КНИГА 1
Beginners

Въведение в PHP

Какво представляват настоящите материали по PHP?

Представените материали по PHP са предназначени за юзери, които имат базови познания по HTML, но никога не са се занимавали както с PHP, така и с какъвто и да е друг програмен език. Поне основното познаване на HTML е единственото задължително условие за работа с материалите.

В настоящия справочник са представени основите на PHP като е използван максимално опростен език и са давани прости и лесни за разчитане примери. Силно препоръчително е не само да изчетете, но и да си "поиграете" с дадения прост примерен код като заменяте различни елементи и стойности, за да видите какви други ефекти могат да се получат.

След като имате базовите познания по езика е препоръчително да се запознаете с официалното ръководство по PHP на адрес www.php.net и с допълнителна литература по PHP.

Кратка история на PHP

PHP (пълно название PHP: Hypertext Preprocessor) е понастоящем най-популярния сървърен (server-side) скриптов език. Създаден е през 1994 от датчанина Размус Лердорф (Rasmus Lerdorf). Първото му приложение е било скрипт, който следял посетителите на личната страница на Лердорф. Първата версия на езика била наречена Personal Home Page Tools 1. 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 езика е пренаписан от Зеев Сураски (Zeev Suraski) и Анди Гутманс (Andi Gutmans)- пренаписаната версия получава названието PHP 3. От този момент нататък PHP започва да се превръща в най-популярния език за динамични уеб разработки.

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. През 2005 г. цифрата на сайтовете използващи PHP е близо 25 милиона.

В основата си синтаксиса на PHP е близък с този на езиците C, Java и Perl, но в сравнение с тях PHP е значително по-опростен и по-гъвкав.

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

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 2 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 и "послушен". 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.

Какъв софтуер е нужен, за да работи PHP?

Както беше споменато, 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

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.

Основи на синтаксиса на PHP. Коментари.

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

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>";

?>

Както в HTML съществува таг за коментар (<!--Коментар-->), така и в PHP могат да се пишат коментари, които не влияят върху изпълнението на скрипта и служат за записване на текстове за обяснение и подсещане. Съществуват тагове за коментар на един ред и таг за коментар на няколко реда.

Таговете за едноредов коментар са // и #
Тага за многоредов коментар е /* */

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 */

Променливи

Какво е променлива? Имена на променливи. Задаване стойност на променлива.

1. **Какво е променлива? Имена и синтаксис на променливи.**
   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" (=). За различните видове оператори и действията, които се извършват с тях, ще стане дума по-късно в материалите за PHP оператори.

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 а е равно на 30. При разчитането на този код трябва да кажем "На променливата а е присвоена стойността 30". Може да си представяте променливата $a като сандък, върху който е изписана буквата **а** 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.

Типове стойности на променливи

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)

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

```
<?php

$celochislena_promenliva1 = 102;
$celochislena_promenliva2 = 12895;
$celochislena_promenliva3  = -37;
$celochislena_promenliva4  = 1;
```

```
?>
```

2. 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;

?>
```

3. 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;

?>
```

4. 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.

5. Накрая накратко ще разгледаме още един 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..

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.

Предварително дефинирани променливи: Променливи на средата

В PHP съществуват и предарително 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.

Оператори

Какво е оператор? Аритметични оператори и оператори за присвояване. Приоритет на оператори.

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 (+).

В PHP съществуват няколко вида оператори. Основните от тях са представени в този и следващия материал:

1. 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.

2. 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.

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.

Оператори за сравнение и логически оператори

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).

3. Оператори за сравнение

== 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.

4. 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.

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.

Конструкции

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.

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:
1. дали тези променливи са равни и по стойност и по тип
2. 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.

Какво е цикъл? 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:

1. Присвоени са стойностите 100 и 50 съответно на променливите $prom1 и $prom2
2. Зададено е условието - докато стойността на $prom2 е по-малка от стойността на $prom1 прибавяй по една единица към стойността на $prom2
3. 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:

1. 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).
2. Поставя се условието, като цикъла се изпълнява докато поставеното условие връща резултат "истина".
3. 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).

Функции

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 опредена 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.

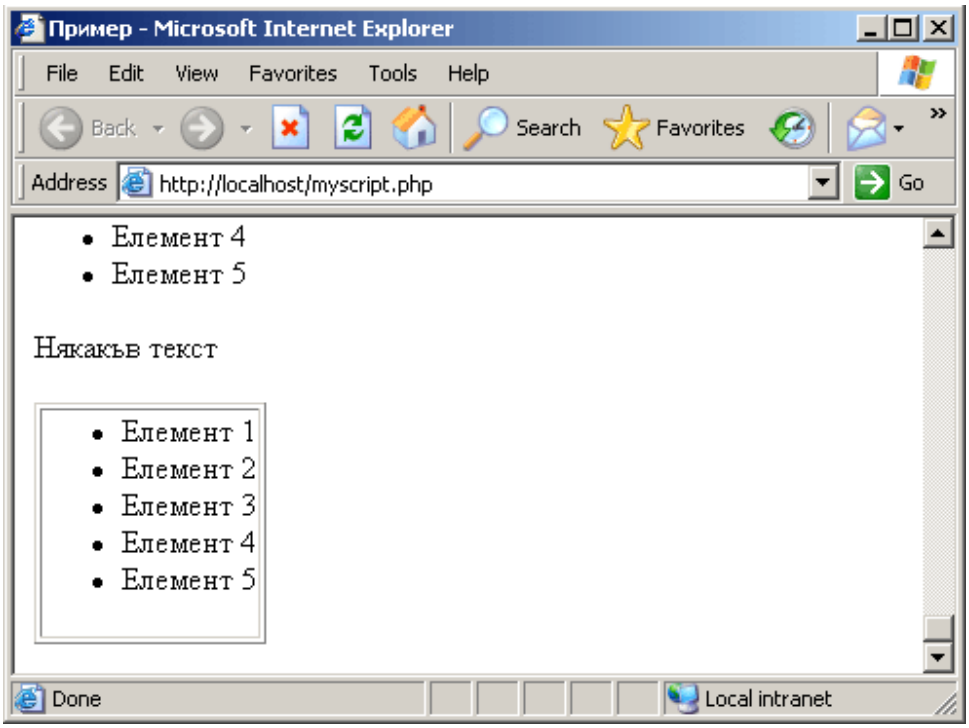
Обърнете внимание, че кръглите скоби задължително се изписват след името на функцията, независимо, че в тях може да няма никакви параметри, както е в случая. Конструкцията на извикването на функция, също като другите конструкции, завършва с точка и запетая.

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:



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 й 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.

Масиви

Какво е масив? Деклариране на масиви.

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).

Вградени функции за работа с масиви - 1-ва част

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
";}**

?>

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.

Вградени функции за работа с масиви - 2-ра част

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

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.

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!

Константи

Какво е константа? Дефиниране на константи. Синтаксис.

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.

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.

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

PHP примери

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

1. 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.

2. 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.

3. 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

1. 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.

2. Извеждане на минути

Чрез малката буква i можете да покажете минутите в двуцифров формат от 00 до 59.

3. 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

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.

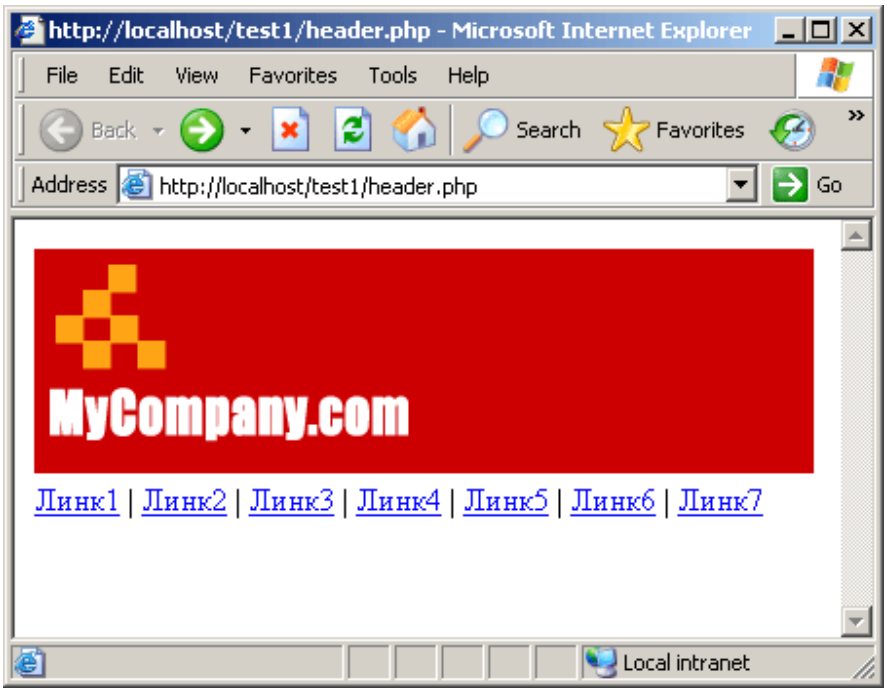
Нека да направим отделна php страница за горната част, която да представлява таблица с два реда. В първия ред ще поставим графичен файл с логото на сайта, а втория ще представлява хоризонтално меню с няколко линка. Отворете някакъв текстов редактор, например Notepad, и напишете в него следния код:

```
<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:



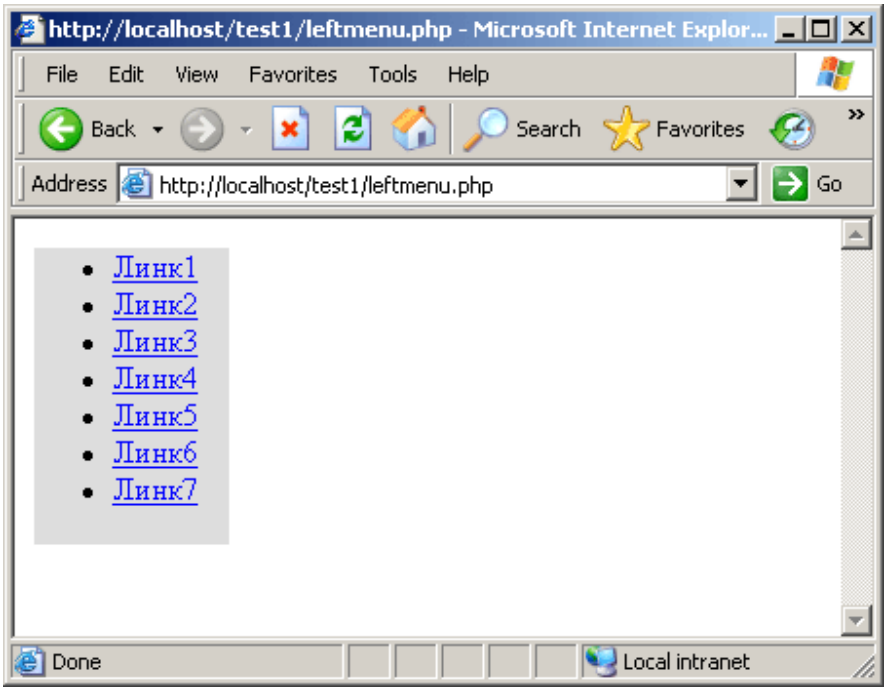
Забележете, че в така написания код няма никакви отварящи или затварящи документа тагове, освен таговете на самата таблица. Няма и никакви php елементи. Езика PHP позволява да вземете произволно "парче" HTML код и да го съхраните във вид на самостоятелна PHP страница като просто го запишете във файл 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>
```

**
**
</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>

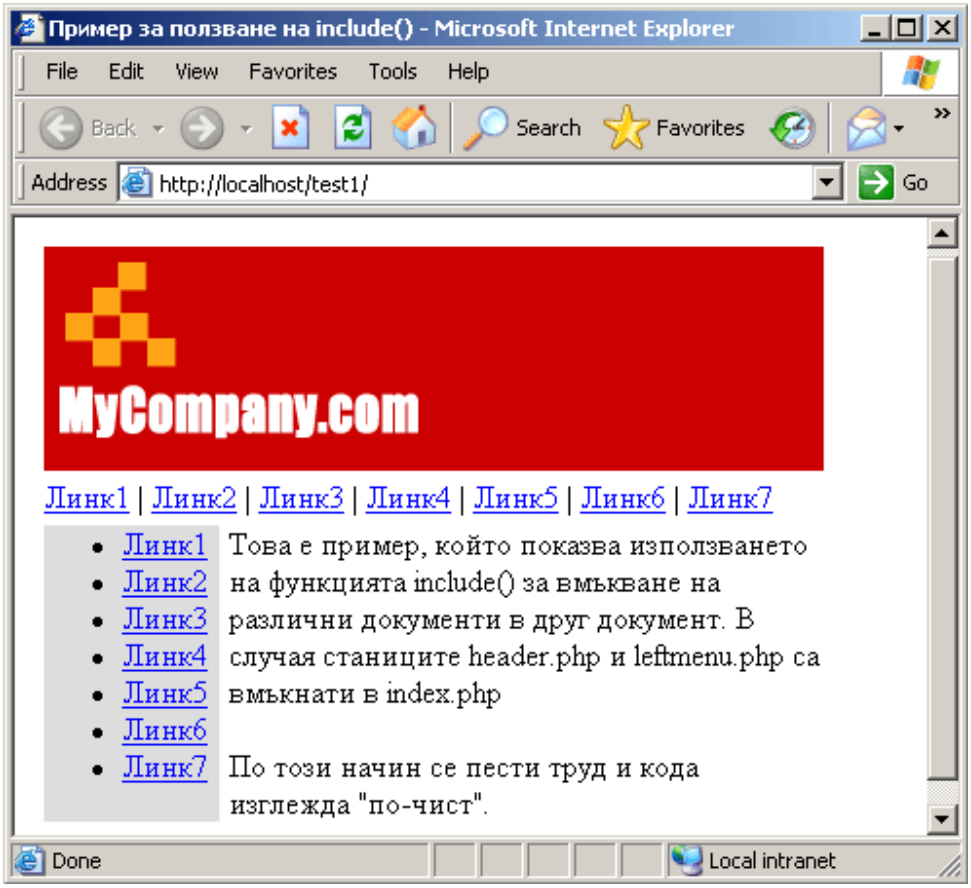
<table width="400" border="0" cellspacing="5" cellpadding="0">

<tr>
<td colspan="2">

<?php

```
include ("header.php");
?>

  </td>
</tr>

<tr>
  <td>

<?php
include ("leftmenu.php");
?>

  </td>
  <td>
<p>
Това е пример, който показва
използването на функцията include() за вмъкване на
различни документи в друг документ. В случая станиците
header.php и leftmenu.php са вмъкнати в index.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");

?>
```

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:

1. имейла на който ще бъде пратено съобщението (вашия имейл)
2. темата (заглавието) на съобщението
3. текста на самото съобщение
4. 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:

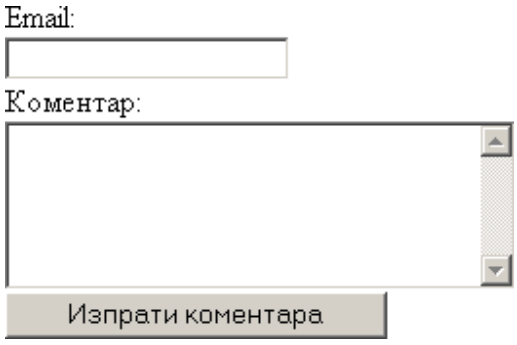
```
<html>
<head>
<title>Форма за обратна връзка</title>
</head>

<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.php

Съхранете горния код като html или php файл, например contactform.html

В 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:

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

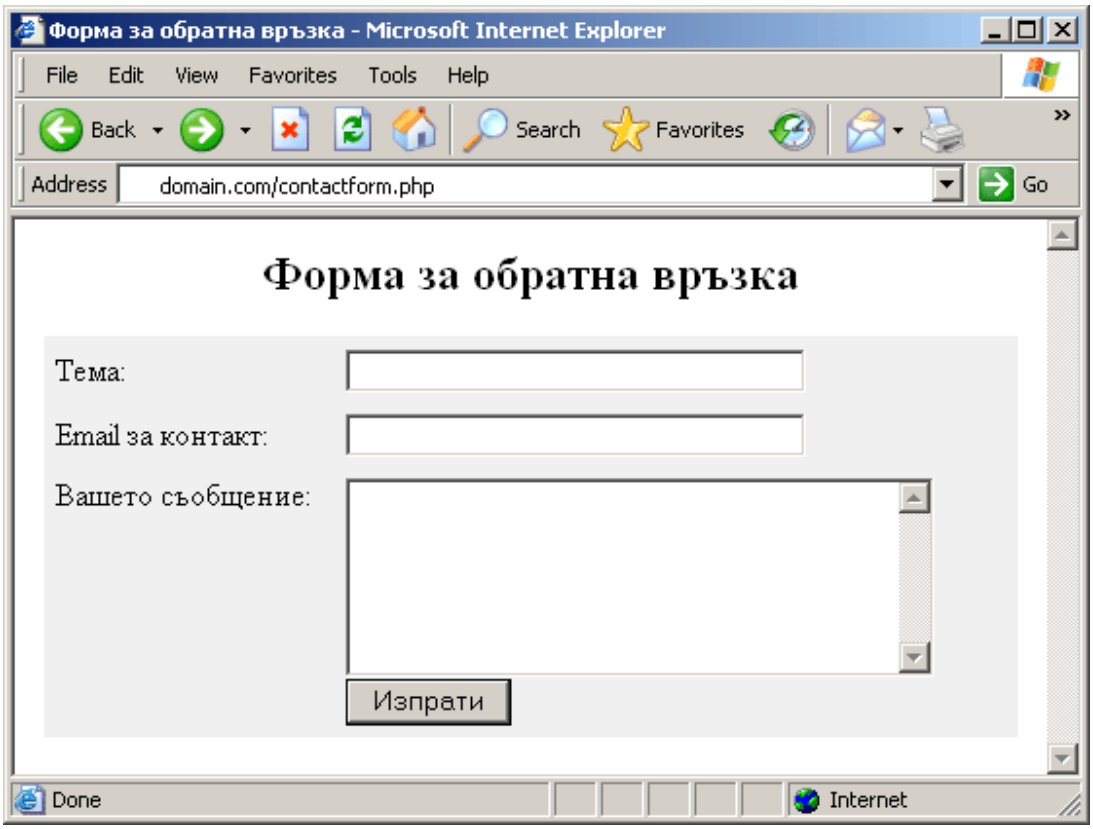
Веднага след края на скрипта сме задали изречението "Вашето съобщение беше изпратено успешно.", което ще се показва при успешно пращане на съобщение.

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.

Изработване на проста Login форма

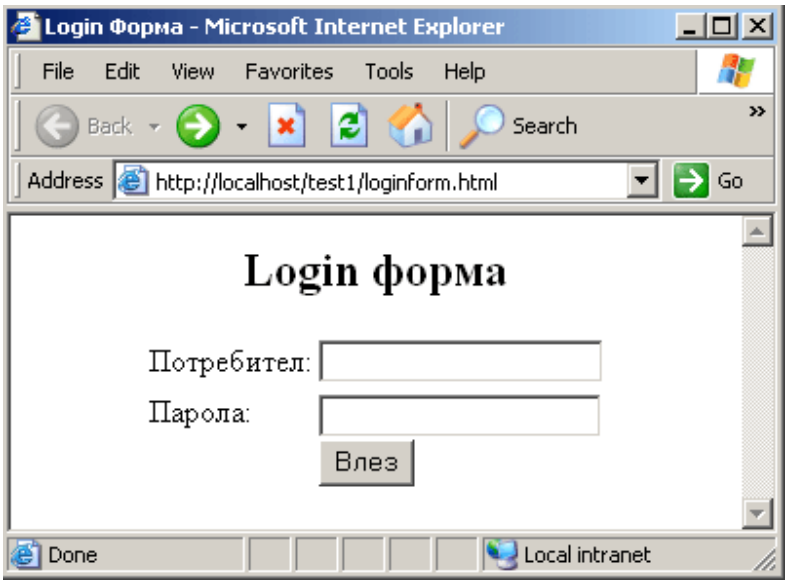
Често в сайтовете се налага да се дава ограничен достъп до някои части от сайта, например до определени директории или страници. Ще изработим проста login форма за достъп до определени места от сайт, защитени с парола и потребителско име.

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.

КНИГА 2
PHP1

Увод в PHP

Най-популярният език за създаване на интерактивни и динамични web сайтове е PHP.
Това е не просто твърдение, а реален факт според изследователските фирми. Причините са много - гъвкав, компактен, лесен за изучаване и използване, предлагащ всички възможности за създаване на динамичен, съвременен сайт, задоволяващ и най-големите капризи...
Създаден като личен проект, той бива преработен от група разработчици и бързо намира приложението си в Интернет. За да улесни грижите си за собствения си личен сайт, през 1994 година Размус Лердорф създава скриптов език, наричайки го Personal Home Page Tools. От това наименование по-късно излиза и съкращението PHP.
Нововъведението бързо събужда интереса на множество разработчици и скоро след това първата версия на езикът е вече факт. Съкращението PHP вече се разбира като PHP: Hypertext Preprocessor, но основната идея - простотата на използване, е запазена. PHP и до днес остава силен скриптов език с много възможности за web програмиране. И може би най-лесния за изучаване и прилагане.

Факт е, че PHP позволява да постигнете определен резултат само с няколко реда код, докато ако използвате други програмни езици за целта може да се наложи да изпишете десетки редове. Това е основната причина, поради която езикът получи толкова бърза и голяма популярност сред web разработчиците.Използването на PHP спестява време, а и позволява създаването на интерактивни сайтове дори от начинаещи. Малко по малко почти всички хостващи компнии започват да предлагат поддръжка за езика.

Програмни езици

Езиците, които програмистите използват, за да накарат компютъра да направи това, което искат, се наричат програмни езици. Но едва ли можем да очакваме че машината ще прочете сложните комбинации от букви и цифри, и ще разбере какво очакваме от нея. За компютъра такива редове са безсмислени :

```
<?
echo "Здравейте";
?>
```

За да може машината да изпълни желанието ни, което в случая беше да изпише на екрана "Здравейте", тази инструкция трябва да се преведе на нейния език. Това може да стане чрез така наречените интерпретатори, които "смилат" инструкциите в кода и ги подават на компютъра в разбираем за него вид или чрез предварително превеждане на кода в такъв вид - компилация. От тази гледна точка програмните езици се делят на скриптови и компилируеми.
Кодът, написан на скриптов език, се обработва от междинна програма - интерпретатор, в момента на изпълнението си. А кодът, написан на компилируем език трябва да се приведе предварително във вид, разбираем за машината. Затова във втория случай имаме два вида код - изходен (source code) и изпълним (executable).
За пример можем да посочим Java и Java Script, които в съзнанието на повечето хора са почти едно и също. Разбира се, това не е вярно. Въпреки че имат сходен синтаксис, двата езика са доста различни по начина си за използване. Java е компиулируем език. Файловете, съдържащи изходния код на Java, имат разширения .java. Ако ги отворим, в тях ще видим всички инструкции, които програмистът е написал, във вид, разбираем за човека. Но тези файлове не могат да бъдат изпълнени в този си вид. Те трябва да бъдат компилирани - приведени в двоичен вид. Изпълнимите файлове на Java имат разширение .class и ако ги отворим с текстов редадактор ще видим поредица от символи, която няма да означава нищо за нас.
От друга страна Java Script е език, който се интерпретира в момента

на изпълнението си. Когато отворите web страница, съдържаща в себе си Java Script код, той се обработва в реално време от интерпретатор, вграден в браузъра, и след това се подава за изпълнение. По - старите браузъри, които нямат вграден такъв интерпретатор, няма да се справят със задачата.
PHP е скриптов език. Когато потърсите с браузъра си PHP страница в Интернет, кодът се обработва в момента, в който сървърът изпълнява заявката ви, от програма, намираща се на него. От тази гледна точка, разликата между двата скриптови езика - Java Script и PHP е в това, че кодът на първия се интерпретира от браузъра ви(web клиента), а на втория - от програма на сървъра. Затова се казва, че JS е клиентски език (client side), а PHP - сървърен (server side).

Основата

Голямата популярност на PHP се дължи главно на гъвкавостта му при включване в HTML документи. Възприето е мнението, че концепцията на езика е смесица от Perl, Java и C, но най-голяма е близостта с C. Така PHP предлага лесни възможности за изпълняване на сложни математически изчисления, изпълняване на мрежови функции, възможностти за обработка на електронна поща, работа с обикновени изрази и множество други. Но безспорно най-голямата сила на езика е във възможностите му за работа с бази данни. Осигурена е поддръжка на най-разпостранените бд, като MySQL, PostgreSQL, Oracle, Sybase, mSQL и други. Връзката с базата данни и работата с нея са облекчени и позволяват лесното създаване на ефективни динамични сайтове.

От начало

След толкова уводни думи вече е крайно време да направим първия си PHP скрипт. Макар и банално, ще накараме компютъра да изпише "Здравей свят!"

```
<HTML>
<HEAD>
```

```
<META HTTP-EQUIV="Content-Type" CONTENT="text/html;
charset=windows-1251">
<TITLE>Здравей, свят!</TITLE>
</HEAD>
<BODY>
Първият ми PHP скрипт<br>
<?
echo ("<b>Здравей, свят!</b>");
?>
</BODY>
</HTML>
```

Когато запишете този файл и го сложите на сървър, поддържащ PHP, след като го извикате в браузъра си ще видите изписано там поздравлението. Разбира се, полза от такъв скрипт няма, но той ще ни помогне да посочим основните положения.
За разлика от cgi файловете, не е необходимо да променяте разрешенията на файла с PHP код. Необходимо е просто да го сложите на сървъра. Няма значение и дали ще използвате бинарен трансфер или ascii, въпреки че е препоръчително да използвате втория метод.
PHP кодът се поставя между **<? и ?>.** Така сървърът разбира, че трябва да подаде това съдържание на интерпретатора за PHP. Може да се използва и друг вариант - **<?PHP ...?>.** Изборът е ваш.
Всяка инструкция трябва да завършва с точка и запетая. За да изпишете текст на екрана можете да използвате вградените функции echo или print. В горния пример можехме да постигнем идентичен резултат използвайки инструкцията print ("Здравей, свят!"); Изразите в PHP се ограждат с единични или двойни кавички.
Примерът демонстрира как може да се вгради скриптовия код в HTML. Нека сега направим обратното, имайки предвид, че всичко, което подадем като аргумент на echo ще бъде изведено на екрана.

```
<?
```

```
echo "<HTML>
<HEAD>
<META HTTP-EQUIV="Content-Type" CONTENT="text/html;
charset=windows-1251">
<TITLE>Здравей, свят!</TITLE>
</HEAD>
<BODY>
Първият ми PHP скрипт<br>
<b>Здравей, свят!</b>
</BODY>
</HTML>";
?>
```

Резултатът от този код ще бъде абсолютно същия. Но може би забелязахте обратната наклонена черта, поставена преди кавичките в реда, където се указва енкодинга на страницата. По този начин указваме на интерпретатора, че трябва да изобрази кавички на екрана, а не че низът завършва. Много грешки в PHP скриптовете се дължат на неправилно боравене с тях.

Кавичките

Както вече стана дума, един израз може да се огради с единични или с двойни кавички. Двата начина на запис

```
echo ("<b>Здравей, свят!</b>");
```

и

```
echo ('<b>Здравей, свят!</b>');
```

ще доведат до един и същ резултат. Но това важи само за изразите. Разлика между двата записа има, когато използвате променливи, но за това ще стане дума по-късно.
Нека сега включим кавички в самия израз. Примерно за да изпишем "Този скрипт се казва "Здравей, свят!".

Ако използваме единични кавички, за да укажем на интерпретатора, че това изречение е един низ, то проблеми няма.

```
echo ('<b>Този скрипт се казва "Здравей, свят!"</b>');
```

В случая интерпретаторът е отчел отварящите единични кавички и очаква затварящите, за да отчете края на низа. Но ако използваме двойни кавички, ще се получи объркване:

```
echo ("<b>Този скрипт се казва "Здравей, свят!"</b>");
```

След като запишем и видим скрипта така, ще получим съобщение за грешка - "Parse error".
Причината е, че интерпретаторът очаква да види затварящ двойни кавички, за да отбележи край на низа. Тоест в случая това е "<b>Този скрипт се казва ". Останалият текст, обаче, не може да се впише по никакъв начин в синатаксиса на PHP и интерпретаторът не може да го разбере. Затова и дава грешка при обработването (парсването).
За да решим проблема, трябва да използваме обратно наклонената черта. Така ще укажем, че става дума за символ, който трябва да се изобрази, а не да се интерпретира :

```
echo ("<b>Този скрипт се казва \"Здравей, свят!\"</b>");
```

Сега вече всичко ще бъде наред.
Обратно наклонената черта може да ни послужи и за вкарване на нов ред (\n), на табулация (\t) и така нататък.

Коментари

Поставянето на коментари в кода е полезно, не само когато работите с големи файлове. Понякога, когато се връщате към скрипт, който сте писали преди по-дълго време, може да се окаже, че коментарите са полезни и на самите вас. Друга полезно приложение е, когато имате нужда временно да отстраните известно

количество код. Вместо да го изтривате и след това възстановявате, можете просто да го коментирате.
В PHP имате възможност да коментирате съдържанието на 1 ред или на цял блок редове.
За да поставите коментар на 1 ред можете да използвате символа # или две наклонени черти (//). А за да коментирате блок от редове, оградете го с /* и */.
Ето и един пример:

```
<?
/* Този скрипт ще изпише на екрана коя е датата днес.
Затова ще използваме функцията date(), за да я разберем.*/
$today = date("Y-m-d");
//А сега ще изпишем датата.
echo "<CENTER>Днес е: $today.</CENTER>";
//И това е всичко.
?>"
```

Усложняване на примера

Нека сега добавим няколко неща към скрипта си. Ще приветстваме посетителя на страницата си, ще му кажем датата и часа и ще го "стреснем", като му покажем с какъв браузър е дошъл и от коя страница. Така примерът ще добие повече смисъл.

Примерен резултат от този скрипт виждате на изображението.
Използвахме функцията date() за да разберем колко е часът и коя е датата според сървъра, а така наречените "променливи на срадата" за да изясним от къде е дошъл посетителя и с какъв браузър (в случая IE 5.5).

Променливите

Една от най-големите сили на PHP е възможността му да работи с променливи. Няма да се впускаме в теоретични подробности за

същността и концепцията им, а ще демонстрираме основните правила при използването им.
Най-общо, променливата се състои от две "части" - име и стойност. Обръщайки се към името на променливата можем да й присвоим нова стойност или да получим достъп до текущата.
Има две основни неща, които трябва да имате предвид, работейки с променливи в PHP. За разлика от повечето програмни езици, тук те не се декларират предварително. Създаването на променливата става в момента, в който и присвоявате стойност. В PHP името на променливата винаги е предшествано от знак за долар - $. Без значение дали и присвоявате стойност или използвайте стойността и по някакъв начин.

```
<?
$ime="Иван Петров";
echo $ime;
?>
```

Тези редове код ще изпишат на екрана съдържанието на променливата $ime, което сме дефинирали по-рано. Ако бяхме пропуснали да й присвоим стойност, на екрана нямаше да се изпише нищо. Проверката за това дали на определена променлива е присвоена стойност е много полезна и често използвана възможност за проверка на това дали определено събитие се е случило или не. Как става това ще демонстрираме по-късно.

Видове променливи

PHP предлага големи облекчения при работат с променливите, което в началото би ви се сторило странно ако сте се занимавали с други програмни езици. Стана вече дума, че не е необходимо да декларирате променивата преди да й присвоите стойност. Друга важна особеност е, че не е необходимо да указвате какъв тип информация се съдържа в нея - дали е низ (string), цяло число (int), число с плаваща запетая (float) и т.н. PHP сам определя типа на променливата в зависимост от съдържанието й. Според някои

напреднали програмисти това е по-скоро вредно, но както скоро ще се убедите, освен улеснението, гъвкавостта на PHP в това отношение има и множество други плюсове.
Съществуват няколко вградени функции с чиято помощ можете да разберете какъв тип е дадена променлива. Това са
is_bool() - връща истина ако променливата е булева (истина/лъжа);
is_float() - връща истина ако променливата съдържа число с плаваща запетая;
is_integer() - връща истина ако променливата е цяло число;
is_string() - връща истина ако променливата е низ от символи;
is_array() - връща истина ако променливата е масив;
is_object() - връща истина ако променливата е обект;

Enviroment variables

Съществува един особен вид променливи, наричани "променливи на обкръжението". Най-общо казано, това са променливи, поддържани от сървъра и PHP интерпретатора, описващи текущото състояние на връзката и характеристиките на двете страни в нея - клиент и сървър.
В предната част на този материал използвахме две от тези промливи - $HTTP_USER_AGENT и $HTTP_REFERER, с чиято помощ открихме с какъв браузър посетителят разглежда страницата и от къде е дошъл на нея.
Събирането и анализирането на подобни данни е безкрайно полезно за всеки web разработчик. Така можете да научите кой сайт ви праща най-много посетители и да организирате рекламната си стратегия, наблягайки на него. Полезно е също и да следите кои от страниците в сайта ви се посещават повече и предизвикват по-голям интерес, така че да наблегнете на тях и на подобен вид съдържание в по-нататъчното изграждане на сайта.
Пълен преглед на променливите на средадата може да видите, използвайки вградената в PHP функция - phpinfo(). Напишете в текстов файл следните редове :

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

Запишете го на сървъра с име info.php, примерно, и го отворете в браузъра си. Ще видите най-важната информация за PHP инсталацията, както и всички създадени в текущата връзка променливи на средата на PHP и сървъра. Изписаното в левите полета на таблицата представляват имената на променливите, а в дясната част - стойностите им.
Нека сега разширим примера и добавим още малко информация в него.

```
<HTML>
<HEAD>
<META HTTP-EQUIV="Content-Type" CONTENT="text/html;
charset=windows-1251">
<TITLE>Здравей свят!</TITLE>
</HEAD>
<BODY>
<?
$data=date("d-m-Y");
$chas=date("H:i:s");
$browser=$HTTP_USER_AGENT;
$ref=$HTTP_REFERER;
$user_ip=$REMOTE_ADDR;
$forw=$HTTP_X_FORWARDED_FOR;
$serv=$SERVER_SOFTWARE;
$rem_host=gethostbyaddr($user_ip);
echo ("<b>Здравейте,<br><br>Добре дошли в сайта!</b><br>");
echo ("Сега е $chas часа, по моя часовник, на $data, а Вие дойдохте
тук от $ref с $browser<br>");
echo ("Вашето IP е $user_ip($forw) - $rem_host. <br><p>Поздрави:
<br>$serv");
?>
```

</BODY>
</HTML>

Нека сега обясним отделните редове в него. Нямаше нужда да използвам променливите $browser, $ref, $user_ip, $forw, $serv и $rem_host, но ги включих за по-голяма прегледност. Всяка от тях получи за стойност стойността на някоя от променливите на средата.
На променливата с име "browser" бе присвоена стойността на $HTTP_USER_AGENT, която винаги съдържа идентификационната информация на браузъра. В случая "Mozilla/4.0 (compatible; MSIE 5.5; Windows 98)". Браузърът Opera позволява на потребителите си да променят идентификацията й, така че тя може да се представя като IE или Netscape. Причината е в рестрикциите, които някои сайтове налагат на използващите браузър, различен от Internet Expolorer. Обикновено това също става с помощтта на $HTTP_USER_AGENT. Имайки предвид особеностите на различните браузъри, можете да покажете на посетителя на сайта си страница, оптимизирана за неговата програма, съдейки по съдържанието на тази променлива на средата. В скоби казано, възможността на Opera да се представя по различен начин може да ви изправи пред куриозната информация, че посетителят използва IE под операционна система Linux.
На променливата с име "ref" бе присвоена стойността на $HTTP_REFERER, съдържаща URL на мястото, от което идва посетителят. Вече обяснихме по-подрбоно тази променлива, но трябва да имате предвид, че тя не винаги дава полезна информация. В конкретния случай едва ли мога да очаквам, че Yahoo са поставили на началната си страница линк към моя тестов файл.
Променливата, наименувана "user_ip" получи стойността на променливата на средата $REMOTE_ADDR. Тя винаги съдържа в себе си IP адреса на посетителя. Тук много рядко може да се получи разминаване на стойността с истинската. Защото сървърът използва

стойността й за да определи на кого да изпрати поискания файл. Много често се случва множество потребители да се намират зад едно или повече прокси сървъри. Няма да коментираме сега защо и как се получва това. В случая трябва да имаме предвид, че ако посетителят е зад прокси, то $REMOTE_ADDR ще покаже IP адреса на проксито. За адреса на клиентската машина се създава нова променлива - $HTTP_X_FORWARDED_FOR. Повечето прокси сървъри осигуряват тази информация, особено ако администраторите им не желаят да изкарат живота си по съдебните зали.
Използването на последните две описани променливи на средата е полезно, когато давате достъп до чувствителна информация или по една или друга причина е важно да знаете кой точно е отсреща. Така можете да ограничите достъпа до сайта си на хора, които нарушават добрия тон в него или да подсигурите сигурността му при e-commerce приложения.
Тук използвахме и още една променлива на средата - $SERVER_SOFTWARE, която съдържа в себе си информация за сървърния софтуер - версия на сървъра и операционна система. В горния пример използвахме това, като "подпис" на съобщението.

Видове променливи

PHP предлага големи облекчения при работата с променливите, което в началото би ви се сторило странно ако сте се занимавали с други програмни езици. Стана вече дума, че не е необходимо да декларирате променивата преди да й присвоите стойност. Друга важна особеност е, че не е необходимо да указвате такъв тип информация се съдържа в нея - дали е низ (string), цяло число (int), число с плаваща запетая (float) и т.н. PHP сам определя типа на променливата в зависимост от съдържанието й. Според някои напреднали програмисти това е по-скоро вредно, но както скоро ще се убедите, освен улеснението, гъвкавостта на PHP в това отношение има и множество други плюсове.
Съществуват няколко вградени функции с чиято помощ можете да разберете какъв тип е дадена променлива. Това са

is_bool() - връща истина ако променливата е булева (истина/лъжа);
is_float() - връща истина ако променливата съдържа число с плаваща запетая;
is_integer() - връща истина ако променливата е цяло число;
is_string() - връща истина ако променливата е низ от символи;
is_array() - връща истина ако променливата е масив;
is_object() - връща истина ако променливата е обект;

Изпращане на променлива

Съществуват три начина по които можете да изпратите стойността на променлива към един PHP скрипт. Това са методите POST, GET и чрез cookies. Нека ги илюстрираме чрез прост пример. Ще създадем две страници - първата ще съдържа форма, в която потребителите ще напишат името и възрастта си, а втората ще изпише тези данни на екрана. Ще наречем първия файл index.html, а втория - show.php.
Нека напишем HTML формата, в която посетителите ще попълнят данните си :

```
<!-- index.html -->
<p>Здравейте! Моля попълнете следната информация : <br>
<form action="show.php" method="post">
Вашето име : <input type="text" name="ime"><br>
Възраст : <input type="text" name="godini"><br>
<input type="submit" value="Изпрати!">
</form></p>
```

Когато посетителят въведе даните и натисне бутона "Изпрати!", тези данни ще достигнат до файла show.php във вид на променливи. Можете да ги използвате веднага. Тук е мястото да уточним, че за разлика от други езици, PHP не ви кара да проверявате дали информацията е изпратена чрез GET или POST. Не е необходимо и да правите нищо, за да получите достъп до съхраняваната в променливите данни. PHP се грижи за това и сам създава

променливите. Затова можем направо да ги използваме, например - да ги изпишем:

```
<?
//show.php
echo "<p>Здравейте, $ime, вие сте на $godini години! ";
?>
```

Очевидно това не е най-смисления ред код, който можете да напишете, но по-късно ще разгледаме възможностите, които ни дава информацията за възрастта на посетителя, за да го пренасочим към съответното място в сайта.
Най-често няма значение кой от двата метода - POST или GET използвате, с изключение на случаите когато трябва да изпратите файл или по-дълга информация.
Най-просто казано, GET използва така нареченото URL кодиране, тоест така данните се дописват след името на файла в неговия URL. Ако промените значението на "action" във формата в index.html от POST на GET и изпратите информация, ще видите в адресното поле на браузъра си нещо като show.php?ime=Ivan&godini=25. При изпращане на данните чрез POST, те се предават в тялото на заявката.
Много често информацията се задава на скрипта не чрез HTML форми, а пряко чрез URL. Това е така, защото в съвременните динамични сайтове един PHP файл би могъл да генерира множество web страници в зависимост параметрите, които му се подадат.

URL кодиране

За да постигнете това, трябва да зададете необходимите параметри в адреса, с който викате файла. Данните се поставят след името му, като за разделяне се използва въпросителния знак (?). Той указва края на URL и началото на допълнителните данни, известни още като "query string". Там данните се описват по двойки (име=стойност), като всяка двойка се разделя със знака "&" (амперсанд):

файл.php?име1=стойност1&име2=стойност2&име3=стойност3

При отварянето на файл по този начин, в него можем свободно да боравим с променливите "имеХ", които PHP създава и на които присвоява стойностите "стойностХ".
Както можете да си представите, в този начин за предаване на информацията се крият и някои опасности. Нека си преставим, че потребителят въведе двете си имена, разделени с интервал. Тогава бихме получили нещо като

show.php?ime=Иван Петров&godini=25.

Очевидно е, че в този случай няма да получим желания резултат, тъй като интервалът след името ще се възприеме като край на URL и всичко последващо ще се игнорира. Неточни резулати биха се получили и при използване на занака амперсанд (&), %, + и някоии други. Затова информацията трябва да се пригоди преди да бъде изпратена чрез така нареченото URL кодиране. В най-общи линии интервалите се заместват със знак плюс (+), а някои от символите с "%xx", където xx са ascii кодовете на съответните символи в шестнадесетичен вид.
Така ако посетителят (в примера - Иван Петров) напише като възраст 26&7 (за 26 години и 7 месеца), то това ще се предаде така :

show.php?ime=Иван+Петров&godini=26%267
(& се представя чрез %26)

Не е необходимо да мислите за тези усложнения, по простата причина че PHP се грижи сам за правилното кодиране и декодиране на данните в адреса. Но понякога ще се налага да използвате функцията urlencode, за да си осигурите правилна обработка.

низ urlencode (низ)

Така можехме да извикаме спокойно show.php без да се

притесняваме за интервалите и специалните символи. Например

```
echo '<a href="show.php?'.urlencode (ime=Иван Петров&godini=25);
```

Всички двойки от име и стойност се съхраняват в служебната променлива $QUERY_STRING, а ако в конфигурационния файл на PHP е включена опцията track_vars, то данните, изпратени към скрипта могат да се открият и в масивите $HTTP_POST_VARS и $HTTP_GET_VARS, в зависимост от метода, използван за изпращането им. Така че, ако потребителят попълни формата, която включихме в index.html, можем да открием името му в $HTTP_POST_VARS['ime'].

Масиви във формите

PHP предоставя допълнително облекчение, когато трябва да обработвате по-големи и сложни форми. Можете да зададете стойностите в масиви. Например така :

```
<p>Здравейте! Моля попълнете следната информация : <br>
<form action="show.php" method="get">
Вашето име : <input type="text" name="ime"><br>
Възраст : <input type="text" name="godini"><br>
Интереси : <select multiple name="interesi[]">
<option value="1">Интернет
<option value="2">Музика
<option value="3">Кино
<option value="4">Филателия
</select>
<input type="submit" value="Изпрати!">
</form></p>
```

Сега всички избрани от посетителя възможности ще се съхраняват като елементи в масива "interesi[]". Как се обработват масивите в PHP ше изясним по-късно.

Cookies

Невъзможно е да си представим добър и полезен сайт, който не използва "бисквитки" напоследък. Въпреки опитите на някои политически организаци да ограничат използването им в Европа, очевидно е, че ако това, стане световната мрежа би загубила поне половината от потенциала си, а защо не и много повече. Факт е, че положителните страни от използването на cookie файловете са много повече от лошите. При това същите щети могат да се нанесат лесно и чрез други технологии. Затова мога да ви посъветвам да използвате "бисквитки" винаги, когато смятате, че ще имате полза от тях, без да се притеснявате от странични аргументи.

Има две много важни неща, които трябва да знаете, преди да започнете да включвате cookie в сайтовете си. Първото е, че достъп до информацията във всеки такъва файл има само собствения ви домейн. Това, което вие запишете на машината на посетителя не може да бъде прочетено от други, съответно и вие не можете да прочетете записаното от други. Но тук идва и едно предупреждение. Бисквитките са прости текстови файлове, които се предават по Интернет и ако не използвате подсигурена връзка са също толкова незащитени от зложелателна намеса, колкото и цялата информация по мрежата. Затова никога не слагайте в тях пароли, адреси, номера на кредитни карти или друга подобна информация, освен ако тя не е по някакъв начин кодирана. Най-добрата тактика е да записвате някакъв идентификатор, по който можете да разпознавате посетителя, а информацията му да държите в някаква собствена база данни.

Още трябва да знаете, че големината на един такъв файл не може да е повече от 4k. Не можете да запишете на твърдия диск повече от 20 "бисквитки" от един домейн. От своя страна браузърът не може да поддържа повече от 300 файла от всички посещавани сайтове общо. Последното е важно, защото би могло да причини изтриване на вашия файл преди указаната в него дата.

Твърди се, че някои потребители забраняват приемането на cookie чрез съответните опции на браузъра си. Казвам твърди се, защото това звучи невероятно и едва ли някога ще срещнете такъв

посетител. Технологията е толкова масово разпостранена в Интернет и без нейна помощ огромно количество услуги няма да действат, така че вероятността такива потребители наистина да съществуват е доста малка. Можете да направите малък експеримент и да инструктирате браузъра си да иска потвърждение за всяко cookie, което приема. Най-вероятно няам да издържите още след първите два часа.
Сега няма да се впускаме в история и теория на cookie файловете, а ще демоснтрираме начина, по който можете да ги използвате в своите PHP скриптове. За повече информация можете да прочетете официалната спецификация на Netscape, която се намира на адресhttp://www.netscape.com/newsref/std/cookie_spec.html.

За или против ползването на cookie?

"Бисквитките" са малки текстови файлове, които уеб сайтовете записват на твърдите дискове на посетителите си. За разлика от рапозстранените напоследък предразсъдъци, те не са програми, а просто парчета текст. Много се изговори за потенциалната опасност от тях и за начина, по който те застрашавали правото на неприкосновеност на потребителя и неговата лична информация. Всъщност cookie файловете са необходим елемент в дейността на всеки полезен сайт и свързването им само с рекламни или следящи технологии е едностранчиво.
В тези файлове обикновено сайтовете записват важна за обслужването на посетителя информация. Без тях, например, би било невъзможно да се извърши идентификацията на потребителя за множество уеб базирани услуги. Представете си, че се налага постоянно да въвеждате името и паролата си при всяко отваряне на ново писмо, когато проверявате електронната си поща. Това би било безкрайно отегчително, бавно и неефективно. Затова когато се идентифицирате веднъж, уеб интерфейсът на e-mail услугата ще запише на cookie на твърдия ви диск информация, по която ще ви разпознава от тук нататък.
Важно е да се знае, че браузърът изпраща "бисквитката" само на

този адрес от който я е получил. Така че няма начин друг сайт да прочете информация, която не е поставена от него, като по този начин открадне важна лична информация (като парола, например). Идентификацията съвсем не е единственото приложение на cookie файловете. С тяхна помощ сайтът би могъл да ви позволи да променяте външния му вид или да изберете информацията, която да ви предлага. Използването им е особено важно в областта на онлайн търговията и рекламата. И до сега тази технология остава най-доброто решение за идентификация и персонализация на съдържаниео и услугите в Интернет.

Как да използваме cookie?

Има шест неща, които трябва да укажете при работа с една "бисквитка". Това са нейното име, съдържание, продължителност на живота, домейн, път и подсигуреност.
Името на cookie е този низ, с който по-късно ще се обръщате към нея за да получите стойността й. Продължителността на живота указва колко време желаете "бисквитката" да е валидна. Ако не сте задали време или сте го указали като 0, то файлът ще бъде изтирт от браузъра веднага след неговото затваряне. Такива cookie файлове са валидни само за текущото посечение на потребителя. Може би най-рядко ще се налага да задавате домейн за който да е валиден файла. Така или иначе не може да напишете yahoo.com и да чакате да получите файловтете, записани на твърдия диск на посетителя от портала. Браузърът така или иначе ще ви изпрати само вашите. Полза от тази възможност за настройка на cookie може да имате, ако използвате няколко субдомейна на един и същ домейн. По принцип ако "бисквитката" е зададена от уеб страница, намираща се на адрес http://www.domain.com, то тя ще се изпраща само до другите страници в този домейн, но не и до http://www.domain.com или другите негови поддомейни. как да преодолеем това ограничение ще разберем след малко. Пътят на валидност на cookie файла дава възможност да ограничите или разширите страниците, до които той ще се предава в рамките на текущия домейн. Това е полезно, ако имате няколко различни сайта като поддиректории на домейна. Сега

ще обясним това с практически примери.
За разлика от други програмни езици, PHP предлага изключително лесен начин за записване, променяне и изтриване на "бисквитки". Тук това става става само с един ред код, съдържащ функцията setcookie(). Синтаксисът е следният:

число setcookie (низ име [, низ стойност [, число продължителност [, низ път [, низ домейн [, число подсигурена]]]]])

Функцията ще върне 1, ако се е изпълнила успешно и 0 - ако не е успяла.
Запомнете, че "бисквитките" се получават и изпращат в HTPP хедърите - тоест преди каквото и да било същинско съдържание на Уеб страницата. Това е най-често срещаната причина за появата на грешки при работа с тях. Ако искате да изпратите, промените или изтриете cookie това трябва да стане преди да изпратите на браузъра каквото и да било друго. Например :

```
<?
echo "<html><head>";
setcookie ("user", "ivan");
?>
```

Този код ще върне грешка, зашото се опитвате да изпратите "бисквитка" след като вече сте пуснали някакво съдържание. HTTP хедърите нямат нищо общо с хедърите на HTML сттраниците, ограниавани между таговете <head> и </head>. Цялото съдържание на уеб страницата влиза в тялото на HTTP пакета.
Когато се опитате да изпълните този код, PHP ще ви съобщи, че хедърите вече са изпратени, като ще ви каже на кой ред в скрипта е завършило изпращането и на кой ред вие се опитвате да пуснете cookie. Това е наистина полезно, но бих искал да обърна внимание на един случай, когато е много трудно да откриете от къде идва грешката.

```
<?
setcookie ("user", "ivan");
echo "<html><head>";
?>
```

На пръв поглед проблемът би трябвало да е решен и горния код трябва да се изпълни правилно. Но поради една или друга причина в него има интервал пред отварящата скоба на PHP. Този интервал сам по себе си е изход, така че слага край на заглавните части. Затова бъдете сигурни, че "<?" е винаги в самото начало на файла. Макар и на пръв поглед дребен, този пробелм може да прични няколко часа търсене на причната си, особено ако има някоко включени един в друг файлове, затова е добре да го имате предвид. Както забелязахте от начина на представяне на синтаксиса на функцията setcookie(), единствения задължителен елемент в нея е името й. Това обаче не означава, че можете да прескачате аргументите както искате. Това е друга често срещана грешка, която води до грешно записани "бисквитки" или до невъзможността да се изтрие вече съществуваща. Ако желаете да използвате опцията път на действие, но не желаете да указвате продължителност, нямате право да напишете

```
setcookie ("user", "ivan", "/mojat_sajt");
```

Функцията си търси елементите по реда, който те бяха показани по-горе и когато ползвате някой от тях, другите в ляво от него стават задължителни. Така че верният вариант на горния ред е:

```
setcookie ("user", "ivan", 0, "/mojat_sajt");
```

Можете да "прескочите" елеметите, които не искате да задавате с 0 - когато трябва да са числа или с "" - когато трябва да са низове. Ето как ще изглежда една "бисквитка", която искаме да използваме по подигурена връзка (HTTPS), но всички останали аргументи са по подразбиране:

setcookie ("user", "ivan", 0, "", "", 1);

Така записахме "бисквитка" с име "user", стойност "ivan", валидна за всички страници в текущата поддиректория на текущия домейн при използването на подсигурена връзка.
Изтриването на cookie става, като на съществуващ файл се зададе празно съдържание. Но това трябва да стане по същия начин, по който е бил създаден. Така за да изтирем току що създаденото cookie, трябва да напишем:

setcookie ("user", "", 0, "", "", 1);

Нека сега илюстрираме всичко това с пример. Ще създадем малка система за проследяване на посетителя на адрес
http://www.domain.com/site
В началния файл, примерно index.php, включваме форма, в която очакваме да получим името/прякора на потребителя:

<form action=user.php method=post>
<input type="text" name="username">
<input type="submit" value="изпрати">
</form>

Когато потребителят въведе името си и кликне бутона за изпращане, той ше попадне на файла user.php, в който е записано:

<?
setcookie ("user", $username);
?>
По този начин всички страници, намиращи се на
http://www.domain.com/site ще имат достъп до съдържанието на cookie файла user - тоест до името, посочено от посетителя. Важно е да запомните, че това не може да стане в момента на изпращането. С други думи - user.php няма достъп до "бисквитката" в момента. Тя

ще достъпна при следващото зареждане на страница. В конкретния случай това не е проблем, защото ние все пак разполагаме с името на потребителя в променливата, използвана във формата от index.php. Така че можем да напишем :

```
<?
setcookie ("user", $username);
echo "Здравейте, $username!<br>Кликнете <a
href="index.php">тук</a>, за да смените името, което посочихте или
<a href="content.php">тук</a> за да продължите!"
?>
```

За да променим съдържанието на cookie в PHP е необходимо просто да изпратим ново със същото име и същите други атрибути, ако сме посочили такива, но с различно съдържание. Затова ако посетителят иска да смени въведеното от него име, го изпратихме отново в index.php, където ще може да повтори процедурата.
В PHP прочитането на cookie е безкрайно упростено. Тя е винаги достъпна за използване, без да е необходимо да пишете какъвто и да било предварителен код. При зареждането на PHP скрипта се създават автоматично променливи за всички "бисквитки", които сървърът получи. Името на променливата взема името на съответното cookie, а в стойността й ще открием съдържанието му. И ако основното съдържание на сайта се намира във файл content.php, в него можем да запишем :

```
<?
echo "Потребител: $user";
.....
?>
```

След това, знаейки името, можем да направим проверка в база данни за някакви настройки, записани по-рано от посетителя или да извършим други действия с информацията в $user.
Нека сега добавим още един аргумент за "бисквитката" си,

например да укажем, че тя ще е валидна в следващия един час. За целта можем да използваме вградената в PHP функция time(), от която ще вземем текущото време, след което ще прибавим 3 600 секунди:

setcookie ("user", $username, time() + 3600);

При това положение, за да изтрием "бисквитката" можем да напишем:

setcookie ("user", "", time() - 3600);

Нека сега разширим обхвата на дейност, например до Уеб страници, намиращи се в рамките на която и да е директория на www.mydomain.com/ Тъй като cookie файла бе записан от директория "site" в рамките на домейна, при досегашния начин на боравене с нея тя ще е достъпна само до PHP скриптовете в тази поддиректория. Но може да се наложи да го ползваме например в http://www.domain.com/more. За целта трябва да укажем, че съответното cookie е валидно за всички поддиректории на главната директория на домейна, която се обозначава с наклонена черта - "/". Тогава записът би изглеждал така:

setcookie ("user", $username, time() + 3600, "/");

По този начин във всеки PHP скрипт, намиращ се в която и да е поддиректория на http://www.domain.com ще можем да прочетем стойността на променливата $user, която ще ни дава въведеното потребителско име. Но какво ще стане ако посетителят отиде на http://domain.com/site/content.php? За момента бисквитката ни е валидна само за www субдомейна, но не и за останалите. За да решим проблема трябва да използваме атрибута domain и да я изпратим така:

setcookie ("user", $username, time() + 3600, "/", ".domain.com");

Обърнете внимание на точката преди името на домейна. По този начин указваме, че желаем да получаваме "user" в който и да е негов субдомейн. За да изтрием това cookie трябва да напишем :

setcookie ("user", "", time() - 3600, "/", ".domain.com");

Това са основните неща, които трябва да знаете, за да използвате "бисквитки" в сайтовете си.

-php-

Включване на файлове

PHP предоставя няколко лесни възможности да изграждате интерактивни уеб страници в зависимост от нуждите ви. Една от тях е включването на множество файлове един в друг. Освен че по този начин можете да поднасяте на посетителите си динамично съдържание, включването на файлове позволява и многократното използване на един и същи код. По този начин можете да държите конторла върху интерфейса, основните настройки или функции на доста големи сайтове.

include

Един от начините за включване на файл в текущия е чрез функцията **include()**. Тя приема като параметър пътят (абсолютен или относителен) до файла, който искаме да включим. Ако той не съществува, ще получим съобщение за грешка.
Можете да разглеждате включеният чрез **include()** файл като код, който се намира на реда, в който се намира функцията. Той всъщност се "вгражда" в тялото на декущия документ. Така всички променливи, които са дефинирани до появата на функцията ще са достъпни и за кода на включени файл. По принцип няма ограничение какъв тип ще е той - дали ще е текстов, php или някакъв друг. Различните типове файлове имат свои особености, които трябва да имате предвид.
Така например, ако във включения файл има PHP код, който

очаквате да бъде изпълнен, трябва да го заградите с . Ако извикате PHP файл от отдалечен сървър, трябва да сте сигурен, че скриптът може да бъде изпълнен там. В такава ситуация имате възможност и да предадете променливите, които ще ви трябват чрез URL, което е неприложемо за локалните включвани файлове. Ще илюстрираме казаното с прост пример. Създаваме два файла - index.php и content.php :

```
<? // Това е index.php
$name="Иван";
$ip=$REMOTE_ADDR;
include ("content.php");
?>
<? // Това е content.php
echo "Здравейте, $name!<br>Вашият IP адрес е $ip";
?>
```

Както забелязвате, в index.php зададохме стойности на две променливи, които след това изписахме. Двата файла трябва да са в една и съща директория, тъй като не сме задали друг път, когато извикахме функцията - include ("content.php");

Тук би било добре да припомним набързо, че пътищата до даден файл могат да абсолютни и относителни. Абсолютен е пътят от кореннатa директория на системата, а относителен - от единия файл до другия. Нека си представим, че index.php се намира в директория "/httpd/www/pcworld/", а content.php в "/home/www/files". В този случай бихме могли да изпишем функцията по следните два начина :

```
include ("../files/content.php");
```

или

```
include ("/home/www/files/content.php");
```

Ако content.php се намира на друг сървър, можем да го извикаме така :

include ("http://www.server.com/content.php");

В този случай можем да зададем променливите и като част от използваното URL, като index.php би могъл да изглежда :

```
<? // Това е index.php
include ("content.php?name=Ivan&ip=$REMOTE_ADDR");
?>
```

require ()

Същите правила важат и за друга вградена функция на PHP - **require()**. Всъщност, според официалната документация на PHP, това не е функция, а езикова конструкция. Една от последиците от този факт е, че не връща стойност. Т.е. не можете да проверите дали включването на допълнителния файл е било успешно или не.
За да разберем една от другите разлики между **require()** и include() трябва да обясним как става обработката на PHP кода от страна на интерпретатора. Когато сървърът получи заявка за файл, асоцииран с PHP, той подава целия код, затворен между на PHP интерпретатора (парсера). Тук кодът първо се обработва (парсва) и чак след това изпълнява. Важно е да забележите следната особеност - ако имате грешка на последния ред от кода си, той може да не се изпълни изобщо, нито ред от него, защото може да се появи грешка при обработването - Parse error. Всичко, което трябва да знаете за грешките ще откриете в следващия брой на PC World. Сега само ще направим разликата между двата варианта за включване на файл. Ето два възможни варианта на index.php :

```
<?
// Това е вариант 1 на index.php
echo "Поздрав : <br>";
$name="Иван";
$ip=$REMOTE_ADDR;
include ("content.php");
```

```
echo "Добре дошъл в сайта! <br>";
?>
```

Или :

```
<?
// Това е вариант 2 на index.php
echo "Поздрав : <br>";
$name="Иван";
$ip=$REMOTE_ADDR;
require ("content.php");
echo "Добре дошъл в сайта!<br>";
?>
```

Какво ще се случи при изпълнението на двата варианта на кода, ако content.php всъщност не съществува? В първия случй на екрана ще се изпише "Поздрав : ", а на долния ред ще се появи съобщение за грешка. Въпреки това, изпълнението на кода ще продължи и ще видим изписано и приветствието за добре дошъл.

Във втория случай, обаче, изпълнението на кода спира веднага, след липсващия файл и тук поздравът отсъства.

На пръв поглед разликата е дребна, но всъщностт е много важна, както ще се убедим след малко. Тя се подсилва и от факта, че парсерът взема съдържанието на require-натия файл още по време на предварителната обработка, а на include-натия - едва по време на изпълнението. Тоест, дори и редът в който имаме **require()** никога да не се изпълни, кодът на файла, който е там се използва при предварителната обработка.
Това е особено важно, когато искаме да правим определени проверки, в резултат на които да решим кой файл да включим. Това ще бъде демонстрирано след малко. Друга особеност на двете функции е начинът по който те се изпълняват в цикъл. Най- общо казано, **require()** е неподходяща за целта.

Избягване на повторенията

Понякога, при по-сложни включвания на файлове един в друг, може да се получи неколкократно повтаряне на кода на един и същи файл.

Това най- често води до грешки - или в обработката или в логиката на изпълнението. За да се предодвратят подобни нежелани явления може да се използват вариации на двете функции, указващи изрично, че включването трябва да стане еднократно. Конструкциите са **include_once()** и **require_once()**. В този случай, ако PHP интерпретаторът попадне на повторна инструкция да включи посочения външен файл, той ще я пренебрегне. По този начин се избягва предефиниране на функции или подмяна на стойностите на различни променливи.

Изпълними файлове

Особено място трябва да отделим на разликата между включване на файл от отдалечен сървър и на файл от локалната машина. Особено когато става дума за изпълними файлове, каквито са PHP, CGI, PL и други. Нека си представим, че искаме да включим в текущата страница perl скрипт за банерна ротация - banner.cgi. И нека този скрипт приема като променлива page адресът, където ще се покаже банера - т.е. текущата страница.
Ако той се намира на друг сървър, можем да направим включването така:

```
<?
// Включване на банер
include ("http://www.server.php/banner.cgi?page=$PHP_SELF");
echo "<br>Добре дошъл в сайта!<br>"; ?>
```

Така скриптът ще се изпълни на отдалечения сървър, а в нашия файл ще се включи резултатът от това изпълнение - в случая изображение на банер и линк към съответен сайт.
Но ако се опитаме по този начин да извикаме локален файл, резултатът ще бъде друг.

```
<?
// Включване на банер
```

```
include ("banner.cgi?page=$PHP_SELF");
echo "<br>Добре дошъл в сайта!<br>";
?>
```

При изпълнението на тези редове на екрана ще се изпише изходния код на CGI скрипта, а не резултатът от неговото изпълнение. За да предодвратите това можете да използвате virtual().

virtual ()

Функцията **virtual ()** е специфична само за Apache и казва на сървъра, че трябва да изпълни съответния файл и да върне резултата от това изпълнение. По този начин можете да изпълнявате различни видове файлове в PHP скриптовете си, но трябва да запомните, че не можете да използвате **virtual ()** за PHP файлове. С други думи, следният код е коректен:

```
<?
// Включване на банер
$page=$PHP_SELF;
virtual ("banner.cgi");
echo "<br>Добре дошъл в сайта!<br>";
?>
```

Но ако банерната система е написана на PHP, то използването на virtual () ще върне грешка. При всички случаи трябва да използвате include () или require () за да включите PHP код в PHP файл.

Примерен сайт

За да илюстрираме на практика използването на функците, нека съставим малък сайт, съдържащ основните части на този текст. Сайтът ще се състои от няколко страници с еднакъв външен вид - т.е. еднакви горна и долна част, както и еднакво навигационно меню. Сайтовете, правени по тази схема могат да се управляват лесно и са приятелски към потребителите, защото не им поднасят "изненади" на всяка страница - като променено местоположение на менюто или различна цветова схема.
Страниците в примерния сайт ще се състоят от 4 части - header.html, footer.php, menu.php и съответното съдържание за всяка от тях.

В header.html поставяме кода, който ще се вижда в заглавната част на всички страници.

```
<img src="http://www.idg.bg/public_html/bg/iconpcw-small.gif"
alt="PC World - Bulgaria" align="left" hspace="20">
<h1 style="font:bold 2em sans-serif;color:#3300FF;text-align:center;">
Увод в PHP - 5
</h1> <br>
```

След това поставяме и навигацията - в menu.php:

```
<p align="center" style="font:bold 1em sans-serif;color:#000099">
<a href="index.php?p=main">Начало</a><br>
<a href="index.php?p=include">include ()</a><br>
<a href="index.php?p=require">require ()</a><br>
<a href="index.php?p=once">Избягване на повторенията</a><br>
<a href="index.php?p=executables">Изпълними файлове</a><br>
<a href="index.php?p=virtual">virtual ()</a><br>
</p>
```

Както забелязвате, всички линкове тук водят до един и същи файл, но с различна стойност, зададена за променливата "p" (от page). Разбира се името на променливата няма никакво значене, със същия успех можеше да е "a" или "bBbbb". Вече е време да създаден и завършващата част от интерфейса - във footer.php. В него показваме авторските права, както и датата и часа, в които е била заредена текущата страница :

```
<p align="center" style="font:bold .7em sans-serif;color:#000000">
Самоучител по PHP - част 5
<br>
Поредица на <a href="http://www.pcworld.bg">PC World -
Bulgaria</a><br>
<? echo date ("d.m.Y, H:i:s"); ?>
</p>
```

Вече е време за съдържанието на отделните страници. Нека вземем за пример съдържанието на частта "Избягване на повторенията":

```
<h2 style="font:1.5em serif;color:#cc0000;">
Избягване на повторенията
</h2>
<p style="font:.9em sans-serif;color:#000000">
Понякога, при по-сложни включвания на файлове един в друг, може да се получи неколкократно повтаряне на кода на един и същи файл.
</p>
<p style="font:.9em sans-serif;color:#000000">
Това най- често води до грешки - или в обработката или в логиката на изпълнението.....
</p>
```

Записваме този код във файл с име once.php. Остава да съберем парчетата от пъзела в обща картина. Нека съставим примерен index.php, в който да подредим цялостната картина:

```
<html>
<head>
<META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=windows-1251">
<title>Увод в PHP - 5</title>
</head>
<body>
<?php
require("header.html");
?>
<table width="95%" cellspacing="15" cellpadding="5" border="0">
<tr><td valign="top" width="120">
<?php
require("menu.php")
?>
```

```
</td><td valign="top">
<?php
include("$p.php");
?>
</td></tr></table>
<?php
require("footer.php");
?>
</body>
</html>
```

Сега вече всичко си е на мястото. Когато повикаме index.php със съответната стойност на променливата "p", ще видим съдържанието на съответния файл "$p.php". Какво ще се случи обаче, ако посетителят излезе любопитен и промени своеволно стойността на "p", така че сайтът да се опита да включи несъществуващ файл. В конкретния пример ще се появи грозно съобщение за грешка. Как да избегнем това, а и как да се справим с някои проблеми на сигурността, произтичащи от този начин на "събиране" на сайтове, ще обясним в следващия брой. Тогава ще стане дума за условните конструкции и грешките в PHP.

Условни конструкции

В предния брой създадохме малък сайт, като страниците в него се изграждаха "в движение", като вземаха съдържанието си от няколко различни файла. Името на файла със съдържанието се предаваше като част от URL, например http://www.server.com/index.php?p=once. Това означаваше, че index.php трябва да включи в себе си съдържанието на файла once.php. Тогава зададохме въпроса, какво би се получило, ако по някаква причина на index.php му се зададе да включи файл, който не съществува? Например при опит за извикване на /index.php?p=onced, ще получим съобщение за грешка :

Warning: Failed opening 'onced.php' for inclusion (include_path='.;\apache\includes;\apache\htdocs\;') in C:\apache\htdocs\pcworld\index.php on line 17

Макар и силно подценявана, възможността да се случи такава грешка е много голяма. Причините също могат да бъдат разнообразни. При по-големи сайтове става невъзможно да се проследи включването на всеки един файл. В тях се използват конфигурационни файлове, както и файлове с код за множество отделни части от показваната на монитора страница. А и при недообмисляне на рисковете, техниката на включване на множество файлове един в друг би могла да представлява сериозна дупка в сигурността на сайта, дори и на целия сървър. Кракерите знаят това и доста често умишлено предизвикват подобни грешки, чрез манипулиране на URL-то, за да проверят дали няма да открият възможност за експлойтване.

Тук трябва да посочим и друга възможна уязвимост на този начин за съставяне на сайтове. Не използвате в кода си ред като:

include ("$p");

Тук се очаква да бъде включен файлът, чието име се съдържа в променливата $p, например once.php. Ситуацията представлява огромен риск за сигурността на сайта и сървъра, защото дава възможност за извеждане на който и да е файл, за който сървърът има право на четене. Поразиите, които могат да се направят са големи, така че е необходимо да вземете мерки, за да ги предодвратите.
Можете да направите поне две неща - да определеите точен път до файловете, които да се включват и да определите точния им тип (например .txt, .html, .inc и други). Този вариант на същия ред е доста по-сигурен :

include ("/httpd/www/site/files/$p.html");

В примера се предполага, че файловете, които ще включвате се намират в директория files в Уеб директорията на сайта -

/httpd/www/site/. Този път до нея е примерен, тук трябва да проверите как изглежда той в системата, която използвате. Ползвайте абсолютни, а не относителни пътища, защото относителните са лесно манипулируеми. Те биха могли да бъдат преодоляни чрез повикване като :

http://www.server.com/index.php?p=../file.html

If - else

Изобобщо доста често при създаването на динамични файлове ще се налага да правите проверка дали е изпълнено едно или друго условие. Най-често използваната за целта конструкция е if - else. Ще я демонстрираме с пример, като включим към нашия index.php проверка дали търсеният файл наистина съществува. За да направим това ще използваме и функцията, проверяваща за наличието на файл в локалната система file_exists ()

```
<html>
<head>
<META HTTP-EQUIV="Content-Type" CONTENT="text/html;
charset=windows-1251">
<title>Увод в PHP - 5</title>
</head>
<body>
<?php
require("header.html");
?>
<table width="95%" cellspacing="15" cellpadding="5" border="0">
<tr><td valign="top" width="120">
<?php
require("menu.php")
?>
</td><td valign="top">
<?php
if (file_exists ("$p.php")) {
include("$p.php");
}
```

```
else {
echo "<center><b>Не съществува такава глава от
самоучителя!</b><br><br>Моля изберете от менюто
вляво!</center>";
}
?>
</td></tr></table>
<?php
require("footer.php");
?>
</body>
</html>
```

За да изчистим нещата, ето как изглежда синтаксиса на конструкцията:

```
if (условие) {
действие
}
else {
друго действие
}
```

Възможно е да се добавят множество алтернативни проверки с помощтта на elseif :

```
if (условие) {
действие
}
elseif {
второ действие
}
else {
друго действие
}
```

Преди да покажем друг пример, трябва да обърнем внимание на операторите за сравнение в PHP - равно, различно, по-голямо, по-малко, логическо и, логическо или и други. Следният пример сравнява две подадени числа :

```
<?
//Това е test.php
if ($a > $b) {
echo "$a е по-голямо от $b";
}
elseif ($a == $b) {
echo "Двете числа са равни";
} else {
echo "$b е по-голямо от $a";
}
?>
```

Забележете, че при проверката дали числата са равни използвахме два знака "равно" един след друг. Ако бяхме сложили само един, то интерпретаторът щеше да изравни стойностите на двете променливи. И така, ето как се правят проверките в PHP:

== - проверява за равенство;
!= - проверява за неравенство;
> - проверява дали първият параметър е по-голям от втория;
< - проверява дали първият параметър е по-малък от втория;
>= - проверява дали първият параметър е по-голям или равен от втория;
<= - проверява дали първият параметър е по-малък или равен от втория;
&& - логически "и" - проверява дали всички условия отговарят на истината;
|| - логически "или" - проверява дали поне едно от условията отговаря на истината;

Много грешки можете да допуснете в началото поради объркване

на операторите за сравнение, особено в ситуацията с равно. Нека видим два примера:

```
if ($a == $b) {
echo "Двете неща са равни";
//още действия тук
}
```

и

```
if ($a = $b) {
echo "Двете неща са равни";
//още действия тук
}
```

Ако извикаме тези две конструкции със стойности за променливите, например 2 и 5, в първия случай няма да видим изписано твърдението за равенство, защото проверката ще бъде направена според очакванията и тъй като двете числа не са равни, действието няма да бъде изпълнено.
Във втория случай, обаче, на екрана ще се изпише твърдението за равенство, както и ще се изпълнят всички действия, които сме задали. Това е така, защото един знак за равенство се възприема от PHP интерпретатора като инструкция за изравняване на стойността на единия параметър със стойността на другия, а не като оператор за сравнение. Важно е да забележите, че проверките, които се правят в PHP касаят не само числа и низове. В случая конструкцията ще върне положителен резултат от проверката, защото извършването на операцията изравняване е извършена успешно.
Можете да проверявате стойността на булеви променливи или настъпването на някакво събитие, по същите начини, по който можете да разберете дали една променлива съществува или не.
Примери:

```
if ($a) {
echo "Променливата \$a има стойност - $a"
}
```

```
else {
echo "Няма променлива $a";
}
```

Тази конструкция проверява дали съществува променлива $a и изписва стойността й. Но можехме да напишем също :

```
if (!$a) {
echo "Няма променлива $a";
}
else {
echo "Променливата \$a има стойност - $a"
}
```

Както забелязвате, възможностите за проверки са доста гъвкави. Всяка вградена функция на PHP връща стойност (най-често 1), ако бъде изпълнена, и не връща нищо (или 0) ако бъде. В примера за index.php направихме проверка за съществуването на файла, който искаме да включим. Но по една или друга причина е възможно файлът да съществува, но да не може да бъде включен. Затова можем да използваме друг вариант на проверка :

```
if (!include("$p.php")) {
echo "<center><b>Не съществува такава глава от
самоучителя!</b><br><br>Моля изберете от менюто
вляво!</center>";
}
```

Забележете, че извикването на функция в конструкцията води до нейното изпълнение. Така ако функцията include се изпълни, то условието няма да бъде вярно и предупредителния текст няма да се изпише. Но ако по някаква причина не се изпълни, то тогава условието ще е вярно и текстът ще се появи на екрана. Тук трябва да припомним особеността на конструкцията require (), която, както стана дума в предишната част, не връща стойност при изпълнението си.

Съществуват няколко вариации на изписване на тази контролна структура, но е за препоръчване да следвате основния синтаксис и

да не смесвате стиловете. Ако действието, което трябва да се изпълни при удовлетворено условие на проверката е едно, то можем да запишем

if ($name) echo "Моля, въведете потребителско име!";

С подобна конструкция можете например да проверите дали потребителят е попълнил името си в поле във форма. Съществуват и други варианти за алтернативно записване, но те се използват много рядко и сега няма да се спираме на тях.

while

While представлява най-простия начин за изграждане на цикъл в PHP. Чисто и просто той следи за това дали дадено условие отговаря на истината и докато това е така се поддържа изпълнението на определени действия:

while (условие) действие

Конструкцията е доста гъвкава и позволява свобода на използването си за различен вид условия. Най-простият вариант е създаваане на цикъл, който да се повтаря определен брой пъти :

```
$i=1;
while ($i <= 100) {
echo "Това е ред ".$i."<br>";
$i++;
}
```

В този пример дадохме на променливата $i начална стойност 1, което е по-малко от 100, следователно води до изпълнение на действията, указани в конструкцията. При всяко изпълнение на цикъла се проверява истинността на условието и когато то спре да е истина, спира и изпълнението на цикъла.
Ето този код в действие

Най-често while конструкциите се използват в по-сложни ситуации, като например обхождане на масиви :

```
while (list ($key, $val) = each ($masiv) ) echo $key - $val;
```

С масиви ще се занимаваме по-късно и тогава ще обясним подробно какво прави горния ред. Сега е важно да запомните, че проверката за истинността на условието се прави в началото на всяка итерация от цикъла. Така че е възможно той да не се изпълни нито веднъж, ако условието е невярно още при първата проверка.
Съществува конструкция, при която проверката се прави в края на итерациите. Тоест действията в цикъла задължително ще се изпълнят поне веднъж. Това е конструкцията do..while :

```
$i = 1;
do {
echo $i;
} while ($i < 100);
```

for

Най-популярният начин за създаване на цикли в PHP остава конструкцията for :

```
for (действие1, условие, действие2) {
действия
}
```

Първото действие се осъществява само веднъж, в самото начало на цикъла, а второто - при всяка негова итерация. Условието се оценява при всяка итерация, а цикълът продължава докато то е вярно. На пръв поглед изглежда сложно, но сега ще демонстрираме практическото приложение с пример:

```
for ($i = 1; $i <= 100; $i++) {
echo "Това е ред ".$i."<br>";
}
```

В първото действие присвояваме на променливата $i стойност 1. Това става само веднъж, преди същинското начало на цикъла. След това проверяваме дали стойността на $i е по-малка от 100. Ако това е вярно изпълняваме и второто действие - добавяме едно към стойността на $i. (Записът $i++ е съкратен вариант на $i=$i+1.) След това изпълняваме и действията, поставени в блока, заграден в големите скоби.

Практическия резулат от изпълнението на горните редове код ще е същия като от изпълнението на примерните while конструкции - изписване на изречението 100 пъти, като всяко ще бъде на нов ред.

Управление на цикъла

PHP предоставя допълнителни възможности за управление на цикъла, чрез break и continue. Те могат да се използват във for, foreach while, do..while или switch (ще я разгледаме след малко) конструкциите и дават по-голяма гъвкавост за реакция при определени условия. С тяхна помощ можем да определяме и други условия, влияещи на хода на циклите, които използваме. Вижте този пример :

```
for ($i = 1; $i <= 100; $i++) {
if ($i==3) continue;
if ($i==$a) break;
echo "Това е ред ".$i."<br>";
}
```

При изпълнението на този код на екрана ще се изпише изречението не 100 пъти, а толкова, колкото сме указали в променливата $a. Използвайки break можем да прекъснем изпълнението на определен цикъл при възникване на условие, различно от основното. В случая основното условие е стойността на променливата $i да е по-малка от сто. Но тук добавихме по-голяма гъвкавост, като посочихме допълнително условие, което може да прекъсне цикъла.
Друга особеност в горния код е, че в изхода му няма да има ред "Това е ред 3". Използвахме continue за да укажем, че не желаем действията в цикъла да се изпълнят, ако е вярно условието, че стойността на $i е 3. В този случай изпълнението на цикъла продължава нормално, като текущата итерация се прекратява преждевременно.
Практическото изпълнение на кода е тук. Променете стойността на $a за да промените изпълнението на цикъла.

Трябва да имате предвид, че continue и break действат считано от реда на който се намират. Нека погледнем този пример :

```
for ($i = 1; $i <= 100; $i++) {
//някакво действие тук
if ($i==3) continue;
if ($i==$a) break;
echo "Това е ред ".$i."<br>";
}
```

В този случай някаквото действие, определено в цикъла ще бъде изпълнено дори ако стойността на $i е три или е равна на стойността на $a.

switch

Това е друг вариант за управление на изпълнението на скрипта, при който имаме по-голяма възможност да разчитаме на една променлива. От стойността на тази променлива зависи кое точно действие ще бъде изпълнено. Най-удобно ще е да илюстрираме синтаксиса на конструкцията с пример :

```
switch ($i) {
case "edit":
echo "Редактиране на информацията...";
break;
case "view":
print "Преглед на информацията ";
break;
case "delete":
print "Изтриване на информацията ";
break;
}
```

В зависимост от съдържанието на променливата, парсерът ще изпълни съответните действия.

Съобщаване на грешки в PHP

Едно от големите предимства на PHP пред другите скриптови езици са съобщенията за грешка. Те са доста дружелюбни и ясни, и често казват какво точно трябва да се направи, за да се премахне

грешката. Ако вземем за сравнение писането на CGI скриптове, при появата на грешка това което виждате е простото "Error 550 - Internal Server Error". За да се ориентирате къде точно е проблема в сгрешения скрипт, трябва да отворите лога на сървъра, където ще намерите сравнително по-ясно обяснение за грешката. Но често дори и тази информация е по-обща и неясна от необходимото ви.

При PHP положението е по-различно. Тук често ще срещнете съобщения за грешка, които кзват какво точнотрябва да направите, за да премахнете проблема и на кой ред се намира некоректния код. Можем да разделим грешките в PHP на три категории - синтактични, семантични и логически.

Нека отново напомним как става изпълнението на един PHP скрипт от страна на сървъра. Когато се получи заявка за определен скрипт, сървърът го подава на PHP парсера, който първо го "парсва", иначе казано "смила" и едва след това изпълнява. Важно е да се подчертае, че парсерът обработва целия скрипт преди да започне да го изпълнява. На това ниво на предварителна обработка могат да се появят синтактичните грешки. Ако тук всичко е наред, то следва изпълнението на кода, процес при който биха могли да се появят семантичните грешки. Най-сложни и за откриване, и поправяне са логическите грешки, които не водят до преки съобщения за проблеми, но могат да доведат до некоректно изпълнение на скрипта.

Parse error

Синтактичните грешки се наричат още грешки при обработването (parse error). Нека погледнем следния пример :

```
<?
$ime=Иван";
echo $ime;
?>
```

При изпълнението на този код ще получим следното съобщение :

Parse error: parse error, unexpected '\'' in /www/idg-web.001/pcworld/php/parse_error_primer.php on line 2

Виждаме, че на втория ред от кода има синтактична грешка. Ако се вгледаме ще открием, че са пропуснати отварящите кавички на низа, задаващ стойност на променливата "ime". Понякога обаче съобщението за грешка може да ни подведе. Нека погледнем друг пример:

```
<?
if ($name=="Иван")
echo "Здравейте, Иван!";
}
else {
echo "Вие не сте Иван!";
}
?>
```

При изпълнение на този код ще получим съобщение, че съществува грешка на ред 4. Всъщност, обаче, истинският проблем е скрит във втория ред, където сме пропуснали отварящата голяма скоба.
Още един подобен пример :

```
<?
if ($name=="Иван") {
echo "Здравейте, Иван!";
}
else {
echo "Вие не сте Иван!";
?>
```

В този случай ще се сблъскаме с една от най-влудяващите ситуации, когато парсерът отчита грешка на последния ред от кода, където се намира единствено затварящата комбинация от въпросителна и средна скоба. Най-често този феномен се дължи на пропусната затваряща или отваряща скоба. В конкретния случай сме изтървали скобата, затваряща условието else.
Запомнете, че когато се получи съобщение за грешка при

парсването, скриптът озобщо няма да се изпълни. По този начин парсерът се защитава от евентуалните усложнения, които биха могли да настъпят при изпълнение на грешно написания код.

Fatal error

Фаталните грешки са една от разновидностите на семантичните. Те се появяват в процеса на изпълнение на скрипта и най-често се причиняват от извикване на несъществуващи и недефинирани функции или файлове.
Нека погледнем следния пример:

```
<?
require "config.php";
if (!$user) {
not_auth();
}
?>
```

Тук бихме могли да получим фатална грешка по някоя от следните причини - ако не съществува файлът "config.php" или ако в него не е дефинирана функцията not_auth().
Фаталните грешки се изписват на екрана в момента в който се появят в процеса на изпълнение на скрипта. В този момент изпълнението се прекратява.

Warning

Предупрежденията са друга разновидност на семантичните грешки. Те също се появяват в процеса на изпълнение, но той продължава и след тях. Предизвикват се от грешки, които не са фатални за цялостния ефект, макар че най - често водят до грешно изпълнение на кода. Нека вземем горния пример, но ще сменим функцията за включване на файл от "require" на "include".

```
<?
include "config.php";
if (!$user) {
header ("location: login.php");
}
```

...
?>

В този пример проверяваме дали на променливата $user е присвоена някаква стойност или не. Ако не, тогава препращаме потребителя към форма за идентификация, ако да - продължаваме с изпълнението на скрипта. Нека сега приемем, че файлът config.php не съществува. Понеже сме използвали "include", ще получим предупреждение за невъзможността да се включи указания файл, но изпълнението на скрипта ще продължи.
Фаталните грешки и предупрежденията са съобщения, които трябва да възприемате сериозно, ако искате кода, който пишете наистина да изпълнява предназначението си.

Забележки

Забележките са този вид съобщения, с чиято помощ можете да се преборите с логическите грешки в скриптовете си. Под логически грешки се разбират неточности в кода, които де факто не възпрепятстват изпълнението му, но водят до грешни резултати. Те могат да са резултат от грешка на изписването или на недобре съставен алгоритъм. По подразбиране, настройките на PHP за показване на грешки са така направени, че забележките не се изписват на екрана. Това е напълно логично, защото в противен случай не бихме могли да използваме конструкции като тези, показани в горния пример. Ако го изпълните на сървър с PHP, настроено да показва всичкии грешки, ще получите дълго съобщение за грешка, което твърди, че се опитвате да използвате неинициализирана променлива. Както вече стана дума, променливите в PHP не е необходимо да бъдат обявявани преди да се използват. Тяхното инициализиране става автоматично, в момента в който на променливата се присвоява някаква стойност. В случая ние разчитаме на променливата $user за да проверим дали посетителят вече се е идентифицирал или не. Ако обаче показването на забележки е включено, тогава ще видим изписаното на екрана съобщение, че се опитваме да използваме недефинираната променлива в случаите, когато потребителят все

още не се е идентифицирал.
Използването на забележки е особено полезно в стадиите на тестване на сайта, когато реузлтатите не са такива, каквито очакваме. Например напълно възможно е да допуснем грешка в името на някоя променлива при писането на кода. Разбира се това ще причини множество проблеми и няма да ни доведе до желания резултат. Използвайки забележките, можем да открем подобни грешки.

КНИГА 3
PHP2

Управление на съобщенията за грешка

В PHP имате пълната власт да управлявате извеждането на грешки на екрана на посетителите си. Имайте предвид, че макар и полезни за вас, тези съобщения са последното нещо, което посетителите ви желаят да видят. Съществуват два начина да се справите с проблема. Първия е да пишете кода си по такъв начин, че да не се налага парсера да издава служебни съобщения в случай на непредвидени гешки. Но е невъзможно да предвидите всички възможности, които биха могли да доведат до срив. Затова е добра практика да подтискате извеждането на съобщенията за грешка щом веднъж пуснете сайта за свободно посещаване от хората. Можете да управлявате извеждането на съобщенията чрез директивата error_reporting.

число error_reporting ([число ниво])

Чрез тази директива се задава ниво на съобщаването за открити грешки чрез число или константа. Връща старата стойност на нивото. Препоръчва се използването на константи, а не на числа. Директивата може да се използва както в конфигурационния файл на PHP, като в този случай ще се отнася до всички скриптове, така и в отделните скриптове по отделно.
Така например ако жеаем да подтиснем изписването на каквито и да било съобщения за грешка, можем да поставяме в самото начало на скриптовете следния ред :

error_reporting(0);

Ето как можете да зададете предочитаните от вас нива за съобщаване на грешките :

// Съобщаване на всички грешки
error_reporting (E_ALL);

```
//Положението по подразбиране, при което се съобщават всички грешки, освен забележките
error_reporting (E_ALL ^ E_NOTICE);

// Ако искате да виждате и забележките, поставете това в началото на скрипта
error_reporting (E_ERROR | E_WARNING | E_PARSE | E_NOTICE);
```

Прихващане на грешките

Правилното обработване на грешките е особено важно за преставянето на един сайт. За съжаление то се подценява масово в България и често сме свидетели как дори най-големите и посещавани нашенски страници посрещат посетителите с не особено красивите и приятни съобщения за таймаутване на скриптове или за невъзможност за връзка с базата данни. Трябва да имате предвид, че служебните съобщения на PHP могат да ударят доста сериозно доверието на потребителите към сайта. В крайна сметка, ако със собствения си сайт не можете да се справите, как ще ги убедите, че ще можете да сте полезни и на тях?
Има най-различни начини да осигурите свое, интелигентно решение за обработка на грешките. Повечето функции в PHP връщат резултат, ако се изпълнят или "0", ако не що им попречи. Както вече демонстрирахме, можете да се възползвате от това и да се подсигурите срещу непредвидените обстоятелства. В такива случаи можете смело да използвате и символа "@", който, поставен пред името на функцията подтиска извеждането на съобщения за грешка. Ето как бихме могли да обработим горния пример, за да се подсигурим срещу невъзможноост да бъде зареден нужния файл "config.php".

```
<?
if (!@include "config.php") {
```

```
echo "<b>Сайтът не може да обработи завката Ви в момента!</b><br><br> Моля опитайте по-късно! <br>Извиняваме се за причиненото неудобство!";
exit;
}
if (!$user) {
header ("location: login.php");
}
?>
```

Масиви

Масивите са изключително полезни, когато обработвате повече и по-разноообразна информация. Всъщност приложението им може да е доста разнообразно, но те са едно от основните неща, на които ще разчитате, когато работите с бази данни. Тогава всички данни от един ред от таблицата в базата ще бъдат получавани в масив, който ще трябва да обработим,за да достигнем точно до тази информация, която ни трябва. Използването на масиви дава възможност да направите кода си по - ясен, стегнат и четлив.

Най-просто казано, масивите са "контейнери" от променливи - поредица от елементи, всеки от които има свой маркер (наименование), чрез който би могъл да бъде извикан.

Създаване на масиви

Нека си представим, че работим върху приложение, в което се използват имената на градове в България и техните пощенски кодове. Ето част от списъка, с който разполагаме:

Айтос - 8500
Асеновград - 4230
Балчик - 9600
Ботевград - 2140
Бургас - 8000

Варна - 9000
В. Търново - 5000

След малко ще направим масив, в който се съдържа целия този списък във вид на низово индексиран масив. Но нека започнем от нещо по-просто. Ще съставим масив с имената на всички градове, представени в списъка ни. В структурата на масива, всяко име на град ще получи автоматично идентификатор - число от нула нагоре. Преди това ще уточним, че за разлика от повечето други програмни езици, PHP ни позволява да слагаме в един и същи масив различни типове данни. Освен това тук имаме избор дали да инициализираме изрично масива или направо да започнем да добавяме данни към него. Ето как бихме могли да съставим простия примерен масив с имената на градовете :

```
<?
$gradove[]='Айтос';
$gradove[]='Асеновград';
$gradove[]='Балчик';
$gradove[]='Ботевград';
$gradove[]='Бургас';
$gradove[]='Варна';
$gradove[]='В. Търново';
?>
```

В реузлтат на изпълнението на тези редове, в рамките на скрипта ще имаме готов масив ***$gradove[]***, в който ще са изброени имената. Разликата в изписването на масивите и променливите е в квадратните скоби, в които се указва с коя точно стойност желаем да работим. За да обясним това ще дадем пример с купчина папки (масив от папки), във всяка от които се съхраняя някаква информация. Самите папки са надписани, така че да можем да намерим точно тази информация, която ни интересува. И ако желаем да боравим с нея - да я прочетем, да добавим нова информация или

изобщо да я премахнем, то ще трябва да открием точно папката, с която искаме да работим. Можем да кажем, че надписа (името) на папката е ключа (key), а съдържащата се в нея информация - стойността (value) на една единица от целия масив от папки.

По същия начин стоят нещата и с масивите в PHP. Те са съставени от отделни единици информация, всяка от която има своя идентификатор (ключ). Точно този идентификатор записваме в скобите, когато желаем да добавим, променим, прочетем или изтрием нещо. Ако пропуснем идентификтора и не запишем нищо в скобите, то PHP интерпретаторът ще трябва сам да прецени какво да направи.
В горния пример добавихме имената на градовете в масива, без да определяме ключ за всяко отделно име. Затова интерпретаторът е добавил автоматично номерация, започваща от нула. И ако извикате стойността на $gradove[0] ще установите, че това е "Айтос". По същия начин Варна може да се извика като $gradove[5].

Можем да използваме и друга конструкция за да получим същия масив:

```
<?
$gradove = array ("Айтос", "Асеновград", "Балчик",
"Ботевград", "Бургас", "Варна", "В. Търново");
echo "$gradove[4]";
?>
```

Изпълнението на горния код ще изпише на екрана "Бургас". Бихме могли да променим началото на индексацията на масива, като укажем от къде да започне тя:

```
<?
$gradove = array (1 => "Айтос", "Асеновград", "Балчик",
"Ботевград", "Бургас", "Варна", "В. Търново");
echo "$gradove[4]";
```

?>

Тук вече резултатът ще бъде изписването на "Ботевград", защото индексирането в масива е започнало от едно, а не от нула. Знаейки ключа към различните елементи от информацията, можем и да я обработваме. Ето как ще сменим името на Ботевград с Благоевград :

<?
$gradove[4]='Благоевград';
?>

Обхождане на масиви

Едно от най-ценните свойства на масивите е възможността да бъдат "обхождани" - т.е.можете да получите достъп до всеки елемент от масива последователно в рамките на цикъл. Единият вариант да направите това е чрез For цикъл. За да стане това, обаче, трябва да знаете от къде започва индексирането на масива и колко елемента има в него. Второто можете да научите с помощта на функцията ***count ()*** :

<?
$kolko=count ($gradove);
echo $kolko;
?>

Ако изпълним тези редове, ще видим, че в масива $gradove има общо 7 елемента. Знаейки, че в случая сме започнали индексирането от нула, можем да направим следния цикъл:

<?

//създаваме масива
$gradove = array ("Айтос", "Асеновград", "Балчик", "Ботевград", "Бургас", "Варна", "В. Търново");

//започваме падащото меню
*echo "Моля, изберете град :
<select name="grad">";*

```
//преброяваме масива
$kolko=count ($gradove);

//създаваме цикъла за обхождане на масива и опциите на
падащото меню
for ($i=0; $i<$kolko; $i++) {
echo "<option value=$i>$gradove[$i]</option>";
}

//завършваме падащото меню
echo "</select>";

?>
```

В най-общи линии горният код ще създаде падащо меню, от което потребителят ще може да избере съответния град. Пропуснах кода за създаването на формата, за да запазя примера ясен.
Често обаче не можем да сме сигурни как точно е индексиран масива, дали изобщо ключовете представляват числа и ако да - дали са поредни. Затова използването на for() не е най-доброто решение. Съществува възможност за обхождане на масив, за който нямаме предварителна информация с използването на функциите ***each()*** и ***list(),*** в комбинация с конструкцията ***while ()***.
С тази комбинация можете да обиколите всички елементи на масива и да получите информация за данните, съхранявани в него, както и за ключовете на отделните елементи.

```
<?

//създаваме масива
$gradove = array ("Айтос", "Асеновград", "Балчик",
"Ботевград", "Бургас", "Варна", "В. Търново");
```

```
//започваме падащото меню
echo "Моля, изберете град : <br><select name="grad">";

while (list ($key, $value) = each ($gradove)) {
echo "<option value=$key >$value</option>";
}

//завършваме падащото меню
echo "</select>";

?>
```

Не е трудно да запомните тази конструкция, която ще ви трябва доста често. Преведено на български, тя казва "Прелисти (list) ключовете ($key) и съдържанието ($value) на всеки (each) елемент на масива. Цикълът ще се изпълнява до обхождането на всички елементи на масива. Когато това стане, условието вече няма е вярно (т.е присвояване няма да може да се извърши).
Функцията each () се грижи за изместването на активния елемент, конструкцията задава на променливата $key името на елемента, а на променливата $value - стойността му.

PHP предоставя доста голяма свобода на движението, позволявайки да се движите напред (да прескачате елементи) или назад (да се връщате към вече обработени). За целта се използват съответно функциите next () и prev (), а за да навигирате свободно из масива можете да използвате array_walk ().

Индексиране с низове

До сега боравихме с масиви, чиито елементи бяха обозначени с числа. Но PHP позвоялва да използваме за тази цел и низове. На основата на $gradove[] ще направим друг масив, в който ще се съдържат пощенските кодове на

съответните градове. Кодовете ще представляват същинската информация, а имената на градовете ще са ключовете. За тази цел ще използваме същата "стрелка" (=>), която ползвахме преди малко. Но в този случай преди всяка стойност ще укажем изрично ключа, който искаме да й съответства. Ето как би могло да стане това :

```
<?
$kodove = array ("Айтос" => "8500", "Асеновград" => "4230",
"Балчик" => "9600", "Ботевград" => "2140", "Бургас" => "8000",
"Варна" => "9000", "В. Търново" => "5000");
?>
```

За да вкараме новосъздадения масив в действие, ще направим пример за файла, който би могъл да посреща заявката от предходния пример. Тоест, когато посетителят избере град от падащото меню, което вече създадохме по-горе, скриптът, който ще съставим сега ще изпише името града и неговия пощенски код.

```
<?
//създаваме масивите
$gradove = array ("Айтос", "Асеновград", "Балчик",
"Ботевград", "Бургас", "Варна", "В. Търново");
$kodove = array ("Айтос" => "8500", "Асеновград" => "4230",
"Балчик" => "9600", "Ботевград" => "2140", "Бургас" => "8000",
"Варна" => "9000", "В. Търново" => "5000");

//номерът на града се подава от предходния пример в
променливата $grad
//променливата $gr ще съдържа името на града, а $c - кода
му
$gr=$gradove[$grad];
$c=$kodove[gr];

//изписваме информацията
echo "Град $gr е с пощенски код $c.";
```

?>

Работа с файлове и директории

PHP предлага множество удобства при работа с файлове, без значение дали те се намират на локалния сървър или на отдалечен домейн. Има много случаи в които ще ви се наложи да обработвате съдържанието на различни файлове, например при използването на по-прости броячи или обработка на информация от най-различен вид. Както повечето други възможности на езика, работата с файлове в PHP е облекчено и бързо.
Тук са представени основните функции и конструкции за извършване на най-необходимите операции с файловата система на сървъра, както и за работа с отдалечени файлове. Но PHP предлага богати възможности, затова когато се почувствате комфортно с основните неща, разгледайте официалната PHP докуметация.

Отваряне на файл

Преди да направим каквото и да било със съдържанието на един файл, трябва да го отворим. Можем да направим това с функцията fopen().

```
<?
$filename = '/www/mysite/dir/file.txt';
$fp = fopen($filename, "r");
?>
```

Както виждаме, функцията fopen() приема два параметъра - името на файла и "състоянието" в което ще се отвори той. Можем да зададем пълния или относителния път до файла, ако се намира на локалната система или неговото URL, ако е на отдалечен сървър в Интернет. Използвайки параметъра "r" за да укажем състоянието на файла при отваряне показахме, че искаме да го отворим само за прочитане. Ето какви са вариантите за този параметър :

'r' - отваря файла само за четене, като поставя показалеца в началото му;
'r+' - отваря файла за четене и записване, като поставя показалеца в началото му;
'w' - отваря файла само за писане, като поставя показалеца в началото му. Ако такъв файл съществува, цялото му съдържание ще бъде унищожено, ако не съществува - парсерът ще опита да го създаде.
'w+' - отваря файла за четене и писане, като поставя показалеца в началото му. Ако такъв файл съществува, цялото му съдържание ще бъде унищожено, ако не съществува - парсерът ще опита да го създаде.
'a' - отваря файла само за писане, като поставя показалеца в края на файла. Ако не съществува такъв файл ще се опита да го създаде.
'a+' - отваря файла за четене и писане, като поставя показалеца в края на файла. Ако не съществува такъв файл ще се опита да го създаде.

Тук трябва да въведем някои понятия, които ще използваме за да обясним как действат файловите функции. При изпълнението си, fopen() връща така наречения файлов идентификатор (в случая $fp), който след това използваме винаги, когато боравим с отворения файл. Освен това, обаче, имаме и показалец, от който зависи къде точно в отворения файл ще се направят промените или какво точно ще прочетем. Например ако отворим файла в който искаме да добавим няколко реда текст чрез параметър "a+", то добавянето ще става в края на съществуващата информация. Но ако използваме "w+', то добавянето ще е в самото начало, преди данните, които вече съществуват.

Прочитане на файла

Нека сега направим нещо повече с нашия примерен файл.

```
<?
$filename = '/www/mysite/dir/file.txt';
```

```
$fp = fopen($filename, "r");
$string = fread($fp,filesize($filename));
fclose($fp);
echo $string;
?>
```

Преди да обясним горните редове, ще подчертаем, че отворените с функцията fopen () файлове трябва да се затварят! За целта се използва функцията fclose(), която приема като параметър идентификатора на файла, който искаме да затворим.
Функцията filesize() връща големината на файла в байтове. Ако по някаква причина не успее да се изпълни, връща FALSE. Файлът, чиято големина ни интересува, трябва да се намира във файловата система, защото функцията не работи с URL.
След като вече знаем колко е голям файлът, можем да го прочетем целия с функцията fread(). Тя приема два аргумента - файловия идентификатор, върнат от fopen (), и големината на парчето информация, което искаме да прочетем. В случая искаме да прочетем целия файл, затова като втори аргумент подаваме изхода от финкцията filesize(). Често ще ни се налага да четем различни по-малки фрагменти от файловете.
В крайна сметка при изпълнението на горния код, на променливата $string ще бъде присвоено като стойност съдържанието на файла, което след това ще бъде изведено на екрана. Нека си представим, че във file.txt има това :

ред1
ред2
ред3
ред4
ред5

След прочитането му, променливата $string ще съдържа

"ред1\nред2\nред3\nред4\nред5". С "\n" обозначаваме нов ред, което означава, че когато съдържанието се изведе на екрана, то ще съответства напълно на съдържанието на прочетения файл, със запазени нови редове в текста.

Писане във файла

PHP ни предоставя няколко възможности за добавяне на съдържание във файл, в зависимост от конкретната необходимост. Нека първо създаден файл, който да съдържа изречението "Това е нов файл".

```
<?
$filename = "/www/mysite/test.txt";

//отваряме файла само за четене, ако до сега в него е имало някаква информация, тя ще бъде изтрита
$fp = fopen($filename,"w');

//записваме изречението във файла
fwrite($fp,"Това е нов файл");
fclose($fp);
?>
```

Функцията fwrite () записва определени данни във файл. Тя приема три входни аргумента, като единият не е задължителен.

fwrite (файлов идентификатор, низ [, дължина])

Файловият идентификатор се получава чрез функцията fopen (), низът е съдържанието, което искаме да запишем във файла, а дълллжината указва какво количество байтове искаме от това съдържание искаме да запишем. Ако пропуснем третия аргумент, ще бъдат записани всички данни. Функцията връща количеството записана информация, а ако по някаква причина не успее да се изпълни, връща FALSE.

След като вече имаме създаден файл, можем да продължим да го пълним с информация. За целта ще използваме функцията fputs (), за да зашишем данните, започвайки от текущото положение на показалеца. Функцията е идентична на fwrite () и приема същите три входни параметъра.

fputs (файлов идентификатор, низ [, дължина])

Сега ще добавим още няколко реда текст в създадения преди малко test.txt.

```
<?
$filename = "/www/mysite/test.txt";

//отваряме файла за добавяне, като показалецът е в края му
$fp = fopen($filename, "a");

//добавяме два нови реда и затваряме файла
$string = "\nТова е втори ред на файла\nТова е трети ред на файла";
$write = fputs($fp, $string);
close($fp);
?>
```

Примерен брояч

Нека сега направим малък брояч, който ще показва на посетителя кое по ред посещение регистира страницата. За целта ще използваме текстов файл (broyach.txt), в който ще съхраняваме текущата стойност на брояча. Важно условие е сървът да има права за писане в директорията, където ще се намира файла, за да може да го създаде и обновява.

```
<?
//Дефинираме колко цифрен да бъде брояча (пример 00001 или 01)
```

```
$digits = 5;

//Указваме в кой файл да се съхранява стойността (добре би било този файл да се намира в директория, извън достъпната през Уеб, ако, разбира се, е възможно)
$filelocation="/user/files/logs/broyach.txt";

//проверяваме дали такъв файл съществува и ако не - създаваме нов, в който поставяме числото 1.
if (!file_exists($filelocation)) {
$newfile = fopen($filelocation,"w+");
$content=1;
fwrite($newfile, $content);
fclose($newfile);
}

//прочитаме числото във файла
$newfile = fopen($filelocation,"r");
$content = fread($newfile, filesize($filelocation));
fclose($newfile);

//увеличаваме числото с 1 и го записваме
$newfile = fopen($filelocation,"w+");
$content++;
fwrite($newfile, $content);
fclose($newfile);

//изписваме текущата стойност на брояча
echo "".sprintf ("%0"."$digits"."d",$content)."";
?>
```

Функцията sprintf () се използва за извеждане на екрана на форматирани низове. Повече за нея ще откриете в PHP документацията на адрес http://www.php.net/manual/en/function.sprintf.php.

Да изпратим файла

Една от най-често употребяваните функции за работа с файлове в PHP е **fpassthru ()**. Тя прочита съдържанието на отоврен файл и го изпраща направо към изхода на приложението (например към браузъра). При грешка връща FALSE. Приема само един аргумент - файловият идентификатор, върнат от fopen().
Ще демонстрираме действието й, като променим брояча от горния пример. Сега той само ще записва, но няма да пооказва на посетителите броя на посещенията. А ще бъде включен в страницата чрез изображение.

```
<?
$img_file="php_logo.gif";
$filelocation="broyach.txt";

if (!file_exists($filelocation)) {
$newfile = fopen($filelocation,"w+");
$content=1;
fwrite($newfile, $content);
fclose($newfile);
}

$newfile = fopen($filelocation,"r");
$content = fread($newfile, filesize($filelocation));
fclose($newfile);

$newfile = fopen($filelocation,"w+");
$content++;
fwrite($newfile, $content);
fclose($newfile);

$fd=fopen($img_file,'r');
fpassthru($fd);
?>
```

Записваме този файл като counter.php. Сега можем да го включваме в която пожелаем уеб страница, чрез "img" тага, защото файлът всъщност представлява изображение.

Създаване и изтриване на директориите

Ето накратко и няколко функции за работа с директории. За да създадем нова, използваме mkdir () :

mkdir ("път", "състояние")

Функцията приема два входни параметъра - името на директорията, която искаме да създадем (с относителния или пълен път до нея) и състоянието (разрешенията) за достъп и писане на файлове в нея. Например, ако искаме да създадем директория "access" във вече съществуващата "/home/mydir/logs", можем да напишем следното :

mkdir ("/home/mydir/logs/access", 0700)

Съдържанието на така създадената директория ще бъде достъпно за четене и писане от скриптовете, действащи от името на потребителя, който я е създал. Още за разрешенията ще стане дума по-късно, в последната част на този самуочител. Тъй като използвахме осмична бройна система за указване на правата, ги записахме с 0 в началото.

Премахването на директорията става с функцията rmdir (), която приема само един аргумент - името на директорията, кояти искаме да изтрием.

rmdir ("/home/mydir/logs/access");

Този ред ще премахне създадената по-горе директория, но само ако е празна. И разбира се, ако сървърът има права за писане в нея. Връща TRUE ако успее и FALSE ако се провали.

-php-

MySQL и PHP

Невъзможно е да си представим модерен и високофункционален Уеб сайт, в основата на който не стои база данни. Множество от функциите, които правят страниците привлекателни не биха могли да съществуват, услуги като онлайн банкиране, пазаруване и дори просто организиране на информацията, са немислими без някакъв вид система за бази данни. Няма да навлизаме в технологични и терминологични подробности, а ще покажем как да използвате напрактика една от най-популярните и разпространени в Интернет системи за управление на бази данни - MySQL. Повече за самата система можете да откриете в сайта на фирмата - производителка http://www.mysql.com.
Основната версия на системата е безплатна за използване, без значение с комерсиални или идеални цели, като това е една от основните причини за широката ѝ популярност. Това не означава, че възможностите, които предлага са недостатъчни. Системата може да изпълнява заявките на неограничен брой потребители, издържа до 50 милиона записа, предлага лесна за използване система за управление на правата на различните потребители, отлична скорост на обработка на заявките.

MySQL

MySQL е система за управление на релационна база данни (БД), която използва Structured Query Language (SQL) - най-популярният език за добавяне, прочитане и обработка на информация в базите данни днес. Системата е с отворен код и използването ѝ се подчинява на лицензa GPL. Първата версия на MySQL се появи през януари 1998 година. Може да се използва с голяма група програмни езици - C, C++, Eiffel, Java, Perl, PHP, Python и Tcl и има версии за Linux, UNIX и Windows.
Системата за управление на релационна база данни

(RDBMS) позволява създаването, администрирането и работата с релационни бази данни. Тези БД представляват съвкупности от информационни единици, организирани формално в таблици. Достъпът и промяната на данните се извършва без да е необходимо реорганизирането на таблиците или каквото и да било друго в БД. Едно от най-важните предимства на релационните БД е лекотата с която се създават, четат и изтриват записите, както и лесната разширяемост.
Спецовете обясняват, че релационната БД се състои от таблици, в които данните са подредени по колони(категории). Всеки ред съдържа уникално смислено съчетание от данни (стойности на колоните). Така таблицата всъщност свързва категориите, тя е "релация" между тях. Данните от различни таблици също могат да са свързани -- така имаме отново релация, но този път между таблиците. По този начин таблиците и връзките между тях спояват данните в едно логическо цяло, и именно то е (релационна) базата данни..
Ако последните няколко изречения не са ви ясни, няма проблеми. За да не ви се смеят "знаещите", трябва само да запомните, че MySQL не е база данни, а система за управление на база данни. И то не на каква да е БД, а на релационна. Толкова с теорията.
За да стане по-ясен текстът от тук нататък, ще кажем, че базата данни се състои от таблици, всяка от които има колони (указващи каква информация трябва да се съхрани там) и редове (представляващи записите). Всяко парче от данните се поставя в полагащото му се сечение между редовете и колоните - т.е. в съответното поле.

Таблиците

За да покажем как се използва MySQL ще направим една примерна и доста упростена директория от линкове. Нещо като dir.bg, където хората отиват, избират си категория и гледат какви линкове е сложил администраторът вътре. Нашият

вариант на директория ще е доста по-простичък, но нищо не пречи след време да се поупражнявате върху него и да направите супер-съвършенна система за класификация на адреси.

Тук няма да обясняваме как се създава потребител за MySQL сървъра, как се определят правата му, как се създават бази данни и т.н. Приемаме, че сте си намерили хостинг, където администраторът се е погрижил да ви даде потребителско име, парола, база данни, както и информация за адреса и порта на MySQL сървъра (обикновено 6667).
За онагледяване на примерите ще използваме пакета phpMyAdmin (http://www.phpmyadmin.net/), който предоставя Уеб базиран интерфейс към системата. Ще покажем и основните конструкции за изпращане на запитване към БД сървъра.

За да създадем директорията имаме нужда от две таблици - в едната ще съхраняваме категориите, в другата - информацията за сайтовете. Логическата връзка между двете таблици ще се осъществява посредством полета, съдържащи вътрешен идентификатор на категориите, както ще покажем след малко. Като допълнителна възможност - всяка категория ще има възможност да съдържа неопределен брой подкатегории.
Таблицата с категориите ще съдържа три колони - идентификатор на категорията (catid), който ще е число, име на категорията (catname), което ще е низ до 100 знака, без значение дали съдържа букви, цифри и т.н. и идентификатор на родителската категория (catparent).
Основният ключ (primary key) на таблицата ще е полето catid, а имената на категориите няма да могат да се повтарят.
Създаваме таблицата така:

```
CREATE TABLE categories (
catid int(20) NOT NULL auto_increment,
```

```
catname varchar(100) NOT NULL,
catparent int(20),
PRIMARY KEY (catid),
UNIQUE CatName (catname)
);
```

Таблицата със сайтовете ще има общо седем колони - идентификатор на линка (linkid), идентификатор на категорията, в която е класифициран сайта (catid), който е логическата връзка между двете таблици, URL (url) - низ до 255 знака, име на сайта (linkname) - низ до 100 знака, име на добавилия сайта в директорията (izpratil), неговия e-mail адрес (izpratilemail) и дата на добавяне. Основният ключ на таблицата ще е идентификатора на сайта, а за да не стават повторения, URL полето ще трябва да е уникално. Тази таблица създаваме така:

```
CREATE TABLE links (
linkid int(20) NOT NULL auto_increment,
catid int(20) DEFAULT '0' NOT NULL,
url varchar(255) NOT NULL,
linkname varchar(100) NOT NULL,
izpratil varchar(100) NOT NULL,
izpratilemail varchar(100) NOT NULL,
data int(20) DEFAULT '0' NOT NULL,
PRIMARY KEY (linkid),
UNIQUE Url (url)
);
```

Създаваме таблиците с SQL инструкцията **CREATE TABLE име (дефиниране на таблицата)**. Синтаксисът може да е доста по-сложен, за подробности погледнете документацията на адрес http://www.mysql.com/doc/en/CREATE_TABLE.html. В

скобите указваме структурата на бъдещата таблица, като имаме предвид типа на данните, които искаме да съхраняваме в нея. Отделните иснтрукции се разделят със запетайки, но след последната запетайка не се поставя! Няма значение дали оставяте интервал след запетайката или не.

За създаване на таблици можете да използвате phpMyAdmin, за който вече стана дума, но така или иначе трябва да разберете смисъла на основните типове данни, за да постигнете някакъв резултат. Ако се опитате да създадете таблица, съдържаща полета с неправилно описани изисквания за типа информация, ще получите съобщения за грешка. Ако все пак успеете, то проблемите ще се появят на следващ етап, когато таблицата няма да се държи по начина, по който очаквате.

Базата ни данни вече притежава двете таблици, необходими за директорията :

Типове данни в MySQL

MySQL предлага поддръжка на различни типове данни, но най-вероятно поне на първо време ще са ви необходими четири или пет от тях. Ето описанията на основните, с "x" в скобите е обозначено число, указващо дължината на записа, а квадратни скоби са показани параметри, които не са задължителни.

CHAR (x) - низове с точно определена дължина. Възможната стойност за x е от 1 до 255. Пример :user_id CHAR(10) - очаква се името на модела да е точно 10 символа.

VARCHAR (x) - низове с максимална дължина x. Отново възможните стойности са между 1 и 255. Пример : user_id VARCHAR(10).

INT (x) [Unsigned] - използва се за указване на цели числа между -2147483648 и 2147483647. Ако се използва Unsigned, тогава валидните числа са между 0 и 4294967295. Пример : user_phone INT.

FLOAT [(M,D)] - указва малко дробно число, като M регулира общия брой цифри, от които може да се състои числото, а D ограничава колко от тях може да са зад десетичната запетая. Ако числото е с повече цифри след запетаята, се прави закръгляне. Пример : suma FLOAT (4,2) - означава, че ако поставите в това поле числа като 2,31 или 22,56, а също и 1, 34 или 1,2, те ще бъдат съхранени правилно. Но ако се опитате да сложите нещо като 32,567, на практика в полето ще се запише 32,57. Ако искате да сте сигурни, че няма да се получи закръгляне, можете да използвате типа DECIMAL.

DATE - използва се за съхранение на дати, както говори и името. Форматът по подразбиране е "ГГГГ-ММ-ДД", като можете да поставяте стойности от '0000-00-00' до '9999-12-31'. MySQL поддържа множество команди за работа с дати, които няма да обсъждаме тук. Пример : born DATE.

TEXT - служи за съхраняване на по-големи низове - от 255 до 65535 символа. При търсене в низовете, съхранявани в такива полета, MySQL няма да прави разлика между малкии големи букви.

BLOB - също като TEXT, с разликата, че търсенето тук взима предвид малките и големи букви.

SET - по-сложна дефиниция, която може би никога няма да използвате, но е добре да знаете. Може да съдържа до 64 предварително дефинирани стойности, от които могат да се избират една или повече. Например : transport SET ("vlak",

"avtobus") NOT NULL - от тук имам четири възможности за избор - нито едно от двете, само едното от тях (влак ИЛИ автобус) или и двете (влак И автобус). Както забелязвате, възможните стойности се задават в момента на създаването на таблицата.

ENUM - също като SET, но е позволено избирането само на една стойност. Ако се върнем на горния пример, тук не можем да изберем варианта влак И автобус.

Още за създаването на таблиците

Както забелязахте, когато създавахме двете таблици за директорията - "categories" и "links", дадохме и други инструкции за структурите им, освен типовете данни на отделните полета. Нека погледнем следните инструкциии :

linkid int(20) NOT NULL auto_increment,
PRIMARY KEY (linkid),
UNIQUE Url (url)

NOT NULL означава, че полето не може да остане празно в нито един от записите в таблицата. Попълването му е задължително, в противен случай ще се появи информация за грешка.

AUTO_INCREMENT означава, че стойността на това поле автоматично ще бъде добавена като към стойността на полето от предходния запис се добави 1. Обикновено при този тип полета посочвате NULL като стойност при добавянето на записа. Използва се често за създаване на уникален идентификатор за отделните записи и може да ви свърши много полезна работа.
PRIMARY KEY (първичен ключ на таблицата) е онова поле, което системата използва за да разграничава различните

записи. Не може да има два различни реда с една и съюа тойност на полето, определено за първичен ключ. Всяка таблица има свой PRIMARY KEY.
UNIQUE също ще ви гарантира уникалността на информацията. Например в таблицата с линковете искаме да сме сигурни, че в директорията ни няма да има отправки към един и същ сайт два пъти.

Попълване на таблиците

Продължаваме изграждането на примерната директория с добавяне на записите в таблицата за категориите.
Съзнателно пропускам подробностите за работа с MySQL от командния ред и в следващите редове ще продължим да използваме phpMyAdmin за да добавяме, редактираме или премахваме записи. Причините за това са две - показаните в шела на Linux таблици от база данни не са най-конфортния начин да разберете какво се случва в базата данни ако тепърва започвате да се занимавате с това. Освен това има доста потребители, ползващи MySQL под Windows, както и такива, чиито хостинг не позволява SSH достъп до сървъра. Затова усложняването е излишно.

Добавяме категориите

Категориите, в които ще разпределяме линковете в нашата директория, ще се съхраняват в таблицата categories, която вече създадохме.
Тя има следните три полета :
- *catid* - число, съставено от до 20 цифри, което ще представлява уникален идентификатор на категорията. Това число ще бъде добавяно автоматично и ще е с една единица по - голямо от числото в предишния запис.
- *catname* - низ, който може да е съставен от различни символи с максимална дължина от 100 символа. Тук ще

съхраняваме името на категорията.
- *catparent* - число, съставено от до 20 цифри, което ще указва идентификационния номер (catid) на директорията, която е родителска на текущата.

Ще създадем категория "Музика", в която ще има три под-категории : "Изпълнители", "Групи", "Компании". В подкатегорията "Изпълнители" ще добавим още две подкатегории - "Жени" и "Мъже", а "Групи" ще разделим на "Български" и "Чужди".

В началото ще създадем трите основни категории. За целта ще използваме командата INSERT с чиято помощ ще вкараме три записа в празната все още таблица.

INSERT INTO таблица VALUES ('първа стойност', 'втора стойност', ...);

Този ред ще създаде нов запис в указаната таблица, попълвайки по реда полетата от ляво надясно - в първото поле ще влезе "първа стойност" и така нататък. Стойностите, които искаме да добавим изписваме винаги в единични кавички. Изключение са само числата, които поставяме в поле с типа на данни INT.

Има два алтернативни записа на показания по-горе код:

INSERT INTO таблица (поле1, поле2) VALUES ('стойност1', 'стойност2');

или

INSERT INTO таблица SET поле1=('стойност1'), поле2=('стойност2');

Както се вижда, двата варианта ви позволяват да създадете запис, без да попълвате всички полета в него. Това обаче може да се направи и с първия вариант. Нека демонстрираме на практика тези редове, като ги използваме за да създадем

категориите.

```
INSERT INTO categories VALUES (NULL, 'Музика', 0);
```

Както можете да видите на долното изображение, вече имаме създадена категория "Музика", която е с идентификационен номер (catid) 1 и родителска категория 0 - тоест, тази категория е от най-високо ниво. Забележете, че използвахме NULL за да укажем, че не искаме да укажем стойност за "catid". Съответно около числото 0, което поставихме в поле, дефинирано като INT, не сложихме единичните кавички.

Сега ще добавим трите подкатегории, като вече в "catparent" ще поставяме числото 1, което е идентификор на категорията "Музика". Ще използваме трите различни начина за употреба на INSERT, създавайки трите подкатегории:

```
INSERT INTO categories VALUES (NULL, 'Изпълнители', 1);
INSERT INTO categories (catname, catparent) VALUES ('Групи', 1);
INSERT INTO categories SET catname=('Компании'), catparent=(1);
```

Сега трябва да създадем подкатегориите към "Групи" и "Компании", указвайки за catparent съответно "2" и "4".

```
INSERT INTO categories VALUES (NULL, 'Жени', 2);
INSERT INTO categories VALUES (NULL, 'Мъже', 2);
INSERT INTO categories VALUES (NULL, 'Блъгарски', 4);
```

INSERT INTO categories VALUES (NULL, 'Чужди', 4);

Промяна и изтриване на записи

Промяна на запис

Сигурно забелязвате, че при създаването на категорията "български" сме допуснали правописна грешка. За да я поправим, ще се наложи да променим съдържанието на полето "catname" в записа, в който catid има стойност 7. За целта ще използваме командата **UPDATE** :

UPDATE categories SET catname='Български' WHERE catid='7';

След изпълнението на този ред, грешката ще бъде поправена. Внимавайте с тази конструкция, защото ако пропуснете да посочите условието в WHERE, ще бъде променена стойноста на поелто във всички редове на тази таблица. След малко отново ще се върнем на начините за промяна на вече направени записи.

Изтриване на запис

След като размислим за кратко, можем да решим, че искаме в директорията ни да има само линкове към сайтове за българска музика. Което означава, че ще премахнем категорията за чужди музикални компании. За да направим това ще използваме командата **DELETE**, за да премахнем този запис в таблицата "categories", където "catid" има стойност 8 :

DELETE FROM categories WHERE catid=8;

Сега ще премахнем и категорията "Български", която вече е

излишна :

DELETE FROM categories WHERE catid=7;

В крайна сметка получаваме следния резултат :

Можехме да премахнем двата записа в таблицата само с един ред код, указвайки, че искаме да изтрием редовете, чиито стойности за catid са 7 или (OR) 8 :

DELETE FROM categories WHERE catid=7 OR catid=8 ;

Попълваме директорията

Нека сега добавим по няколко линка във всяка подкатегория на директорията. Сайтовете, които ще използваме в примера са подбрани случайно.

INSERT INTO links VALUES (NULL, 6, 'http://slaviweb.hit.bg', 'Слави Web', 'admin', 'admin@site.com', 0);
INSERT INTO links VALUES (NULL, 5, 'http://milena.hit.bg/', 'Милена', 'admin', 'admin@site.com', 0);
INSERT INTO links VALUES (NULL, 5, 'http://lili.dir.bg/', 'Лили Иванова', 'admin', 'admin@site.com', 0);
INSERT INTO links VALUES (NULL, 6, 'http://valdobrev.hit.bg/', 'Стефан Вълдобрев', 'admin', 'admin@site.com', 0);
INSERT INTO links VALUES (NULL, 3, 'http://www.remapool.com/btr/', 'БТР', 'admin', 'admin@site.com', 0);
INSERT INTO links VALUES (NULL, 3, 'http://wickeda.hit.bg/', 'Уикеда', 'admin', 'admin@site.com', 0);
INSERT INTO links VALUES (NULL, 3, 'http://www.geocities.com/Hollywood/Screen/2009/', 'Нова Генерация', 'admin', 'admin@site.com', 0);
INSERT INTO links VALUES (NULL, 3,

'http://bulgarianspace.com/music/artists/pbb/', 'Подуене Блус Бенд', 'admin', 'admin@site.com', 0);
INSERT INTO links VALUES (NULL, 4, 'http://www.paynermusic.com/', 'Payner Music.com/', 'admin', 'admin@site.com', 0);
INSERT INTO links VALUES (NULL, 4, 'http://www.bol.bg/kamusic/', 'KA Music', 'admin', 'admin@site.com', 0);
INSERT INTO links VALUES (NULL, 4, 'http://bghiphop.hit.bg/producers_sniper.htm', 'Sniper Records', 'admin', 'admin@site.com', 0);

Резултатът виждате на следното изображение :

-php-

MySQL и PHP

Невъзможно е да си представим модерен и високофункционален Уеб сайт, в основата на който не стои база данни. Множество от функциите, които правят страниците привлекателни не биха могли да съществуват, услуги като онлайн банкиране, пазаруване и дори просто организиране на информацията, са немислими без някакъв вид система за бази данни. Няма да навлизаме в технологични и терминологични подробности, а ще покажем как да използвате напрактика една от най-популярните и разпространени в Интернет системи за управление на бази данни - MySQL. Повече за самата система можете да откриете в сайта на фирмата - производителка http://www.mysql.com.
Основната версия на системата е безплатна за използване, без значение с комерсиални или идеални цели, като това е една от основните причини за широката ѝ популярност. Това не означава, че възможностите, които предлага са недостатъчни. Системата може да изпълнява заявките на

неограничен брой потребители, издържа до 50 милиона записа, предлага лесна за използване система за управление на правата на различните потребители, отлична скорост на обработка на заявките.

MySQL

MySQL е система за управление на релационна база данни (БД), която използва Structured Query Language (SQL) - най-популярният език за добавяне, прочитане и обработка на информация в базите данни днес. Системата е с отворен код и използването й се подчинява на лицензa GPL. Първата версия на MySQL се появи през януари 1998 година. Може да се използва с голяма група програмни езици - C, C++, Eiffel, Java, Perl, PHP, Python и Tcl и има версии за Linux, UNIX и Windows.

Системата за управление на релационна база данни (RDBMS) позволява създаването, администрирането и работата с релационi бази данни. Тези БД представляват съвкупности от информационни единици, организирани формално в таблици. Достъпът и промяната на данните се извършва без да е необходимо реорганизирането на таблиците или каквото и да било друго в БД. Едно от най-важните предимства на релационните БД е лекотата с която се създават, четат и изтриват записите, както и лесната разширяемост.

Спецовете обясняват, че релационната БД се състои от таблици, в които данните са подредени по колони(категории). Всеки ред съдържа уникално смислено съчетание от данни (стойности на колоните). Така таблицата всъщност свързва категориите, тя е "релация" между тях. Данните от различни таблици също могат да са свързани -- така имаме отново релация, но този път между таблиците. По този начин таблиците и връзките между тях спояват данните в едно логическо цяло, и именно то е (релационна) базата данни..

Ако последните няколко изречения не са ви ясни, няма

проблеми. За да не ви се смеят "знаещите", трябва само да запомните, че MySQL не е база данни, а система за управление на база данни. И то не на каква да е БД, а на релационна. Толкова с теорията.
За да стане по-ясен текстът от тук нататък, ще кажем, че базата данни се състои от таблици, всяка от които има колони (указващи каква информация трябва да се съхрани там) и редове (представляващи записите). Всяко парче от данните се поставя в полагащото му се сечение между редовете и колоните - т.е. в съответното поле.

Таблиците

За да покажем как се използва MySQL ще направим една примерна и доста упростена директория от линкове. Нещо като dir.bg, където хората отиват, избират си категория и гледат какви линкове е сложил администраторът вътре. Нашият вариант на директория ще е доста по-простичък, но нищо не пречи след време да се поупражнявате върху него и да направите супер-съвършенна система за класификация на адреси.

Тук няма да обясняваме как се създава потребител за MySQL сървъра, как се определят правата му, как се създават бази данни и т.н. Приемаме, че сте си намерили хостинг, където администраторът се е погрижил да ви даде потребителско име, парола, база данни, както и информация за адреса и порта на MySQL сървъра (обикновено 6667).
За онагледяване на примерите ще използваме пакета phpMyAdmin (http://www.phpmyadmin.net/), който предоставя Уеб базиран интерфейс към системата. Ще покажем и основните конструкции за изпращане на запитване към БД сървъра.

За да създадем директорията имаме нужда от две таблици - в едната ще съхраняваме категориите, в другата -

информацията за сайтовете. Логическата връзка между двете таблици ще се осъществява посредством полета, съдържащи вътрешен идентификатор на категориите, както ще покажем след малко. Като допълнителна възможност - всяка категория ще има възможност да съдържа неопределен брой подкатегории.
Таблицата с категориите ще съдържа три колони - идентификатор на категорията (catid), който ще е число, име на категорията (catname), което ще е низ до 100 знака, без значение дали съдържа букви, цифри и т.н. и идентификатор на родителската категория (catparent).
Основният ключ (primary key) на таблицата ще е полето catid, а имената на категориите няма да могат да се повтарят.
Създаваме таблицата така:

```
CREATE TABLE categories (
catid int(20) NOT NULL auto_increment,
catname varchar(100) NOT NULL,
catparent int(20),
PRIMARY KEY (catid),
UNIQUE CatName (catname)
);
```

Таблицата със сайтовете ще има общо седем колони - идентификатор на линка (linkid), идентификатор на категорията, в която е класифициран сайта (catid), който е логическата връзка между двете таблици, URL (url) - низ до 255 знака, име на сайта (linkname) - низ до 100 знака, име на добавилия сайта в директорията (izpratil), неговия e-mail адрес (izpratilemail) и дата на добавяне. Основният ключ на таблицата ще е идентификатора на сайта, а за да не стават повторения, URL полето ще трябва да е уникално. Тази таблица създаваме така:

```
CREATE TABLE links (
```

```
linkid int(20) NOT NULL auto_increment,
catid int(20) DEFAULT '0' NOT NULL,
url varchar(255) NOT NULL,
linkname varchar(100) NOT NULL,
izpratil varchar(100) NOT NULL,
izpratilemail varchar(100) NOT NULL,
data int(20) DEFAULT '0' NOT NULL,
PRIMARY KEY (linkid),
UNIQUE Url (url)
);
```

Създаваме таблиците с SQL инструкцията **CREATE TABLE име (дефиниране на таблицата)**. Синтаксисът може да е доста по-сложен, за подробности погледнете документацията на адрес http://www.mysql.com/doc/en/CREATE_TABLE.html. В скобите указваме структурата на бъдещата таблица, като имаме предвид типа на данните, които искаме да съхраняваме в нея. Отделните иснтрукции се разделят със запетайки, но след последната запетайка не се поставя! Няма значение дали оставяте интервал след запетайката или не.

За създаване на таблици можете да използвате phpMyAdmin, за който вече стана дума, но така или иначе трябва да разберете смисъла на основните типове данни, за да постигнете някакъв резултат. Ако се опитате да създадете таблица, съдържаща полета с неправилно описани изисквания за типа информация, ще получите съобщения за грешка. Ако все пак успеете, то проблемите ще се появят на следващ етап, когато таблицата няма да се държи по начина, по който очаквате.

Базата ни данни вече притежава двете таблици, необходими за директорията :

Типове данни в MySQL

MySQL предлага поддръжка на различни типове данни, но най-вероятно поне на първо време ще са ви необходими четири или пет от тях. Ето описанията на основните, с "x" в скобите е обозначено число, указващо дължината на записа, а квадратни скоби са показани параметри, които не са задължителни.

CHAR (x) - низове с точно определена дължина. Възможната стойност за x е от 1 до 255. Пример :user_id CHAR(10) - очаква се името на модела да е точно 10 символа.

VARCHAR (x) - низове с максимална дължина x. Отново възможните стойности са между 1 и 255. Пример : user_id VARCHAR(10).

INT (x) [Unsigned] - използва се за указване на цели числа между -2147483648 и 2147483647. Ако се използва Unsigned, тогава валидните числа са между 0 и 4294967295. Пример : user_phone INT.

FLOAT [(M,D)] - указва малко дробно число, като M регулира общия брой цифри, от които може да се състои числото, а D ограничава колко от тях може да са зад десетичната запетая. Ако числото е с повече цифри след запетаята, се прави закръгляне. Пример : suma FLOAT (4,2) - означава, че ако поставите в това поле числа като 2,31 или 22,56, а също и 1, 34 или 1,2, те ще бъдат съхранени правилно. Но ако се опитате да сложите нещо като 32,567, на практика в полето ще се запише 32,57. Ако искате да сте сигурни, че няма да се получи закръгляне, можете да използвате типа DECIMAL.

DATE - използва се за съхранение на дати, както говори и името. Форматът по подразбиране е "ГГГГ-ММ-ДД", като можете да поставяте стойности от '0000-00-00' до '9999-12-31'.

MySQL поддържа множество команди за работа с дати, които няма да обсъждаме тук. Пример : born DATE.

TEXT - служи за съхраняване на по-големи низове - от 255 до 65535 символа. При търсене в низовете, съхранявани в такива полета, MySQL няма да прави разлика между малкии големи букви.

BLOB - също като TEXT, с разликата, че търсенето тук взима предвид малките и големи букви.

SET - по-сложна дефиниция, която може би никога няма да използвате, но е добре да знаете. Може да съдържа до 64 предварително дефинирани стойности, от които могат да се избират една или повече. Например : transport SET ("vlak", "avtobus") NOT NULL - от тук имам четири възможности за избор - нито едно от двете, само едното от тях (влак ИЛИ автобус) или и двете (влак И автобус). Както забелязвате, възможните стойности се задават в момента на създаването на таблицата.

ENUM - също като SET, но е позволено избирането само на една стойност. Ако се върнем на горния пример, тук не можем да изберем варианта влак И автобус.

Още за създаването на таблиците

Както забелязахте, когато създавахме двете таблици за директорията - "categories" и "links", дадохме и други инструкции за структурите им, освен типовете данни на отделните полета. Нека погледнем следните инструкциии :

```
linkid int(20) NOT NULL auto_increment,
PRIMARY KEY (linkid),
UNIQUE Url (url)
```

NOT NULL означава, че полето не може да остане празно в нито един от записите в таблицата. Попълването му е задължително, в противен случай ще се появи информация за грешка.

AUTO_INCREMENT означава, че стойността на това поле автоматично ще бъде добавена като към стойността на полето от предходния запис се добави 1. Обикновено при този тип полета посочвате NULL като стойност при добавянето на записа. Използва се често за създаване на уникален идентификатор за отделните записи и може да ви свърши много полезна работа.
PRIMARY KEY (първичен ключ на таблицата) е онова поле, което системата използва за да разграничава различните записи. Не може да има два различни реда с една и съюа тойност на полето, определено за първичен ключ. Всяка таблица има свой PRIMARY KEY.
UNIQUE също ще ви гарантира уникалността на информацията. Например в таблицата с линковете искаме да сме сигурни, че в директорията ни няма да има отправки към един и същ сайт два ПЪТИ.

КНИГА 4
PHP3

Попълване на таблиците

Продължаваме изграждането на примерната директория с добавяне на записите в таблицата за категориите. Съзнателно пропускам подробностите за работа с MySQL от командния ред и в следващите редове ще продължим да използваме phpMyAdmin за да добавяме, редактираме или премахваме записи. Причините за това са две - показаните в шела на Linux таблици от база данни не са най-конфортния начин да разберете какво се случва в базата данни ако тепърва започвате да се занимавате с това. Освен това има доста потребители, ползващи MySQL под Windows, както и такива, чиито хостинг не позволява SSH достъп до сървъра. Затова усложняването е излишно.

Добавяме категориите

Категориите, в които ще разпределяме линковете в нашата директория, ще се съхраняват в таблицата categories, която вече създадохме.
Тя има следните три полета :
- *catid* - число, съставено от до 20 цифри, което ще представлява уникален идентификатор на категорията. Това число ще бъде добавяно автоматично и ще е с една единица по - голямо от числото в предишния запис.
- *catname* - низ, който може да е съставен от различни символи с максимална дължина от 100 символа. Тук ще съхраняваме името на категорията.
- *catparent* - число, съставено от до 20 цифри, което ще указва идентификационния номер (catid) на директорията, която е

родителска на текущата.

Ще създадем категория "Музика", в която ще има три под-категории : "Изпълнители", "Групи", "Компании". В подкатегорията "Изпълнители" ще добавим още две подкатегории - "Жени" и "Мъже", а "Групи" ще разделим на "Български" и "Чужди".

В началото ще създадем трите основни категории. За целта ще използваме командата INSERT с чиято помощ ще вкараме три записа в празната все още таблица.

INSERT INTO таблица VALUES ('първа стойност', 'втора стойност', ...);

Този ред ще създаде нов запис в указаната таблица, попълвайки по реда полетата от ляво надясно - в първото поле ще влезе "първа стойност" и така нататък. Стойностите, които искаме да добавим изписваме винаги в единични кавички. Изключение са само числата, които поставяме в поле с типа на данни INT.

Има два алтернативни записа на показания по-горе код:

INSERT INTO таблица (поле1, поле2) VALUES ('стойност1', 'стойност2');

или

INSERT INTO таблица SET поле1=('стойност1'), поле2=('стойност2');

Както се вижда, двата варианта ви позволяват да създадете запис, без да попълвате всички полета в него. Това обаче може да се направи и с първия вариант. Нека демонстрираме на практика тези редове, като ги използваме за да създадем категориите.

INSERT INTO categories VALUES (NULL, 'Музика', 0);

Както можете да видите на долното изображение, вече имаме

създадена категория "Музика", която е с идентификационен номер (catid) 1 и родителска категория 0 - тоест, тази категория е от най-високо ниво. Забележете, че използвахме NULL за да укажем, че не искаме да укажем стойност за "catid". Съответно около числото 0, което поставихме в поле, дефинирано като INT, не сложихме единичните кавички.

Сега ще добавим трите подкатегории, като вече в "catparent" ще поставяме числото 1, което е идентификор на категорията "Музика". Ще използваме трите различни начина за употреба на INSERT, създавайки трите подкатегории:

```
INSERT INTO categories VALUES (NULL, 'Изпълнители', 1);
INSERT INTO categories (catname, catparent) VALUES ('Групи', 1);
INSERT INTO categories SET catname=('Компании'), catparent=(1);
```

Сега трябва да създадем подкатегориите към "Групи" и "Компании", указвайки за catparent съответно "2" и "4".

```
INSERT INTO categories VALUES (NULL, 'Жени', 2);
INSERT INTO categories VALUES (NULL, 'Мъже', 2);
INSERT INTO categories VALUES (NULL, 'Блъгарски', 4);
INSERT INTO categories VALUES (NULL, 'Чужди', 4);
```

Промяна и изтриване на записи

Промяна на запис

Сигурно забелязвате, че при създаването на категорията "български" сме допуснали правописна грешка. За да я поправим, ще се наложи да променим съдържанието на полето "catname" в записа, в който catid има стойност 7. За целта ще използваме командата **UPDATE** :

```
UPDATE categories SET catname='Български' WHERE catid='7';
```

лед изпълнението на този ред, грешката ще бъде поправена. Внимавайте с тази конструкция, защото ако пропуснете да посочите условието в WHERE, ще бъде променена стойността на поелто във всички редове на тази таблица. След малко отново ще се върнем на начините за промяна на вече направени записи.

Изтриване на запис

След като размислим за кратко, можем да решим, че искаме в директорията ни да има само линкове към сайтове за българска музика. Което означава, че ще премахнем категорията за чужди музикални компании. За да направим това ще използваме командата **DELETE**, за да премахнем този запис в таблицата "categories", където "catid" има стойност 8 :

DELETE FROM categories WHERE catid=8;

Сега ще премахнем и категорията "Български", която вече е излишна :

DELETE FROM categories WHERE catid=7;

В крайна сметка получаваме следния резултат :

Можехме да премахнем двата записа в таблицата само с един ред код, указвайки, че искаме да изтрием редовете, чиито стойности за catid са 7 или (OR) 8 :

DELETE FROM categories WHERE catid=7 OR catid=8 ;

Попълваме директорията

Нека сега добавим по няколко линка във всяка подкатегория на директорията. Сайтовете, които ще използваме в примера са подбрани случайно.

```
INSERT INTO links VALUES (NULL, 6, 'http://slaviweb.hit.bg', 'Слави Web', 'admin', 'admin@site.com', 0);
INSERT INTO links VALUES (NULL, 5, 'http://milena.hit.bg/', 'Милена', 'admin', 'admin@site.com', 0);
INSERT INTO links VALUES (NULL, 5, 'http://lili.dir.bg/', 'Лили Иванова', 'admin', 'admin@site.com', 0);
INSERT INTO links VALUES (NULL, 6, 'http://valdobrev.hit.bg/', 'Стефан Вълдобрев', 'admin', 'admin@site.com', 0);
INSERT INTO links VALUES (NULL, 3, 'http://www.remapool.com/btr/', 'БТР', 'admin', 'admin@site.com', 0);
INSERT INTO links VALUES (NULL, 3, 'http://wickeda.hit.bg/', 'Уикеда', 'admin', 'admin@site.com', 0);
INSERT INTO links VALUES (NULL, 3, 'http://www.geocities.com/Hollywood/Screen/2009/', 'Нова Генерация', 'admin', 'admin@site.com', 0);
INSERT INTO links VALUES (NULL, 3, 'http://bulgarianspace.com/music/artists/pbb/', 'Подуене Блус Бенд', 'admin', 'admin@site.com', 0);
INSERT INTO links VALUES (NULL, 4, 'http://www.paynermusic.com/', 'Payner Music.com/', 'admin', 'admin@site.com', 0);
INSERT INTO links VALUES (NULL, 4, 'http://www.bol.bg/kamusic/', 'KA Music', 'admin', 'admin@site.com', 0);
INSERT INTO links VALUES (NULL, 4, 'http://bghiphop.hit.bg/producers_sniper.htm', 'Sniper Records', 'admin', 'admin@site.com', 0);
```

Резултатът виждате на следното изображение :

Понеже това е първоначалното "зареждане" на директорията, всички линкове бяха вкарани с подател admin, който притежава e-mail адрес admin@site.com. За дата на този етап поставихме нула, по късно ще покажем какви са вариантите за попълване на това поле.

Изграждане на директорията

PHP дава възможност да постигнете резултата, който искате, по различни начини. За да направят директорията, която сега ще изградим, някои ще напишат кода в поне 10 файла, други ще съберат всичко в един. Някои ще използват функции с които да рисуват отделните части от интерфейса, други ще включат HTML кода направо в PHP блоковете.

Нашият вариант ще се състои от 4 файла - два, които ще представляват общия интерфейс и два,.съдържащи PHP кода. Ще имаме header.php, footer.php, index.php и edit.php. Първите два файла ще се грижат за общия вид на директорията, index.php ще показва линковете, а в edit.php ще добавяме, редактираме или изтриваме линкове.

Общият интерфейс

Когато правите голям сайт е добра идея да използвате един и същ HTML код за оформянето на страниците в него. Най-малкото това ще ви спести пренаписванеот на големи количества код, когато се наложи да промените нещо. Подробно този въпрос разгледахме в 5-а част на самоучителя. Сега ще създадем header.php, който ще съдържа "горната" и "лявата" части от интерфейса, но и ще изпълнява допълнителни функции, като например връзка с базата данни :

```
<?
//това е header.php

//име на хоста, на който работи mysql сървъра
$bd_host="";
//име на базата данни
$bd="php_test";
//потребителско име
$bd_user="potrebitel";
//парола за това потребителско име
$bd_pass="parola";
//осъществяне на връзката с базата данни
$link=mysql_connect ($bd_host, $bd_user, $bd_pass);
//избираме базата данни,с която ще работим
mysql_select_db ($bd, $link);
//следва интерфейсът
echo "<HTML>\n<HEAD>\n<TITLE>PC World - Увод в PHP - Примерна директория</TITLE>\n<META NAME=\"KEYWORDS\" CONTENT=\"PHP, Web, tutorial, директория\">\n<META NAME=\"DESCRIPTION\" CONTENT=\"Примерна PHP директория\">\n<META HTTP-EQUIV=\"Content-Type\" CONTENT=\"text/html; charset=windows-1251\">\n";
echo "<body><img src=\"http://www.idg.bg/public_html/bg/iconpcw-small.gif\" alt=\"PC World - Bulgaria\" align=\"left\" hspace=\"20\">\n<h1 style=\"font:bold 2em sans-serif;color:#3300FF;text-align:center;\">\nУвод в PHP - Директория\n</h1>\n<br>\n ";
?>
```

Връзка с базата данни

Преди да направите каквото и да било с базата данни и информацията, съхранявана там, трябва да осъществите връзка с mysql сървъра. За целта трябва да знаете името му, портът на който слуша, трябва да имате потребителско име с

някакви права и парола. Информация за тези неща, както и за имената на базите дани, с които можете да оперирате, ще получите от системния си администратор.

Връзката със сървъра става чрез **mysql_connect ()** :

$vryzka=mysql_connect ("име на хост", "потребител", "парола");

Както забелязахте, в примерния файл (header.php) оставихме променливата $bd_host, съдържаща името на хоста празна. В този случай PHP ще се опита да се свърже с mysql сървър, който работи на локалния компютър - localhost. Всъщност освен да оставим променливата празна, можем да й зададем точно такава стойност:

$bd_host="localhost";

В много случаи обаче ще е необходимо да се свързвате с mysql сървър, който не работи на локалната машина. Например

$bd_host="db.idg.bg";

В този случай се предполага, че базите данни се намират на адрес db.idg.bg, а сървърът слуша на порт 3306, който се използва по подразбиране за mysql. Ако на този хост има няколко mysql сървъра или по някаква причина се ползва друг порт (например 3305), връзката би могла да се направи така :

$vryzka=mysql_connect ("име на хост:порт", "потребител", "парола");

Пример:

$vryzka=mysql_connect ("db.idg.bg:3305", "potrebitel", "parola");

Забележете променливата $vryzka, която от този момент нататък става идентификатор на осъществената връзка. Когато искаме да отправим запитване към базата данни, ще използваме точно тази промелива, за да уточним към коя по-точно база данни се обръщаме. Възможно е в един скрипт да осъществим връзки към няколко бази дани, намиращи се на

различни mysql сървъри. Тогава идентификаторът е особено важен.

След като вече сме се свързали със сървъра, трябва да уточним с коя база данни искаме да работим:

mysql_select_db ("име на БД", "идентификатор на връзка");

Тук, както и във всички останали функции за работа с mysql, идентификаторът на връката не е задължителен. Ако сте отворили връзка само към един mysql сървър, то всички функции ще използват именно тази връзка.

След като приключим заявките към БД е добре да затворим връзката. Това става чрез функцията **mysql_close ("идентификатор")**. Нейното използване обаче не е задължително, защото PHP така или иначе ще затвори връзката след като изпълни текущия скрипт.

Ето и footer.php :

```
<?
mysql_close ($link);
?>
<p align="center" style="font:bold .7em sans-serif;color:#000000">
Самоучител по PHP - Тестова директория
<br>
Поредица на <a href="http://www.pcworld.bg">PC World - Bulgaria</a><br>
<? echo date ("d.m.Y, H:i:s"); ?>
</p>
</body>
</html>
```

Постоянните връзки

Има и още един начин за осъществяване на връзка с базата данни:

$vryzka=mysql_pconnect ("име на хост", "потребител", "парола");

Така отваряме постоянна (persistent) връзка със mysql сървъра, която няма да бъде затворена нито в края на изпълнението на скрипта, нито чрез функцията mysql_close (). Веднъж изградена, връзката си остава отворена за последващи употреби.
Този вариант е добър при наистина натоварени и сложни сайтове, изискващи множество заявки към базата. Но ако използвате споделен mysql сървър, на който разчитат още потребители, постоянните връзки са сигурен източник на проблеми. Освен това в някои версии на PHP резултатите могат да се окажат непредвидими.
С няколко думи - не използвайте mysql_pconnect освен ако не сте 100% убедени, че ви е необходимо, ако системният администратор ви е разрешил и ако сте тествали изцяло как ще се представи сайтът ви.

Търсене в таблиците

Да прегледаме категориите

Сега можем да пристъпим към създаването на индексният файл, който ще показва линковете в директорията. Той има две основни задачи - да провери за налични категории и подкатегории, както и за съответните им линкове.

Заявките се изпращат към базата данни чрез функцията **mysql_query ()** :

$result=mysql_query ("заявка", "идентификатор");

И тук указването на идентификаотр на връзка не е необходимо, ако използваме една база данни. Отговорът от заявката ще бъде записан в променливата $result за по-

нататъчна употреба. Ако по някаква причина възникне грешка, функцията ще върне FALSE.
Ето как ще направим заявка за да проверим кои са категориите от най-високо ниво в нашата директория:

```
$top_cat=mysql_query ("SELECT * FROM categories WHERE catparent=0");
```

Резултатът, който заявката връща е доста по-абстрактен отколкото можете да си представите, така че не можем просто да го изпишем на екрана. най-просто казано той ще представлява масив, съставен от други масиви. За да разберем какво точно ни е върнала базата ще се наложи да обходим резултата. Тъй като знаем, че в таблицата има три полета, ще предвидим три променливи, които да съхраняват стойностите. Сега пред нас остават три възможности. Първата- да получим всеки ред от таблицата в БД във вид на числово номериран масив. Втората възможност е подобна, но ключовете в масива ще отговарят на имената на полетта в таблицата. Третата е да получим резултатат в обектен вид. За да получим редовете във вид на числово индексиран масив, използваме функцията **mysql_fetch_row()**. Тя ще обходи текущия ред от резултата и ще върне масив с ключове от 0 нагоре. При всяко поредно извикване на функцията, тя ще се премества сама на следващия ред в резултата, а когато достигне края ще върне FALSE. Това означва, че можем да я използваме в while цикъл, както показвахме това в 8-ма част на самоучителя :

```
while ($cat_row=mysql_fetch_row ($top_cat)) {
echo "Категорията $cat_row[1] е с ID $cat_row[0] и родителска категория с ID - $cat_row[2]";
}
```

Накратко - заявката, която изпратихме върна само един резултат, защото само една е категорията от най-високо ниво, която дефинирахме.
Чрез mysql_fetch_row получихме всички данни от този ред в

масив, като първото поле (ID на категорията) влезе в елемент с ключ 0, второто поле (името на категорията) влезе в елемент с ключ 1, а третото поле - с ID на родителската категория, откриваме в елемент с ключ 2.

Същият резултат бихме могли да постигнем и чрез функцията **mysql_fetch_array()**, но тук вече няма да се сблъскваме с номерация :

```
while ($cat_row=mysql_fetch_array ($top_cat)) {
echo "Категорията $cat_row[catname] е с ID $cat_row[catid] и родителска категория с ID - $cat_row[catparent]<br>";
}
```

Резултатът ще бъде същия. Нека сега за да демонстрираме цикъла премахнем ограничението за категория и да ги покажем всичките:

```
$top_cat=mysql_query ("SELECT * FROM categories WHERE catparent=0");
while ($cat_row=mysql_fetch_array ($top_cat)) {
echo "Категорията $cat_row[catname] е с ID $cat_row[catid] и родителска категория с ID - $cat_row[catparent]<br>";
}
```

Стана дума за обектното представяне на резултата. Същият листинг на категориите ще получим и ако използваме следния код :

```
$top_cat=mysql_query ("SELECT * FROM categories");
while ($cat_row=mysql_fetch_object ($top_cat)) {
echo "Категорията $cat_row->catname е с ID $cat_row->catid и родителска категория с ID - $cat_row->catparent<br>";
}
```

Тук използвахме функцията **mysql_fetch_object()** за да получим реда от резултата във вид на обект.

Важно : PHP предоставя механизми и за обектно

програмиране, които, макар и да не са сред май-силните му страни, в някои случаи могат да са полезни за решаване на по-трудни задачи. Тъй като в този самоучител не се занимаваме с този аспект на PHP, няма да разглеждаме и възможностите за обектна обработка на резултатите от SQL заявките. Другите два варианта са много по-често използване и са напълно достатъчни за да задоволят всички изисквания.

Съществуа и още един начин за съставяне на конструкцията с която обхождаме резултата. Знаем, че в отговор на заявката ще получим данните от три полета - ID, име и родителска категория. Затова можем да определим свои променливи, които да съхраняват тези резултати:

```
while (list ($id, $name, $parent)=mysql_fetch_array ($top_cat)) {
echo "Категорията $name е с ID $id и родителска категория с ID - $parent<br>";
}
```

Извеждане на категориите и линковете

Сега вече можем да напишем кода на index.php, който ще показва категориите и линковете, съхранявани в тях.
В променливата $p съхраняваме идентификационния номер на текущата директория, за да намерим линковете и поддиректориите, принадлежащи към нея.

```
<?
include "header.php";

//извеждане на категориите
echo "<br><h2>Категории: </h2>";
//ако не съществува променлива $p, търсим подкатегориите на основната
if (!$p) {
```

```
$cat=mysql_query ("SELECT * FROM categories WHERE
catparent=0");
}
else {
//намираме името на текущата категория
$tazi_categoria=mysql_fetch_array(mysql_query ("SELECT
catname FROM categories WHERE catid='$p'"));
echo "Текуща категория :
<b>$tazi_categoria[catname]</b><br>";
//намираме подкатегориите на текущата категория
$cat=mysql_query ("SELECT catid, catname FROM categories
WHERE catparent='$p'");
}
while (list ($id, $name)=mysql_fetch_array ($cat)) {
echo "<li><b><a href=index.php?p=$id>$name</a></b><br>";
}

//извеждане на линковете
echo "<br><br><h2>Линкове: </h2>";
if (!$p) {
$site=mysql_query ("SELECT linkid, url, linkname FROM links
WHERE catid=0");
}
else {
$site=mysql_query ("SELECT linkid, url, linkname FROM links
WHERE catid='$p'");
}
//ако няма намерени резултати, изписваме съответното
съобщение
if (!@mysql_num_rows($site)) {
echo "Няма линкове в тази категория!<br><a
href=\"edit.php?p=$p\">Кликнете тук за да добавите сайт!</a>";
}
//в противен случай извеждаме линковете
else {
```

```
while (list ($lid, $lurl, $lname)=mysql_fetch_array ($site)) {
echo "<li><b><a href=$lurl>$lname</a></b> (<a
href=\"edit.php?e=$lid\">редактирай</a>)<br>";
}
echo "<br><br><i><a href=\"edit.php?p=$p\">Кликнете тук за да
предложите сайт!</a></i>";
}
include "footer.php"
?>
```

Управление на линковете

Сега е време да създадем и четвъртия файл за примерната директория, където ще става добавянето, промяната и изтриването на линковете. Той ще показва и формите, в които потребителите ще въвеждат съответната информация.
Ако разгледате внимателно кода на index.php ще видите, че този файл може да бъде извикан с посочени стойности за променливите $p или $e. В зависимост от това коя променлива притежава стойност,. ще бъдат предприемани и съответните действия.
Например ако променливата $p има стойност, а $e е празна, очевидно искаме да добавяме нов линк в директорията, чието ID се съдържа в $p. Ако само $e съдържа стойност, то явно ще редактираме линкът, чието ID е в нея. Ако и двете променливи са празни, ще добавяме сайт към основната категория.
В случаите, когато ще се добавя или редактира линк, ще е необходимо да се изведе и съответната форма. След извършване на съответното действие ще покажем потвърждение или съобщение за грешка с връзка към индекса на директорията.
Вътре в самия файл обаче ще въведем и още няколко роменливи - $ins ще съдържа категорията на евентуалния нов линк, $del ще съдържа ID-то на линк, който евентуално искаме да изтрием, $do_edit на линк, който евентуално ще

променяме. Ето и кода :

```
<?
include "header.php";

//функция, която ще използваме за да проверяваме валидността на информацията във формата
function check_form () {
global $link, $linkname, $linkurl, $linkcat, $izpratil, $izpratil_email;
//дали посоченият адрес е валиден?
if (!eregi("^[a-z0-9]+([_\\.-][a-z0-9]+)*@([a-z0-9]+([\.-][a-z0-9]+))*$",$izpratil_email) ) {
echo "Посочили сте невалиден e-mail адрес! Моля, върнете се и поправете грешката!";
include "footer.php";
exit;
}
//дали всички полета са попълнени?
if ((!$linkname) || (!$linkurl) || (!$linkcat) || (!$izpratil) || (!$izpratil_email)) {
echo "Не сте попълнили всички задължителни полета! Моля, върнете се и поправете грешката!";
include "footer.php";
exit;
}
}

//проверка за променлива $del за да изтрием линка
//идентификационният номер на линка ще намерим в променливата $linkid
if ($del) {
if (!mysql_query ("DELETE FROM links WHERE linkid='$linkid'"))
{
echo "<font color=\"#FF0099\">Линкът не може да бъде премахнат в момента!</font>";
include "footer.php";
```

```
exit;
}
else {
echo "Линкът бе премахнат! <br><br><a
href=\"index.php\">Обратно към директорията.</a>";
include "footer.php";
exit;
}
}

//проверка за промелива $ins
if ($ins) {
//проверка за валидността на формата
check_form ();
//определяне на датата
$data=date ("Y-m-d");
//добавяне на сайта
if (!mysql_query ("INSERT INTO links VALUES (NULL, '$linkcat',
'$linkurl', '$linkname', '$izpratil', '$izpratil_email', '$data')")) {
echo "<font color=\"#FF0099\">Линкът не може да бъде
добавен в момента!</font>";
include "footer.php";
exit;
}
else {
echo "Линкът бе добавен! <br><br><a
href=\"index.php\">Обратно към директорията.</a>";
include "footer.php";
exit;
}
}

//проверка за променлива $do_edit
if ($do_edit) {
//проверка за валидността на формата
```

```
check_form ();
$data=date ("Y-m-d");
//редактиране на сайта
if (!mysql_query ("UPDATE links SET catid='$linkcat',
url='$linkurl', linkname='$linkname', izpratil='$izpratil',
izpratilemail='$izpratil_email', data='$data' WHERE
linkid='$linkid'")) {
echo "<font color=\"#FF0099\">Линкът не може да бъде
редактиран в момента!</font>";
include "footer.php";
exit;
}
else {
echo "Линкът бе редактиран! <br><br><a
href=\"index.php\">Обратно към директорията.</a>";
include "footer.php";
exit;
}

}

//проверка за променлива $e - ако ще редактираме линк,
трябва да изведем текущата информация за него
if ($e) {
//дали има линк с това ID или някой се шегува?
$link_info=mysql_query("SELECT * FROM links WHERE
linkid='$e'");
if (!mysql_num_rows($link_info)) {
echo "Няма такъв сайт в директорията<br><br><a
href=\"index.php\">Моля, проверете отново!</a>";
include "footer.php";
exit;
}
//ако ID-то е коректно, изтегляме информацията за сайта
//всички данни ще се съхраняват в масива $link_arr, като
```

```
ключове за отделните му елементи са имената на полетата
$link_arr=mysql_fetch_array($link_info);
}

//извеждаме формата за въвеждане на информация
//ако в масива $link_arr има дани, те ще се изпишат
?>
<b>Въведете желаната информация</b> : <br><br>
<form method="post" action="edit.php">
Име на сайта : <br><input type="text" name="linkname"
value="<? echo $link_arr[linkname]; ?>"><br><br>
Адрес : <br><input type="text" name="linkurl" value="<? echo
$link_arr[url]; ?>"><br><br>
Категория : <br> <select name="linkcat">
<?
//създаваме падащо меню, съдържащо списък на категориите
$cat_list=mysql_query("SELECT catid, catname FROM
categories");
while (list($catid, $catname)=mysql_fetch_row($cat_list)) {
//ако категорията е тази, която ни е необходима, определяме
тази опция като избрана
if (($catid==$p) || ($catid==$link_arr[catid])) {
echo "<option value=$catid SELECTED>$catname</option>";
} else {
echo "<option value=$catid>$catname</option>";
}
}
?>
</select><br><br>
Вашето име :<br><input type="text" name="izpratil" value="<?
echo $link_arr[izpratil]; ?>"><br><br>
Вашият e-mail адрес : <br><input type="text"
name="izpratil_email" value="<? echo $link_arr[izpratilemail];
?>"><br><br>
<input type="submit" value="Изпрати информацията!">  <input
```

```
type="checkbox" name="del"> Изтрий записа!
<?
//поставяме "флаг" за добавяне или изтриване
if ($e) {
echo "<input type=hidden name=linkid value=$e>";
echo "<input type=hidden name=do_edit value=1>";
}
else {
echo "<input type=hidden name=ins value=1>";
}
?>
</form>
<?
include "footer.php";
?>
```

Този код може да ви се види сложен в началото, но той демонстрира възможностите, които PHP ви дава да изберете как да структурирате приложението си. Някои биха поставили съдържанието на edit.php в поне 4 файла, други пък биха събрали цялата директория в един (поставяйки header и footer съдържанието във функции, а целия edit.php в if конструкции). Демонстрационната директория не е особено красива и функционална, но това е основа, върху която могат да се добавят нови и нови елементи.

Понеже това е първоначалното "зареждане" на директорията, всички линкове бяха вкарани с подател admin, който притежава e-mail адрес admin@site.com. За дата на този етап поставихме нула, по късно ще покажем какви са вариантите за попълване на това поле.

Изграждане на директорията

PHP дава възможност да постигнете резултата, който искате,

по различни начини. За да направят директорията, която сега ще изградим, някои ще напишат кода в поне 10 файла, други ще съберат всичко в един. Някои ще използват функции с които да рисуват отделните части от интерфейса, други ще включат HTML кода направо в PHP блоковете.

Нашият вариант ще се състои от 4 файла - два, които ще представляват общия интерфейс и два,.съдържащи PHP кода. Ще имаме header.php, footer.php, index.php и edit.php. Първите два файла ще се грижат за общия вид на директорията, index.php ще показва линковете, а в edit.php ще добавяме, редактираме или изтриваме линкове.

Общият интерфейс

Когато правите голям сайт е добра идея да използвате един и същ HTML код за оформянето на страниците в него. Най-малкото това ще ви спести пренаписванеот на големи количества код, когато се наложи да промените нещо. Подробно този въпрос разгледахме в 5-а част на самоучителя. Сега ще създадем header.php, който ще съдържа "горната" и "лявата" части от интерфейса, но и ще изпълнява допълнителни функции, като например връзка с базата данни :

```
<?
//това е header.php

//име на хоста, на който работи mysql сървъра
$bd_host="";
//име на базата данни
$bd="php_test";
//потребителско име
$bd_user="potrebitel";
//парола за това потребителско име
```

```
$bd_pass="parola";
//осъществяне на връзката с базата данни
$link=mysql_connect ($bd_host, $bd_user, $bd_pass);
//избираме базата данни,с която ще работим
mysql_select_db ($bd, $link);
//следва интерфейсът
echo "<HTML>\n<HEAD>\n<TITLE>PC World - Увод в PHP - Примерна директория</TITLE>\n<META NAME=\"KEYWORDS\" CONTENT=\"PHP, Web, tutorial, директория\">\n<META NAME=\"DESCRIPTION\" CONTENT=\"Примерна PHP директория\">\n<META HTTP-EQUIV=\"Content-Type\" CONTENT=\"text/html; charset=windows-1251\">\n";
echo "<body><img src=\"http://www.idg.bg/public_html/bg/iconpcw-small.gif\" alt=\"PC World - Bulgaria\" align=\"left\" hspace=\"20\">\n<h1 style=\"font:bold 2em sans-serif;color:#3300FF;text-align:center;\">\nУвод в PHP - Директория\n</h1>\n<br>\n ";
?>
```

Връзка с базата данни

Преди да направите каквото и да било с базата данни и информацията, съхранявана там, трябва да осъществите връзка с mysql сървъра. За целта трябва да знаете името му, портът на който слуша, трябва да имате потребителско име с някакви права и парола. Информация за тези неща, както и за имената на базите дани, с които можете да оперирате, ще получите от системния си администратор.

Връзката със сървъра става чрез **mysql_connect ()** :

```
$vryzka=mysql_connect ("име на хост", "потребител", "парола");
```

Както забелязахте, в примерния файл (header.php) оставихме променливата $bd_host, съдържаща името на хоста празна. В този случай PHP ще се опита да се свърже с mysql сървър, който работи на локалния компютър - localhost. Всъщност

освен да оставим променливата празна, можем да й зададем точно такава стойност:

$bd_host="localhost";

В много случаи обаче ще е необходимо да се свързвате с mysql сървър, който не работи на локалната машина. Например

$bd_host="db.idg.bg";

В този случай се предполага, че базите данни се намират на адрес db.idg.bg, а сървърът слуша на порт 3306, който се използва по подразбиране за mysql. Ако на този хост има няколко mysql сървъра или по някаква причина се ползва друг порт (например 3305), връзката би могла да се направи така :

$vryzka=mysql_connect ("име на хост:порт", "потребител", "парола");

Пример:

$vryzka=mysql_connect ("db.idg.bg:3305", "potrebitel", "parola");

Забележете променливата $vryzka, която от този момент нататък става идентификатор на осъществената връзка. Когато искаме да отправим запитване към базата данни, ще използваме точно тази промелива, за да уточним към коя по-точно база данни се обръщаме. Възможно е в един скрипт да осъществим връзки към няколко бази дани, намиращи се на различни mysql сървъри. Тогава идентификаторът е особено важен.

След като вече сме се свързали със сървъра, трябва да уточним с коя база данни искаме да работим:

mysql_select_db ("име на БД", "идентификатор на връзка");

Тук, както и във всички останали функции за работа с mysql, идентификаторът на връката не е задължителен. Ако сте отворили връзка само към един mysql сървър, то всички функции ще използват именно тази връзка.

След като приключим заявките към БД е добре да затворим връзката. Това става чрез функцията **mysql_close ("идентификатор")**. Нейното използване обаче не е задължително, защото PHP така или иначе ще затвори връзката след като изпълни текущия скрипт.

Ето и footer.php :

```
<?
mysql_close ($link);
?>
<p align="center" style="font:bold .7em sans-serif;color:#000000">
Самоучител по PHP - Тестова директория
<br>
Поредица на <a href="http://www.pcworld.bg">PC World - Bulgaria</a><br>
<? echo date ("d.m.Y, H:i:s"); ?>
</p>
</body>
</html>
```

Постоянните връзки

Има и още един начин за осъществяване на връзка с базата данни:

```
$vryzka=mysql_pconnect ("име на хост", "потребител", "парола");
```

Така отваряме постоянна (persistent) връзка със mysql сървъра, която няма да бъде затворена нито в края на изпълнението на скрипта, нито чрез функцията mysql_close (). Веднъж изградена, връзката си остава отворена за последващи употреби.
Този вариант е добър при наистина натоварени и сложни сайтове, изискващи множество заявки към базата. Но ако използвате споделен mysql сървър, на който разчитат още

потребители, постоянните връзки са сигурен източник на проблеми. Освен това в някои версии на PHP резултатите могат да се окажат непредвидими.
С няколко думи - не използвайте mysql_pconnect освен ако не сте 100% убедени, че ви е необходимо, ако системният администратор ви е разрешил и ако сте тествали изцяло как ще се представи сайтът ви.

Търсене в таблиците

Да прегледаме категориите

Сега можем да пристъпим към създаването на индексният файл, който ще показва линковете в директорията. Той има две основни задачи - да провери за налични категории и подкатегории, както и за съответните им линкове.

Заявките се изпращат към базата данни чрез функцията **mysql_query ()** :

$result=mysql_query ("заявка", "идентификатор");

И тук указването на идентификаотр на връзка не е необходимо, ако използваме една база данни. Отговорът от заявката ще бъде записан в променливата $result за по-нататъчна употреба. Ако по някаква причина възникне грешка, функцията ще върне FALSE.
Ето как ще направим заявка за да проверим кои са категориите от най-високо ниво в нашата директория:

$top_cat=mysql_query ("SELECT * FROM categories WHERE catparent=0");

Резултатът, който заявката връща е доста по-абстрактен отколкото можете да си представите, така че не можем просто да го изпишем на екрана. най-просто казано той ще представлява масив, съставен от други масиви. За да разберем какво точно ни е върнала базата ще се наложи да обходим резултата. Тъй като знаем, че в таблицата има три

полета, ще предвидим три променливи, които да съхраняват стойностите. Сега пред нас остават три възможности. Първата- да получим всеки ред от таблицата в БД във вид на числово номериран масив. Втората възможност е подобна, но ключовете в масива ще отговарят на имената на полетта в таблицата. Третата е да получим резултатат в обектен вид. За да получим редовете във вид на числово индексиран масив, използваме функцията **mysql_fetch_row()**. Тя ще обходи текущия ред от резултата и ще върне масив с ключове от 0 нагоре. При всяко поредно извикване на функцията, тя ще се премества сама на следващия ред в резултата, а когато достигне края ще върне FALSE. Това означва, че можем да я използваме в while цикъл, както показвахме това в 8-ма част на самоучителя :

```
while ($cat_row=mysql_fetch_row ($top_cat)) {
echo "Категорията $cat_row[1] е с ID $cat_row[0] и родителска категория с ID - $cat_row[2]";
}
```