
?>
```

Горния код ще изведе на екран думата "пилешко", защото тази стойност на масива е идентифицирана чрез думата "meso".

Думите чрез които се идентифицират стойностите на масива се наричат ключове. В случая думата "meso" е ключ за стойността "пилешко", думата "plodove" е ключ за стойността

"ябълки" и т.н. Даден масив може да се дефинира и като всяка стойност бъде указана с нейния ключ в кръглите скоби на функцията array. В този случай синтаксиса на масива е

$promenliva = array(ключ1 => стойност1, ключ2 =>стойност2, ключ3 => стойност3);

т.е. всеки ключ сочи чрез символите равенство и стрелка (=>) към своята стойност, като двойките ключ-стойност са отделени със запетаи. Пример:

```
<?php
$hrana = array(plodove => "ябълки", zelenchuci => "домати", testeni => "хляб", meso =>
"пилешко");
echo "$hrana[plodove]";
?>
```

В този случай резултата ще бъде извеждане на екран на думата "ябълки", тъй като ключа на тази стойност е "plodove".

Същия принцип - ключове, сочещи към стойности - можете да видите например при вградените в езика масиви $_POST, $_GET и $_REQUEST, с които се обработват данни от формуляри. Имената на полетата на формулярите, зададени чрез атрибута name, например name="komentar", name="email" и т.н., се явяват ключове, сочещи към стойността (съдържанието) на съответното поле от формуляра и затова се задават в квадратните скоби след името на масива, например $_POST[komentar], $_POST[email] и т.н. Повече информация за работа с тези масиви може да видите на страница Предварително дефинирани масиви. Обхват на променливите. Обработка на данни от формуляр.

Ключовете на стойностите зададени чрез функцията array() могат да бъдат и цифрови, например:

```
<?php
$hrana = array(5 => "ябълки", 8 => "домати", 25 => "хляб", 17 => "пилешко");
echo "$hrana[5]";
```

Горния код отново ще изведе на екран думата "ябълки", тъй като е зададено ключа на тази стойност да бъде 5.

Ако желаете стойностите на даден масив да бъдат номерирани от точно определена цифра нататък трябва да зададете тази цифра като ключ на първата стойност на масива. Следващите стойности на масива ще получат автоматично пореден идентификационен номер според местоположението си, например:

```
<?php
$hrana = array(10 => "ябълки", "домати", "хляб", "пилешко");
echo "$hrana[11]";
?>
```

Горния код ще изведе на екран думата "домати", тъй като на първата стойност на масива, която е "ябълки", сме задали номер 10, т.е. броенето започва от цифрата 10. Стойността "домати" е втора в изброяването и затова автоматично получава идентификационен номер 11. Стойностите "хляб" и "пилешко" получават съответно номера 12 и 13.

Същото правило важи и когато стойностите се задават поотделно на масива, например:

```
<?php
$hrana[10] = "ябълки";
$hrana[] = "домати";
$hrana[] = "хляб";
$hrana[] = "пилешко";
echo "$hrana[11]";
?>
```

В този случай отново ще получим като резултат извеждане на екран на стойността "домати", тъй като тя автоматично получава пореден идентификационен номер 11.

Масивите могат да съдържат смесено стойности от различни типове и във всеки момент на стойностите им може да бъде променено съдържанието и типа данни, например:

```
<?php
$hrana[plodove] = "ябълки";
$hrana[zelenchuci] = 1.25;
$hrana[zarneni] = "хляб";
$hrana[meso] = "пилешко";
echo "$hrana[meso]<br />";
$hrana[meso] = 5.75;
echo "$hrana[meso]";
?>
```

В горния пример стойността на $hrana[meso] е променена от "пилешко" (низ) на 5.75 (число с плаваща запетая).

Урок 17

Вградени функции за работа с масиви

PHP поддържа множество вградени функции чрез които масивите могат да бъдат обработвани. Например функцията **foreach()** обхожда целия масив и присвоява на променлива списъка с елементите на масива, а функцията **count()** преброява елементите на масива.

Синтаксиса на функцията foreach() е

foreach(име на масив as име на променлива) {изпълним код с променливата}

Т.е. в кръглите скоби се записва името на масива, ключовата дума as и след нея - името на променлива, на която се присвоява като стойност целия масив. След това в големите скоби се задава действие с променливата, която е получила като стойност съдържанието на масива.

Синтаксиса на функцията count() е

count(име на масив)

Т.е. в кръглите скоби просто се записва името на масива. Като резултат функцията връща броя на стойностите в този масив.

Пример:

```
<?php
$hrana = array("ябълки", "домати", "хляб", "пилешко");
```

```
foreach($hrana as $spisak_na_hrani)
{echo "$spisak_na_hrani<br />";}
$broi = count($hrana);
echo "Обща бройка: $broi";
?>
```

В горния код най-напред създаваме масива $hrana с четири стойности, които са "ябълки", "домати", "хляб" и "пилешко".

След това задаваме обхождане на масива чрез функцията foreach() като целия масив $hrana се присвоява във вид на стойност на променливата $spisak_na_hrani. Чрез echo в големите скоби извеждаме този списък на екрана.

Накрая чрез функцията count() задаваме преброяване на елементите от списъка и извеждаме бройката на екран.

Резултата от този код в браузъра ще бъде:

ябълки
домати
хляб
пилешко
Обща бройка: 4

Може да сортирате по азбучен ред елементите на даден масив чрез функцията **sort(име на масив)**, например:

```
<?php
$hrana[plodove] = "ябълки";
$hrana[zelenchuci] = "домати";
$hrana[testeni] = "хляб";
$hrana[meso] = "пилешко";
sort($hrana);
foreach($hrana as $spisak_na_hrani)
{echo "$spisak_na_hrani<br />";}
?>
```

Горния код ще изведе списъка на названията на храните подредени според мястото на началните им букви в азбуката, а именно:

домати
пилешко
хляб
ябълки

Ако вместо названия като стойности на масива са зададени цифри, функцията sort() ще сортира списъка по големина на цифрите.

Когато се ползва функцията sort() тя автоматично заменя ключовете за съответните стойности с идентификационните номера по подразбиране. Т.е. ако използваме функцията sort() както в горния код и след това се опитаме да извикаме например стойността "домати" чрез $hrana[zelenchuci] няма да получим резултат. За да може да извикаме тази стойност ще трябва да напишем $hrana[0], тъй като функцията sort() освен че е сортирала елементите по азбучен ред, поставяйки стойността "домати" на първо място в масива, но също така и автоматично е заменила ключа "zelenchici" с идентификационния номер по подразбиране, който за първия елемент от масива винаги е 0. Т.е. за да получим резултата

домати
пилешко

хляб
ябълки
домати
трябва да напишем

```
<?php
$hrana[plodove] = "ябълки";
$hrana[zelenchuci] = "домати";
$hrana[testeni] = "хляб";
$hrana[meso] = "пилешко";
sort($hrana);
foreach($hrana as $spisak_na_hrani)
{echo "$spisak_na_hrani<br />";}
echo "<b>$hrana[0]</b>";
?>
```

За да се запазят зададените ключове на стойностите трябва вместо sort() да се използва функцията **asort()**. Тя има същото предназначение, т.е. сортира елементите на масива по големина на стойност, ако стойността е цифрова или по азбучен ред ако стойностите са буквени, но запазва зададените от програмиста идентификационни номера или ключове на стойностите на масива.

Ако желаете да сортирате елементите на даден масив в обратен азбучен или цифров ред трябва да използвате функцията **rsort(име на масив)**, например:

```
<?php
$hrana[plodove] = "ябълки";
$hrana[zelenchuci] = "домати";
$hrana[testeni] = "хляб";
$hrana[meso] = "пилешко";
rsort($hrana);
foreach($hrana as $spisak_na_hrani)
{echo "$spisak_na_hrani<br />";}
?>
```

Горния код ще ни даде като резултат списъка:

ябълки
хляб
пилешко
домати

Т.е. елементите се подреждат в обратен азбучен ред, като стойността "домати" става последна, а стойността "ябълки" - първа.

Също като функцията sort() и функцията rsort() автоматично заменя ключовете или зададените номера на стойностите с идентификационните им номера по подразбиране. За да се избегне това е необходимо да ползвате функцията **arsort(име на масив)**, която подобно на rsort() сортира елементите на масива в обратен ред, но за разлика от rsort() запазва зададените идентификационни ключове или номера на стойностите.

Функцията **shuffle(име на масив)** подрежда елементите на даден масив в произволен ред. Ако използвате тази функция, например в горния код на мястото на rsort(), при всяко ново зареждане на страницата ще получавате като резултат нова произволна подредба на наименованията на хранителните продукти в масива $hrana.

Функцията **array_pop(име на масив)** "отрязва" последния елемент на даден масив и го премахва от списъка със стойности на масива, т.е. ако имаме кода

```
<?php
$hrana = array("ябълки", "домати", "хляб", "пилешко");
array_pop($hrana);
foreach($hrana as $spisak_na_hrani)
{echo "$spisak_na_hrani<br />";}
?>
```

резултата от него ще бъде:

ябълки
домати
хляб

тъй като последния елемент на масива - пилешко - е премахнат с помощта на array_pop().

Функцията **array_shift(име на масив)** извършва същото действие като array_pop() с тази разлика, че премахва първия елемент от списъка със стойности на даден масив.

С функцията **array_push()** могат да се добавят нови елементи към списъка с елементи в даден масив. Новите елементи се добавят в края на списъка. Синтаксисът на функцията е:

array_push(име на масив, 1-ви нов елемент, 2-ри нов елемент,...)

Т.е. в кръглите скоби се записва името на масива и след това отделени със запетаи се записват новите елементи на масива, например:

```
<?php
$hrana = array("ябълки", "домати", "хляб", "пилешко");
array_push($hrana, "мляко", "яйца");
foreach($hrana as $spisak_na_hrani)
{echo "$spisak_na_hrani<br />";}
?>
```

Горния код ще изведе на екран списъка:

ябълки
домати
хляб
пилешко
мляко
яйца

тъй като чрез array_push() сме добавили в края на списъка стойностите "мляко" и "яйца".

Функцията **array_unshift()** е със същия синтаксис като array_push() и също добавя нови елементи към даден масив, но ги добавя в началото, а не в края на списъка със стойности на масива.

Чрез функцията **array_merge()** може да обедините два или повече масива в един. Синтаксисът на функцията е

array_merge(име на 1-ви масив, име на 2-ри масив,...)

Т.е. в кръглите скоби се записват имената на масивите които ще обединявате, разделени със запетая, например:

```
<?php
$hrana1 = array("ябълки", "домати", "хляб", "пилешко");
$hrana2 = array("мляко", "яйца");
$hrana3 = array("шоколад", "сладолед");
$hrana = array_merge($hrana1, $hrana2, $hrana3);
foreach($hrana as $spisak_na_hrani)
{echo "$spisak_na_hrani<br />";}
?>
```

В горния код трите масива $hrana1, $hrana2 и $hrana3 са обединени в един и резултата показан на екран ще бъде:

ябълки
домати
хляб
пилешко
мляко
яйца
шоколад
сладолед

Функцията **array_sum(име на масив)** извършва сбор на всички стойности в масива. Например резултата от кода

```
<?php
$chisla = array(1, 2, 3, 4, 5);
echo array_sum($chisla);
?>
```

ще бъде цифрата 15, защото това е сумата на числата от 1 до 5, които са стойности на масива $chisla.

Функциите **max(име на масив)** и **min(име на масив)** откриват съответно най-високата и най-ниската стойност в даден масив. Например кода:

```
<?php
$chisla = array(1, 2, 3, 4, 5);
$naj_goliamo = max($chisla);
$naj_malko = min($chisla);
echo "Най-високата стойност в масива е $naj_goliamo, а най-ниската е $naj_malko";
?>
```

ще върне като резултат съобщението "Най-високата стойност в масива е 5, а най-ниската е 1".

Функциите **reset(име на масив)** и **end(име на масив)** връщат като резултат съответно първата и последната стойност на даден масив. Например кода

```
<?php
$hrana[plodove] = "ябълки";
$hrana[zelenchuci] = "домати";
$hrana[testeni] = "хляб";
$hrana[meso] = "пилешко";
$parvi = reset($hrana);
$posleden =  end($hrana);
echo "Първия елемент на масива е $parvi, а последния е $posleden";
?>
```

ще върне като резултат съобщението "Първия елемент на масива е ябълки, а последния е пилешко".

PHP поддържа десетки функции за обработка на масиви. В настоящите материали са представени само някои от тях. Пълен списък с тези функции и описание на действието им може да видите в официалния сайт www.php.net

Урок 18
Многомерни масиви

Досега разглеждахме масиви, които представляваха прости списъци с по няколко елемента. Например масива $hrana фактически е списък на който зададохме да съдържа няколко храни - домати, ябълки, хляб и пилешко. Такива масиви се наричат едномерни. По-често обаче се налага да се работи със списъци, чиито елементи съдържат подсписъци с елементи. Например в един списък на храни може да имаме елемента "месо", който да съдържа подсписък с няколко вида месо, примерно "пилешко", "телешко" и "свинско". Съответно елемента "пилешко" може да съдържа на свой ред подсписък с елементите "пилешки бутчета", "пилешки крилца" и "пилешки гърди" и т.н.

Такива вложени един в друг списъци е лесно да се представят чрез т.нар. многомерни масиви. Многомерните масиви, както подсказва названието им, съдържат в себе си други масиви. Многомерните масиви могат да бъдат двумерни, тримерни и т.н.

Да направим двумерен масив, който да съдържа елемента "месо" с поделементите "пилешко", "телешко" и "свинско". Кода може да изглежда по следния начин:

```
<?php
$hrana[meso][1] = "пилешко";
$hrana[meso][2] = "телешко";
$hrana[meso][3] = "свинско";
echo $hrana[meso][2];
?>
```

Този код ще изведе на екран елемента "телешко". От примера се вижда, че след като масива е създаден обръщането към някакъв негов елемент става като след името на масива се изпишат в квадратни скоби идентификационните ключовете или номера на съответния елемент. Ако се извиква стойност от двумерен масив, както е в случая, след името на масива ще трябва да се изпишат две квадратни скоби с ключове или номера; ако се извиква елемент от тримерен масив се изписват три квадратни скоби с ключове или номера и т.н.

В един многомерен масив може да разположите колкото желаете списъци и подсписъци, например:

```
<?php
$hrana[meso][pileshko][butcheta] = "пилешки бутчета";
$hrana[meso][pileshko][krilca] = "пилешки крилца";
$hrana[meso][pileshko][gyrdi] = "пилешки гърди";
$hrana[plodove][yabulki] = "ябълки";
$hrana[plodove][krushi] = "круши";
$hrana[plodove][banani] = "банани";
echo "Втория елемент на списъка е {$hrana[meso][pileshko][krilca]}, а последния е
{$hrana[plodove][banani]}";
?>
```

Горния код ще изведе на екран съобщението "Втория елемент на списъка е пилешки крилца, а последния е банани".

Извикването на елемент от многомерен масив в низ става като названието на масива и съответните ключове или номера се поставят в големи скоби. Забележете, че в този пример вътре в низа, т.е. в изречението след echo, името на масива и ключовете на стойностите са поставени именно в такива скоби - {}. Ако махнем скобите и напишем

```
echo "Втория елемент на списъка е $hrana[meso][pileshko][krilca],а последния е
$hrana[plodove][banani]";
```

ще получим като резултат съобщението "Втория елемент на списъка е Array[pileshko][krilca], а последния е Array[banani]".

Многомерен масив може да се създаде и като се ползва функцията array и символите => по следния начин:

```
<?php
$hrana = array("meso" => array("пилешко", "телешко",
"свинско"), "plodove" => array("ябълки", "круши",
"банани"));
echo $hrana[meso][0];
?>
```

Горния код ще ни покаже резултат "пилешко", тъй като в случая няма зададени ключове или номера на стойностите от списъка с видовете месо и затова на стойността "пилешко" автоматично е присвоен идентификационен номер 0.

Урок

Предварително дефинирани масиви. Обхват на променливите. Обработка на данни от формуляр

В материалите за функциите е подчертано, че съществуват предварително дефинирани в езика функции, както и възможност програмиста да създава собствени функции.

По същия начин в PHP съществуват и предарително дефинирани променливи, които могат да се използват в скриптовете без да е необходимо преди това да бъдат декларирани. Например такива предварително дефинирани променливи са променливите на средата за които стана дума в материалите за променливи.

Някои предварително дефинираните променливи са от типа масив. Две от най-ползваните такива променливи са $_POST и $_GET.

Тук е необходимо да кажем няколко думи за обхвата на действие на променливите. Променливите в PHP имат строго определен обсег на действие. Например когато една променлива бъде декларирана във функция, тя може да бъде използвана само от тази функция и се нарича локална променлива.

Друг вид променливи са т.нар. глобални променливи, които могат да се ползват от всички функции в скрипта. За да се възприеме една променлива като глобална е нужно вътре във функцията изрично да се декларира, че тя е такава. Това се прави като пред името на променливата, употребена в дадена функция, се пише ключовата дума global. Ако това не бъде направено PHP ще възприеме променливата като локална по подразбиране. В долния пример променливата $prom1 е декларирана като глобална във функцията funk1():

```
<?php
$prom1 = стойност;
function funk1()
{
global $prom1;
}
?>
```

Няколко променливи може да бъдат декларирани едновременно като глобални. За целта е необходимо те да бъдат записани след ключовата дума global и да са отделени една от друга със запетаи, например:

```
<?php
$prom1 = стойност1;
$prom2 = стойност2;
function funk1()
{
global $prom1, $prom2;
}
?>
```

Съществуват и още един вид променливи - така наречените суперглобални променливи, които са предварително дефинирани в езика и могат да бъдат ползвани в произволен брой скриптове. $_POST и $_GET са именно такива променливи. Те са от типа променливи array (масив), затова освен суперглобални променливи биват наричани още суперглобални масиви (superglobal arrays). PHP не разполага с механизъм чрез който да създавате собствени суперглобални променливи, т.е. могат да се ползват единствено тези, които са предварително вградени в езика.

Масивите $_POST и $_GET позволяват да бъде достъпвана информацията, въведена от юзери във формуляр. Когато даден юзер попълни съответния формуляр и натисне бутона за извършване на действието (Submit), въведените във формуляра данни биват изпратени и стават достъпни след обработката им от PHP скрипта.

Подобен на $_POST и $_GET е и масива $_REQUEST, който може да бъде използван вместо тях.

От HTML би трябвало да знаете, че чрез атрибута method се определя какво действие ще се извърши с формуляра, т.е. какъв ще е метода, по който ще се оперира с данните. Методите могат да бъдат GET и POST. Чрез тях съдържанието на формуляра се изпраща за обработка от скрипта, указан чрез атрибута action.

Да направим един най-прост формуляр с едно поле за въвеждане на име:

```
<html>
<head>
<title>Формуляр за въвеждане на име</title>
</head>
<body>
<form action="showname.php" method="POST">
Въведете името си:<br />
<input type="text" name="ime">
<input type="submit" value="Изпрати данните">
</form>
</body>
</html>
```

Формуляра ще изглежда така:

Въведете името си:

Може да съхраните страницата например като form.html или form.php
Сега трябва да направим скрипта showname.php, който ще обработва въведените данни:

```
<?php
echo $_POST["ime"];
?>
```

Чрез масива $_POST вземаме от формуляра стойността на полето с название ime. Това правим като записваме названието на полето в квадратни скоби след названието на масива. Названието на полето, в случая ime, представлява "ключ", чрез който може да се обърнем и да достъпваме стойността, към която "ключа" сочи. Тази стойност ще бъде името на потребителя, което той трябва да въведе във формуляра. Извеждаме стойността на екран чрез командата echo.

Повече информация за ключовете и стойностите на масивите може да видите в материала - Масиви: Деклариране на масиви.

Сега вече може да ползваме формуляра. При написване на нещо в полето и натискане на бутона "Изпрати данните" скрипта ще покаже въведената информация.

Може да усложним малко примера като направим формуляр с две полета, например за име и имейл:

```
<form action="nameandemail.php" method="post">
```

```
Въведете името си:<br />
<input type="text" name="ime"><br />

Въведете имейла си:<br />
<input type="text" name="email"><br />

<input type="submit" value="Изпрати данните">
</form>
```

Скрипта nameandemail.php може да изглежда така:

```
<?php

$ime = $_POST["ime"];
$email = $_POST["email"];

if ($ime == null || $email == null)
{echo "Не сте въвели име или имейл!";}

else
{echo "Здравейте, $ime! Вашия имейл е $email.";}

?>
```

В случая стойностите на $_POST за името и имейла са присвоени съответно на променливите $ime и $email. След това чрез конструкцията if-else и логическия оператор ИЛИ (||) сме дали на скрипта следната задача:АКО променливата $ime ИЛИ променливата $email съдържат празна стойност (null), тогава покажи съобщението "Не сте въвели име или имейл!". В ПРОТИВЕН СЛУЧАЙ, т.е. ако и двете променливи не са с нулева стойност (ако и в двете полета на формуляра има въведени данни), покажи съобщението "Здравейте (следва написаното в 1-то поле)! Вашия имейл е (следва написаното във 2-то поле)".

По този начин правим елементарна проверка за грешки от страна на юзера, в случая - дали не е оставил някое поле непопълнено.

Ако напишете в полетата от формуляра например името Иван Иванов и имейла ivanov@domain.com, след натискане на бутона "Изпрати данните" резултата ще бъде съобщението:**Здравейте, Иван Иванов! Вашия имейл е ivanov@domain.com**

Ако пропуснете да попълните едното или и двете полета на формуляра резултата ще е съобщението:**Не сте въвели име или имейл!**

Урок 12

Какво е функция? Видове функции. Дефиниране и извикване на функция

Фукциите са друг, също много важен вид конструкции на езика. Може да се каже, че променливите и функциите са най-важните елементи в PHP.

Функцията представлява самостоятелно, автономно "парче" код, което трябва да изпълни определена задача. Бихме могли да я наречем "скрипт в скрипта" или "подскрипт".

1. Видове функции

Функцията, подобно на променливата, трябва да има някакво название. В PHP има както готови (вградени в езика, стандартни) функции, чиито имена са фиксирани, така има и възможност вие да създадете каквито функции желаете, като им зададете всякакви произволни названия. Т.е. функциите в PHP биват:

- стандартни функции (вградени в езика), с фиксирани имена
- собствени функции (дефинирани от разработчика), с произволни имена

PHP поддържа стотици вградени в езика функции, с които могат да се извършват разнообразни операции. В примерите са разгледани някои от най-популярните функции като например функцията mail() с чиято помощ може да се изпращат съобщения на имейл, функцията include() чрез която един документ може да бъде включен в друг документ и функцията date() чрез която може да се позват текущите дата и час. В материалите за масиви са описани някои от функциите за работа с масиви. Списък на функциите в PHP може да видите в официалния сайт www.php.net

2. Названия на функции. Синтаксис на имената на функциите

Функциите се отличават от променливите и по това, че не съдържат знака на долара, нито какъвто и да е друг символ пред името. Всички функции задължително съдържат след названието си кръгли скоби, в които може да има поставени един или няколко параметъра, а може и да няма никакви параметри. Следователно обичайния вид на функциите е: **funkcia()**

След като веднъж е създадена, за разлика от променливата, функцията не може да бъде променяна.

Синтаксисът на имената при функциите се подчинява на същите правила като синтаксиса на имената при променливите, т.е. името на дадена функция може да съдържа латински букви, арабски цифри и долно тире, като може да започва с буква или долно тире, но не и с цифра.

За разлика от променливите обаче при имената на функциите няма значение дали те са зададени с големи или малки букви. Т.е. названията funkcia(), Funkcia() и FUNKCIA() ще бъдат възприети от PHP като една и съща функция. Понеже не е възможно да имате две или повече функции с еднакви названия, то при създаването на всяка функция трябва да задавате уникално име, което се отличава от другите имена на функции поне по един символ, а не по големите и малките букви. Например funk1() и FUNK1() са еднакви названия на функции и не могат да бъдат използвани и двете едновременно, докато funk1() и funk2() са две различни уникални имена на функции.

3. Създаване на собствени функции. Извикване на функции.

Създаването (декларирането) на ваша собствена функция има следния синтаксис:

```
function ime_na_funkcia ($parameter1, $parameter2, $parameter3...)
{код в "тялото" на функцията}
```

Т.е. най-напред се изписва ключовата дума function, след което трябва да запишете произволно избрано име на функцията. В кръглите скоби, които се поставят след названието на функцията, може да има произволен брой параметри, а може и да няма нито един. След това в големи скоби се затваря изпълнимия код от "тялото" на функцията.

Друга важна особеност на функциите е, че след като дадена функция е създадена (дефинирана) веднъж, тя може да бъде извикана колкото искате пъти и на което място в кода пожелаете. В кръглите скоби може да разположите колкото желаете параметри, а в големите скоби може да имате колкото и какъвто пожелаете код. Както разбирате по този начин може да си спестите доста труд, защото кода който ще се наложи да изпишете ще бъде много по-малко и по-прегледен, отколкото ако трябва да направите всичко чрез HTML.

Да си представим, че имате уеб страница в която трябва на няколко места да покажете даден списък с някакви елементи. В такъв случай кода ви може да изглежда по следния начин:

```
<?php
function mylist()
{
echo
"<ul>
<li> Елемент 1
<li> Елемент 2
<li> Елемент 3
<li> Елемент 4
<li> Елемент 5
</ul>";
}
// Тук ще покажем списъка за пръв път
mylist();
// Тук можем да го покажем за втори път
mylist();
?>
```

В горния код създадохме функцията mylist(), която е без параметри и след това я извикахме два пъти. Този код ще има като ефект двукратното показване на списъка с елементите от 1 до 5. Т.е. след като функцията е създадена и е зададен изпълнимия код в големите скоби, след това може да я извикате един или няколко пъти просто като напишете названието й в кода на документа на необходимото място.

Обърнете внимание, че кръглите скоби задължително се изписват след името на функцията, независимо, че в тях може да няма никакви параметри, както е в случая. Конструкцията на извикването на функция, също като другите конструкции, завършва с точка и запетая.

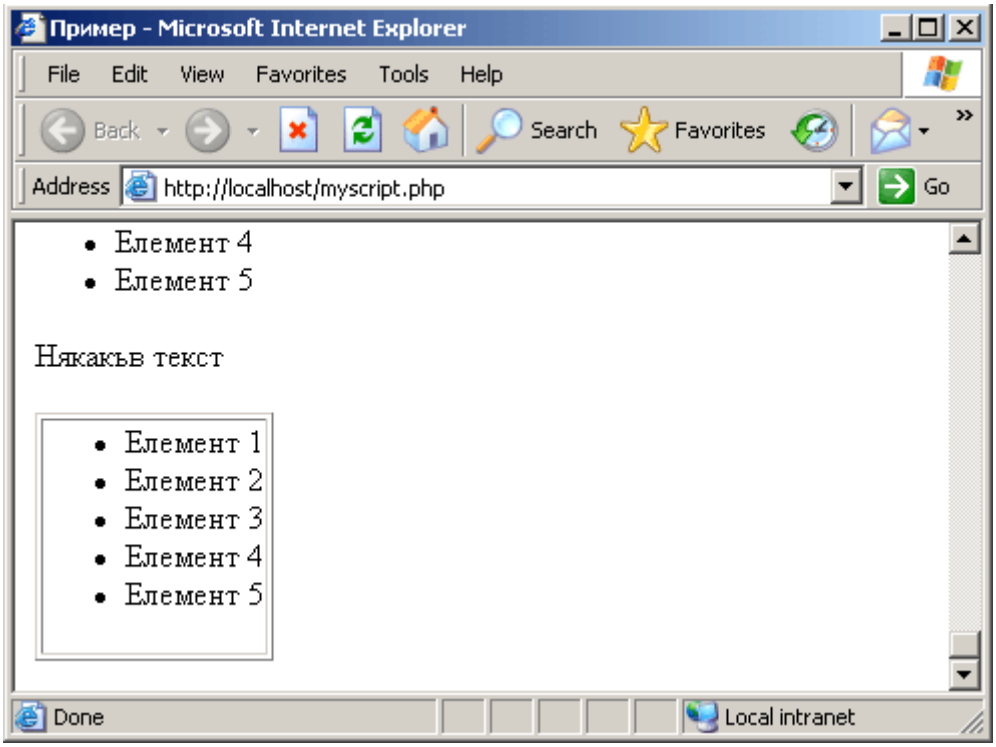
Функцията е отличен инструмент за манипулиране на уеб документи и заради това, че след като веднъж е създадена (декларирана) в един скрипт, тя може да бъде ползвана и в други скриптове без да се налага да я декларирате отново.

Например след горния примерен код, където декларирахме и извикахме 2 пъти функцията mylist(), може да имаме някакъв текст, някакви html тагове и след това да се наложи отново да покажем същия списък с елементи:

```
<p>
Някакъв текст
</p>
<table border="1">
    <tr>
        <td>
```

```
<?php
// А тук ще извикаме функцията за трети път
mylist();
?>
        </td>
    </tr>
</table>
```

В случая ефекта от горния код ще бъде показването на списъка с 5-те елемента в таблицата. Въпреки че предния скрипт е затворен и в таблицата е сложен нов скрипт, няма проблем да се извика същата функция от първия скрипт без да е необходимо повторното и деклариране във втория скрипт. Ето какъв ще е ефекта от кода в браузъра:



Урок 11

Параметри (аргументи) на функция. Ползване на конструкцията return

1. Аргументи на функция

В кръглите скоби, които задължително се поставят след всяка функция, може да има един или повече параметъра (наричат се още аргументи на функция). Аргументите на функциите представляват променливи, които съдържат някаква стойност. Стойностите им могат да бъдат както цифрени, така и буквени. Когато са повече от една, тези променливи се отделят чрез запетаи.

Предварително дефинираните (вградените в езика) функции може да имат някои задължителни параметри. Например функцията mail(), чрез която могат да се изпращат съобщения на имейл, трябва задължително да има в кръглите скоби променливи, които съдържат имейл адреса, до който ще се праща съобщението, темата на съобщението,

съдържанието на самото съобщение и в допълнителните хедъри - имейл адреса на подателя на съобщението.

Когато програмиста създава собствена функция от него зависи дали ще й зададе някакви аргументи или не.

След като са зададени някакви аргументи в кръглите скоби на функцията те може да се използват в изпълнимия код, който се поставя в големите скоби - {}. В този код може да се зададат някакви ефекти на аргументите или извършване на някакви действия с тях.

Когато на дадена функция се задават някакви променливи във вид на аргументи, на тях още при дефинирането на функцията може да им се зададат някакви стойности, а може да им се зададе стойност и по-късно, при извикването на функцията.

Ако на даден аргумент (променлива) на функция се зададе някаква стойност при дефинирането на функцията, тогава се казва, че това е стойността на аргумента по подразбиране. Ако при извикването на функцията не бъде зададена някаква друга стойност на аргумента, се използва стойността по подразбиране. Ако обаче при извикването на фукцията се укаже друга стойност на аргумента, тогава стойността по подразбиране се пренебрегва автоматично и се работи с новата стойност.

Да направим функция podpis() и да й зададем аргументите $ime и $familia. Ще присвоим на аргумента (променливата) $ime стойността "Иван" и на аргумента (променливата) $familia ще присвоим стойността "Иванов". След това в големите скоби ще укажем при извикване на функцията името и фамилията да се извеждат на екран, като на името ще зададем ефекта "удебелен текст", а на фамилията - "удебелен и наклонен текст":

```
<?php
function podpis($ime = "Иван", $familia = "Иванов")
{
echo "<b>$ime <i>$familia</i></b>";
}
?>
```

В този случай се казва, че аргумента $ime има стойност по подразбиране "Иван" и аргумента $familia има стойност по подразбиране "Иванов".

Сега да извикаме функцията чрез кода

```
<?php
podpis(Иван, Иванов);
?>
```

или чрез кода

```
<?php
podpis();
?>
```

Ако изрично не укажем някакви други стойности, както е в случая, ще получим следния резултат:**Иван *Иванов***

Ако напишем кода:

```
<?php
podpis(Иван);
?>
```

ще получим абсолютно същия резултат, т.е. отново **Иван *Иванов***. В този случай не сме извикали втория аргумент, но по подразбиране е указано той да има стойност "Иванов" със съответния текстов ефект, затова при извикване на функцията дори ако напишем само първия аргумент (в случая това е аргумента със стойност малкото име "Иван"), все пак като резултат ще получим показването и на името и на фамилията.

Ако обаче напишем

```
<?php
podpis(Петър);
?>
```

като резултат ще получим на екрана думите **Петър *Иванов***. Това е така, защото стойността "Петър" е заместила стойността по подразбиране "Иван". Както беше обяснено, стойностите на аргументите по подразбиране се ползват само ако при извикването на функцията изрично не бъде зададена друга стойност на аргумента. В случая е зададена друга стойност - името Петър. Стойността на аргумента $familia си остава "Иванов", както е зададена по подразбиране, тъй като за този аргумент не е указана друга стойност.

Можем да декларираме същата функция и по следния начин:

```
<?php
function podpis($ime, $familia)
{
echo "<b>$ime <i>$familia</i></b>";
}
?>
```

Т.е. в този случай не сме задали на аргументите стойности по подразбиране. Тогава каквито и две имена да напишем при извикване на функцията, например

```
<?php
podpis(Петър, Петров);
?>
```

или

```
<?php
podpis(Иван, Стоянов);
?>
```

и т.н. ще получим като резултат извеждане на имената на екран, като първото име ще е с удебелен шрифт, а второто - с удебелен и наклонен. Необходимо е както при декларирането на функцията, така и при извикване на функцията да укажете и двата аргумента, отделени със запетаи, в противен случай PHP ще ви изведе съобщение за грешка.

2. Конструкция return

Конструкцията return се използва във функции, като предназначението ѝ е да върне във вид на стойност резултата от действието, зададено на функцията в големите скоби. Например кода

```
<?php
function suma($prom1, $prom2)
{
```

```
return $prom1 + $prom2;
}
echo suma(100, 50);
?>
```

или някакъв по-сложен вариант, например:

```
<?php
function suma($prom1, $prom2)
{
$prom3 = $prom1 + $prom2;
return $prom3;
}
$prom4 = suma(100, 50);
echo $prom4;
?>
```

ще ни дадат като краен резултат извеждане на екран на числото 150, тъй като сме задали като действие да се изпълни сбор от двата аргумента на функцията, а след това при извикване на функцията е зададено аргументите да имат стойности съответно 100 и 50. Ако от първия пример премахнем ключовата дума return или от втория пример премахнем реда return $prom3; няма да получим резултат.

Урок 12

Популярни вградени функции: функция за извеждане на дата и час

Вече беше споменато, че в PHP съществуват голямо количество вградени (стандартни) функции, които са с предварително дефинирани имена и параметри. Една от най-използваните вградени фунции е фунцията date(), с чиято помощ може да покажете на страницата си текущата дата и час.

Функцията date() има няколко параметъра, които отговарят за различните части от записа на датата и часа - има отделни параметри за ден, месец, година, час, минути и секунди. Някаква комбинация от тях, според желанието ви, трябва да се запише в кръглите скоби на функцията.

Съдържанието на функцията date() може да бъде изведено на екран чрез командата echo, следвана от функцията и нейните параметри. Самата функция може да бъде присвоена като стойност на някаква променлива, след което съдържанието й да бъде показано на екрана чрез echo.

I. Параметри за извеждане на дата

a) Извеждане на текущата година - За показване на текущата година (year) се ползва Y или y. Главната буква Y показва годината в пълен формат, например 2006, а малката y показва само последните 2 цифри, например 06.

b) Извеждане на текущия месец - За показване на текущия месец (month) се ползва M или m. Главната буква M показва английското съкратено название на месеца, например Jan (от January) за януари, а малката буква m показва месеца в двуцифров формат, например 01 за януари и т.н. За да покажете пълното название на месеца (на английски) трябва да използвате главната буква F.За да покажете месеца в едноцифров формат за месеците от 1-ви до 9-ти трябва да ползвате малката буква n.

c) Извеждане на текущия ден - За показване на текущия ден (day) се ползва D или d. Главаната буква D показва английското съкратено название на деня от седмицата, например Mon (от Monday) за понеделник, а малката буква m показва числото на деня от месеца в двуцифров формат, например 01 за първи и т.н. За да покажете пълното английско название на деня (напр. Monday) трябва да използвате малката буква l.За да покажете деня в едноцифров формат за дните от 1-во до 9-то число трябва да използвате малката буква j.

Можем да комбинираме няколко от тези параметъра, например като спазваме реда ден-месец-година, и да покажем датата на екран по следния начин:

```
<?php
echo date("d.m.Y");
?>
```

При така зададени параметри функцията ще съдържа датата изцяло в цифров формат, като деня и месеца ще са показани с по две цифри, а годината - с четири цифри.

Нека да присвоим функцията date("d.m.Y") като стойност на променливата $mydate и да покажем резултата на екран. Кода може да изглежда по следния начин:

```
<html>
<head>
<title>Извеждане на текущата дата</title>
</head>
<body>
<?php
$mydate = date("d.m.Y");
echo "Днес е $mydate";
?>
</body>
</html>
```

Ако например текущия ден е 15-ти август 2006 година, тогава горния код ще изведе на екран съобщението:**Днес е 15.08.2006**

II. Параметри за извеждане на час

a) Извеждане на час - С помощта на date() може да покажете и точния текущ час. За показване на часа (hour) се ползва H или h. Голяма буква H извежда часа в двуцифров 24 часов формат (от 00 до 24), а малката h го извежда в двуцифров 12 часов формат (от 01 до 12).За да покажете часа в едноцифров за часовете от 0-вия до 9-тия 24 часов формат (от 0 до 24) трябва да използвате главната буква G.За да покажете часа в едноцифров за часовете от 1-вия до 9-тия 12 часов формат (от 1 до 12) трябва да използвате малката буква g.

b) Извеждане на минути - Чрез малката буква i можете да покажете минутите в двуцифров формат от 00 до 59.

c) Извеждане на секунди - Чрез малката буква s се показват секундите в двуцифров формат от 00 до 59.

Да комбинираме някои от тези параметри и да покажем на екран текущия час:

```
<?php
echo date("H:i:s");
?>
```

Можем да усложним примера като зададем датата за стойност на променливата $mydate и часът за стойност на променливата $mytime:

```
<?php
$mydate = date("d.m.Y");
$mytime = date("H:i");
echo "Здравейте! Днес е $mydate, часът в момента е $mytime";
?>
```

Ако например текущата дата е 15 август 2006, а текущия час е 4 и половина следобед, резултата от горния код ще бъде съобщението:**Здравейте! Днес е 15.08.2006, часът в момента е 16:30**

Урок 13

Популярни вградени функции: функция за вмъкване на един документ в друг

Отличен инструмент за улеснение при изграждане на уеб страници е функцията include(). Чрез нея може да включвате различни файлове в "тялото" на други файлове. Това спестява доста труд, като в същото време прави кода много по-изчистен и лесен за манипулиране.

Синтаксисът на функцията е **include("име на файл")**, т.е. в кръглите скоби се записва името на файла, който трябва да бъде извикан. Например за да бъде включен в дадена страница файла menu.php трябва на мястото в страницата, където желаем да се вижда съдържанието на menu.php да напишем

```
include("menu.php")
```

По този начин може да извикате множество различни файлове на различни места в даден уеб документ.Да предположим, че имате сайт с множество страници, като всяка страница е изградена от 3 части:

- горна част (header), в която са разположени логото на сайта и хоризонтално меню с връзки
- ляво меню с връзки
- централна част, където е поместена някаква информация, различна за всяка страница

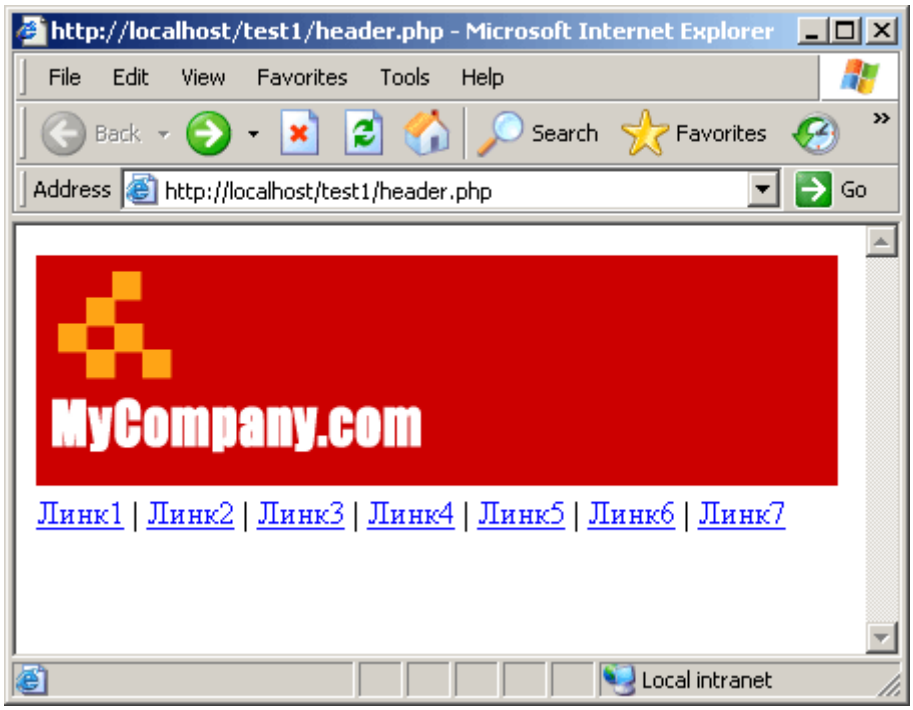
Два от тези три елемента - горната част и лявото меню - се повтарят във всяка от страниците. Ако трябва да направите всички страници само с HTML ще се наложи да копирате навсякъде един и същ код за хедъра и лявото меню. Може да избегнете това, като направите отделни документи за горната и лявата част на страниците и след това ги включите чрез фунцията include() във всички файлове на местата, където трябва да се покажат.

Нека да направим отделна php страница за горната част, която да представлява таблица с два реда. В първия ред ще поставим графичен файл с логото на сайта, а втория ще представлява хоризонтално меню с няколко линка. Отворете някакъв текстов редактор, например Notepad, и напишете в него следния код:

```
<table width="400" border="0" cellspacing="0" cellpadding="0">
<tr>
  <td>
<img src="header.jpg" width="400" height="200" />
  </td>
</tr>
<tr>
```

```
  <td>
<a href="page1.php">Линк1</a> |
<a href="page2.php">Линк2</a> |
<a href="page3.php">Линк3</a> |
<a href="page4.php">Линк4</a> |
<a href="page5.php">Линк5</a> |
<a href="page6.php">Линк6</a> |
<a href="page7.php">Линк7</a>
  </td>
</tr>
</table>
```

След това съхранете страницата като header.php. Предварително трябва да имате изработен графичния файл header.jpg, който изпълва "главата" на страницата и съдържа логото на сайта. Файла може да има следния вид:



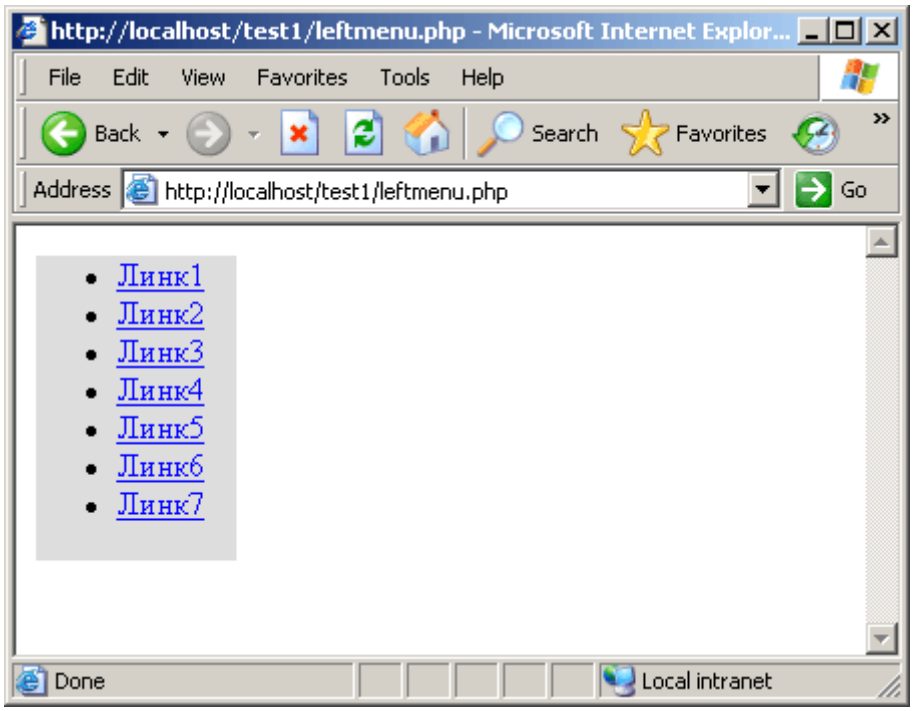
Забележете, че в така написания код няма никакви отварящи или затварящи документа тагове, освен таговете на самата таблица. Няма и никакви php елементи. Езика PHP позволява да вземете произволно "парче" HTML код и да го съхраните във вид на самостоятелна PHP страница като просто го запишете във файл с разширение .php Тези качества на езика са му създали славата на гъвкаво и мощно средство за манипулиране на уеб документи.

Сега да създадем лявото меню, което също ще поместим в таблица и ще съхраним като отделен файл под името leftmenu.php За целта напишете в текстовия редактор следния код:

```
<table bgcolor="#dddddd" width="100" border="0" cellspacing="0" cellpadding="0">
<tr>
  <td>
<ul>
<li><a href="page1.php">Линк1</a>
<li><a href="page2.php">Линк2</a>
<li><a href="page3.php">Линк3</a>
<li><a href="page4.php">Линк4</a>
<li><a href="page5.php">Линк5</a>
```

```
<li><a href="page6.php">Линк6</a>
<li><a href="page7.php">Линк7</a>
</ul>
  </td>
</tr>
</table>
```

Запишете горния код като leftmenu.php Вида му ще бъде следния:



Сега остава да създадем една от страниците, където файловете header.php и leftmenu.php ще бъдат вмъквани чрез функцията include(). Например това може да бъде началната страница на сайта index.php За да я създадете сложете в редактора следния код:

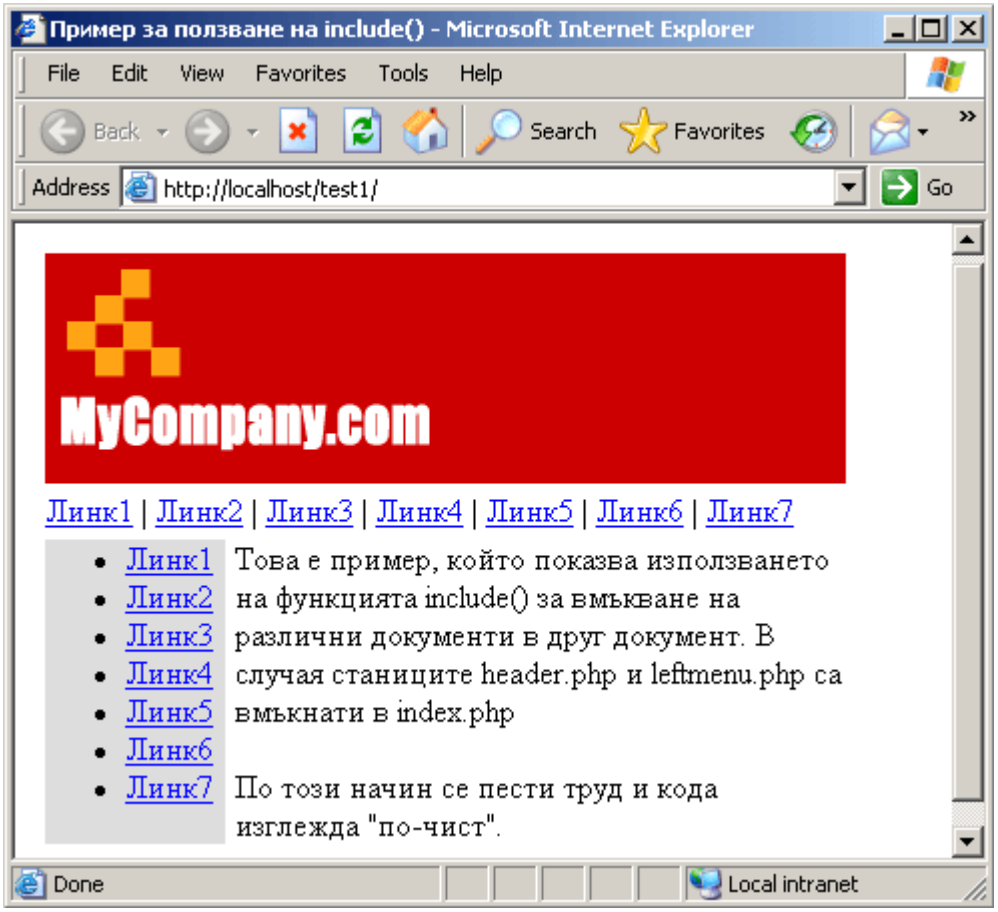
```
<html>
<head>
<title>Пример за ползване на include()</title>
</head>
<body>
<table width="400" border="0" cellspacing="5" cellpadding="0">
<tr>
 <td colspan="2">
<?php
include ("header.php");
?>
 </td>
</tr>
<tr>
 <td>
<?php
include ("leftmenu.php");
?>
 </td>
```

```
 <td>
<p>
```

Това е пример, който показваизползването на функцията include() за вмъкване наразлични документи в друг документ. В случая станиците header.php и leftmenu.php са вмъкнати вindex.php

```
</p>
<p>
По този начин се пести труд и кода изглежда "по-чист".
</p>
 </td>
</tr>
</table>
</body
</html>
```

Запишете горния код като файл с название index.php Когато заредите index.php в браузъра включените страници ще се появят съответно в горната и лявата част на документа:



По същия начин можете да включите файловете header.php и leftmenu.php в произволно количество страници. Ако решите да промените нещо по хедъра или лявата част на страниците ще се наложи на направите промени единствено в двата файла, като тези промени веднага ще се отразят във всички страници, където сте включили файловете.

В случай, че трябва да вмъкнете в някакъв документ външен за вашия сайт файл ще е необходимо да запишете пълния път до файла, например:

```
<?php
include("http://domain.com/folder/file.php");
?>
```

Урок 14

Изработване на проста форма за обратна връзка (Feedback Form)

Поставянето на форма за обратна връзка (форма за контакти, feedback form, contact form) вече е почти задължително за един уеб сайт. Формата за обратна връзка е много полезен инструмент, който дава възможност на посетителите на сайта веднага да осъществят контакт с вас като ви изпратят своите въпроси, коментари и пр. директно от страницата за контакти, без да е необходимо да отварят електронната си поща. Затова е силно препоръчително на страницата ви за контакти да имате и форма за обратна връзка, освен адрес/и, телефон/и и имейл/и, които също трябва да бъдат записани на тази страница.

За да направите скрипт за пращане на съобщения от сайта ви трябва да използвате вградената функция **mail()**. Фукцията mail()може да съдържа в кръглите скоби няколко елемента (наричат се аргументи), като някои от тях са задължителни.

По-конкретно, в кръглите скоби задължително трябва да присъстват следните елементи във вид на низове или като имена на променливите, на които е зададено да съдържат тези елементи като стойност:

- имейла на който ще бъде пратено съобщението (вашия имейл)
- темата (заглавието) на съобщението
- текста на самото съобщение
- допълнителен хедър с имейла на потребителя, т.е. имейла на който да изпратите вашия отговор

При записването на елементите в кръглите скоби трябва да се спазва точно този ред в който те са изброени по-горе. Отделните елементи се отделят един от друг със запетаи. Следователно синтаксиса на функцията е следния:

mail(вашия имейл, тема, съобщение, имейл на подателя)

За да може текста на съобщението на потребителя да се разположи във вид на аргумент в кръглите скоби на функцията mail(), след написването му той трябва да бъде присвоен като стойност на някаква променлива, например $saobshtenie. В такъв случай в кръглите скоби съобщението ще присъства под формата на променливата $saobshtenie, разположена на трета позиция - след имейла на който се изпраща съобщението и заглавието на съобщението. Имейла на който ще се изпраща съобщението (вашия имейл) може да се зададе директно като низ, например yourname@domain.com

Съдържанието на първия и втория аргумент на функцията mail() - вашия имейл и темата на съобщението - влизат в горната част (header) на електронното писмо. Това е частта на всеки имейл, която се намира над самото съобщение. Електронния адрес на който е изпратено писмото(вашия имейл, на който ще получавате съобщенията от формата) се намира в хедъра на реда обозначаван обикновено като "To:" или "До:", а темата (заглавието) на съобщението се намира на реда обозначаван обикновено като "Subject:" или "Тема:".

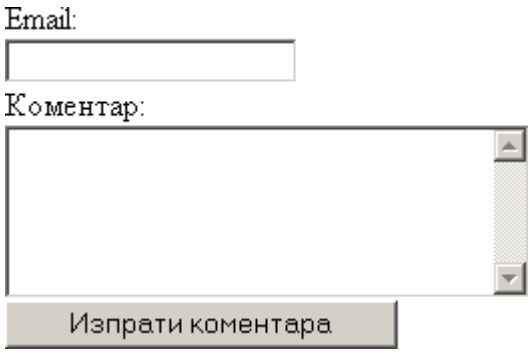
След хедъра на писмото следва основната му част - самия текст на съобщението (message), който се задава като трети аргумент на функцията mail(). След това в кръглите скоби на mail() може да има допълнителни хедъри, които в електронните писма са обозначени като "From:" или "От:", "Cc:" или "Копие до:" и "Bcc:" или "Скрито копие до:". От тях задължителен е допълнителния хедър "From:", който съдържа информация за имейла на подателя на съобщението.

Както беше споменато, информацията от 4-те аргумента на функцията mail() може да бъде включена в кръглите скоби под формата на низове от букви и символи или във вид на имена на променливите, които съдържат като стойност съответните низове. Например като втори елемент (тема или заглавие на съобщението) може да се зададе заглавие във вид на низ (дума или няколко думи), което ще бъде едно и също при всички изпратени съобщения, например "Съобщение от формата за контакти" или "Коментар от потребител" и т.н. Ако във формата има специално поле за заглавие (тема) на съобщението, тогава всеки път ще имаме различно заглавие, написано от съответния потребител. В такъв случай ще извличаме това заглавие от формата чрез масива $_POST и ще го присвояваме на някаква променлива, например $zaglavie, така че заглавието на писмото ще присъства в кръглите скоби на функцията mail() под формата на променливата $zaglavie.

Първо да направим самата форма, като ползваме познатите HTML тагове. Най-простата форма за обратна връзка трябва да съдържа поне 2 полета: едно поле за имейла на посетителя на който вие да изпратите отговора си и едно многоредово поле за самото съобщение. Формата може да представлява нещо подобно:

```
<html>
<head>
<title>Форма за обратна връзка</title>
</head>
<body>
<form action="contactscript.php" method="POST">
Email:<br />
<input type="text" name="email" /><br />
Коментар:<br />
<textarea name="komentar" cols="30" rows="5">
</textarea><br />
<input type="submit" value="Изпрати коментара" />
</form>
</body>
</html>
```

Във формата имаме две полета, на които сме задали съответно имената "email" и "komentar". Тези имена могат да бъдат и всякакви други, т.е. наименованията на полетата ги избирате вие. На атрибута action сме задали като стойност името на скрипта, който ще обработва нашата форма за контакти, в случая contactscript.phpСъхранете горния код като html или php файл, например contactform.htmlВ браузъра формата ще изглежда по следния начин:



Сега трябва да направим скрипта contactscript.php, който ще изпраща съобщението в някаква ваша пощенска кутия.

Скрипта може да изглежда по следния начин:

```
<?php
$email = $_POST["email"];
$message = $_POST["komentar"];
mail("yourname@domain.com", "Съобщение от формата за контакти", $message, "From:
$email");
?>
Вашето съобщение беше изпратено успешно.
```

От материалите за променливите тип array (масив) знаете, че към различните стойности на даден масив може да се обръщате като записвате в квадратни скоби идентификационния номер или ключа на съответната стойност. В горния код извличаме чрез масива $_POST съдържанието на полето за имейла на потребителя и съдържанието на многоредовото поле със самото съобщение. Това правим като записваме ключовете на тези стойности в квадратните скоби след името на масива $_POST. Ключове за стойностите на тези полета са названията на полетата, указани във формата чрез атрибута name. Т.е. поставяме в квадратните скоби наименованията на полетата от формата, в случая "email" и "komentar".

След това задаваме двете стойности на масива $_POST като стойности съответно на променливите $email и $message. По този начин тези две променливи ще съдържат имейла на потребителя и неговото съобщение. Наименованията на променливите $email и $message са произволно избрани и могат да бъдат всякакви други каквито пожелаете.

Накрая чрез функцията mail() задаваме аргументите:

- имейла на който да се прати съобщението, в случая yourname@domain.com (трябва да го замените с вашия имейл адрес)
- заглавието на съобщението, в случая "Съобщение от формата за контакти"
- съдържанието на самото съобщение, което се явява стойност на променливата $message
- имейла на потребителя, който се явява стойност на променливата $email - той ще се покаже в хедъра на имейла и е указан чрез "From: $email"

Веднага след края на скрипта сме задали изречението "Вашето съобщение беше изпратено успешно.", което ще се показва при успешно пращане на съобщение.

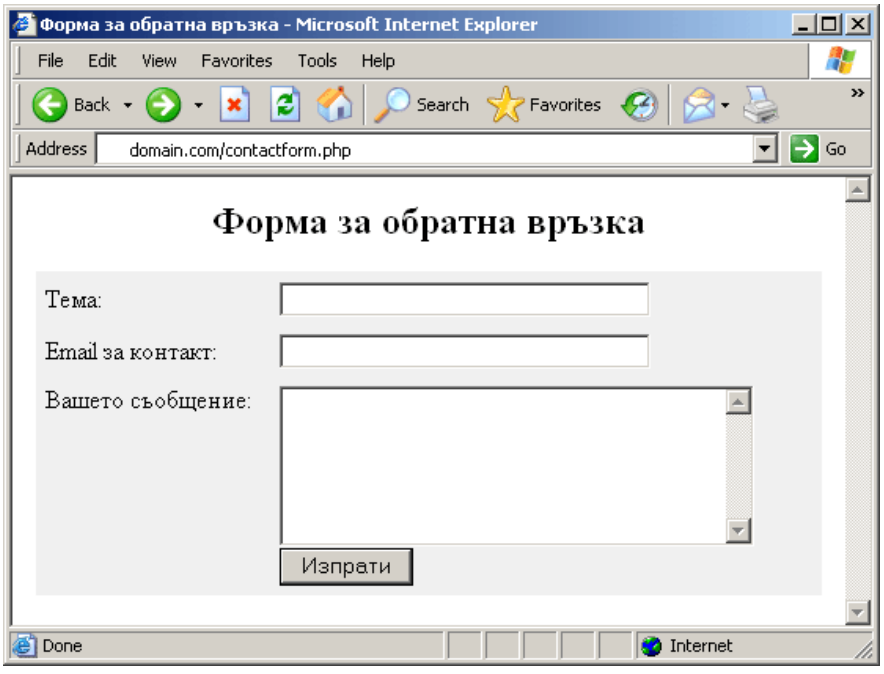
Както се вижда, в горния скрипт заглавието на съобщението е предварително зададено във вид на низ като 2-ри аргумент на функцията mail() и винаги ще бъде "Съобщение от формата за контакти". Може да усложним малко формата, като направим още едно поле специално за темата на съобщението, където потребителите сами да въвеждат заглавия на своите съобщения. Освен това ще поставим формата в таблица, за да подобрим външния й вид. Кода може да бъде следния:

```
<html>
<head>
<title>Форма за обратна връзка</title>
</head>
<body>
<h2 align="center">Форма за обратна връзка</h2>
<form action="contactscript.php" method="POST">
<table bgcolor="#f0f0f0" width="500" border="0" align="center" cellspacing="3"
cellpadding="3">

<tr>
  <td width="140">
Тема:
  </td>
```

```
  <td>
<input type="text" name="zaglavie" size="35" />
  </td>
</tr>
<tr>
  <td width="140">
Email за контакт:
  </td>
  <td>
<input type="text" name="email" size="35" />
  </td>
</tr>
<tr>
  <td width="140" valign="top">
Вашето съобщение:
  </td>
  <td>
<textarea name="saobshtenie" cols="35" rows="6">
</textarea>
<br />
<input type="submit" value="Изпрати" />
  </td>
</tr>
</table>
</form>
</body>
</html>
```

В браузъра формата ще изглежда по следния начин:



Скрипта contactform.php може да изглежда така:

```
<?php
  $email_na_podatel = $_POST["email"];
```

```
  $zaglavie_na_saobshtenie = $_POST["zaglavie"];
  $saobshtenie = $_POST["saobshtenie"];
  $ot_kogo = "From: $email_na_podatel";
mail("yourname@domain.com", $zaglavie_na_saobshtenie, $saobshtenie, $ot_kogo);
?>
<html>
<head>
<title>Успешно пратено съобщение</title>
</head>
<body>
<h4>Благодарим ви! Вашето съобщение беше изпратено успешно.</h4><a
href="http://domain.com">Кликнете тук</a> за да се върнете на началната страница.
</body>
</html>
```

Разликата между първия и втория скрипт е, че сега сме задали съдържанието на полето за заглавие на съобщението като стойност на променливата $zaglavie_na_saobshtenie, а съдържанието на допълнителния хедър, в който е имейла за обратна връзка с потребителя, сме задали като стойност на променливата $ot_kogo.

Веднага след скрипта имаме HTML код, който при успешно пращане показва съобщението "Благодарим ви! Вашето съобщение беше изпратено успешно." и съдържа връзка за връщане към началната страница на сайта.

За да работят и двата скрипта е необходимо да замените примерния имейл yourname@domain.com с вашия собствен имейл адрес.

Имайте предвид, че подобен скрипт ще ви върши работа само ако посетителите попълват коректно съответните полета. Ако обаче някой посетител забрави да напише например имейл адреса си вие ще получите съобщението, но няма да знаете къде да пратите отговора. Затова по принцип подобни скриптове съдържат по-сложен код, който примерно да извършва проверка дали не е оставено празно някое поле и дали в полето за имейла на потребителя присъстват символите "маймунско а" (@) и "точка" (.), които се съдържат във всеки един имейл адрес, т.е. проверява се дали е правилен формата на имейл адреса. Такива скриптове може да намерите и да си изтеглите от различни сайтове за безплатни PHP скриптове.

Трябва също да се има предвид, че скриптовете за пращане на съобщения ще работят от личния ви компютър само ако на него имате инсталиран mail сървър. Затова за да може да се пращат съобщения от подобни скриптове те трябва да са качени на хостинг, който има мейл сървър и поддържа PHP.

Урок 15

Изработване на проста Login форма

Често в сайтовете се налага да се дава ограничен достъп до някои части от сайта, например до определени директории или страници. Ще изработим проста login форма за достъп до определени места от сайт, защитени с парола и потребителско име.

Кода на формата може да бъде следния:

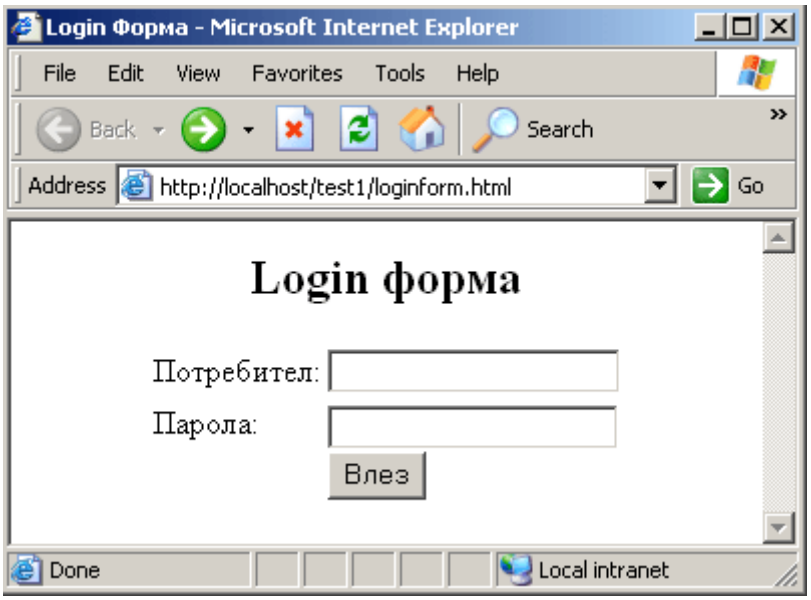
```
<html>
<head>
<title>Login Форма</title>
<body>
```

```
<h2 align="center">Login форма</h2>
<form action="login.php" method="post">
<table align="center" border="0">
<tr>
  <td>
Потребител:
  </td>
  <td>
<input type="text" name="potrebitel" />
  </td>
</tr>
<tr>
  <td valign="top">
Парола:
  </td>
  <td>
<input type="password" name="parola" /><br />
<input type="submit" value="Влез">
  </td>
</tr>
</table>

</form>

</body>
</html>
```

Съхранете горната страница като loginform.html В браузъра тя ще изглежда по следния начин:



Сега трябва да направим скрипта login.php Той може да бъде следния:

```
<?php
$user = $_POST["potrebitel"];
$pass = $_POST["parola"];
if ($user == "user1" && $pass == "pass1")
```

```
{
header("location: http://domain.com/secretpage.php");
}
else
{
echo "<b>Грешно потребителско име или парола!</b>";
}
?>
```

Съхранете кода като login.php

В горния код чрез масива $_POST вземаме от формата съдържанието на полетата за потребителско име и парола като задаваме в квадратните скоби ключовете за тези стойности, т.е. имената на полетата, които са potrebitel и parola. Присвояваме ги като стойности съответно на променливите $user и $pass.

След това чрез конструкцията if-else задаваме на скрипта следната задача:

Ако е изпълнено условието стойността на променливата $user да е user1 И стойността на променливата $pass да е pass1, тогава отвори уеб адреса http://domain.com/secretpage.php Ако това условие не е изпълнено, тогава покажи съобщението "Грешно потребителско име или парола!".

Това означава, че ако използвате в login формата потребителско име user1 и парола pass1 в браузъра ви ще се зареди страницата, до която желаете да има ограничен достъп. В примера тази страница е с название secretpage.php, а адреса и е http://domain.com/secretpage.php, където domain.com е вашия домейн.

Ако искате достъпа да е с друго потребителско име и парола трябва да замените user1 и pass1 с някакви друго съчетание от букви и/или цифри.

За редиректване на бразъра използваме функцията **header()**, чиито синтаксис е

header(Location: URL)

На мястото на уеб адреса (URL) може да се изпише пълния адрес на страницата, която трябва да се зареди (както е в примера), а може да се укаже и само името на страницата, ако тя се намира в същата папка (директория), в която се намират формата и скрипта. Тогава редиректването ще е във вида

header("location: secretpage.php");

В този пример са зададени само една парола и едно потребителско име за достъп до защитената страница. С помощта на конструкцията switch може да направим множество акаунти за различни потребители, които да имат достъп до различни части от сайта и да ползват за целта различни потреителски имена и пароли. В този случай скрипта login.php може да изглежда по следния начин:

```
<?php
$user = $_POST["potrebitel"];
$pass = $_POST["parola"];

switch(true)
{
case ($user == "user1" && $pass == "pass1"): header("location:
http://domain.com/secretdir1/secretpage1.php");
break;
```

```
case ($user == "user2" && $pass == "pass2"): header("location: secretdir2/secretpage2.php");
break;

case ($user == "user3" && $pass == "pass3"): header("location: anothersecretpage.php");
break;

default: echo "<b>Грешно потребителско име или парола</b>";
}
?>
```

Чрез горния скрипт създадохме 3 акаунта за достъп с потребителски имена и пароли съответно user1/pass1, user2/pass2 и user3/pass3, които водят до различни страници от сайта.