А. В. Фоменко,
Г. О. Козуб

Технологія WEB-програмування

PHP, MYSQL

Курс лекцій

Міністерство освіти і науки, молоді та спорту України
Державний заклад
„Луганський національний університет
імені Тараса Шевченка”

А. В. Фоменко,
Г. О. Козуб

Технологія WEB-програмування

PHP, MYSQL

Курс лекцій

з дисципліни „Технологія Web-програмування”
для студентів 5 курсу спеціальності „Інформатика” та
для студентів 3 курсу спеціальності „Комп’ютерна інженерія”

Луганськ
ДЗ „ЛНУ Тараса Шевченка”
2011

УДК 004.738.1:WWW(075.8)
ББК 32.973.202-018я73
Ф76

Р е ц е н з е н т и :

Ашеров А. Т. – доктор технічних наук, професор, завідувач кафедри інформатики і комп'ютерних технологій Української інженерно-педагогічної академії.

Грибанов В. М. – доктор технічних наук, професор, завідувач кафедри прикладної математики Східноукраїнського національного університету імені Володимира Даля.

Адаменко О. В. – доктор педагогічних наук, професор кафедри теоретичної і прикладної інформатики Луганського національного університету імені Тараса Шевченка.

Фоменко А. В.

Ф76 Технологія Web-програмування курс лекцій для студ. спец. „Інформатика", „Комп'ютерна інженерія" / А. В. Фоменко, Г. О. Козуб. – Луганськ : Вид-во ДЗ „ЛНУ Тараса Шевченка", 2011. – 216 с.

Курс лекцій „Технологія Web-програмування" структуровано відповідно до розділів робочої програми дисципліни, містять орієнтовну будову модулів з курсу, запитання для самоконтролю, питання та завдання до самостійної і модульної роботи, довідник основних функцій, термінологічний словник, список рекомендованої літератури.

Курс лекцій призначено для студентів, магістрантів спеціальностей „Інформатика", „Комп'ютерна інженерія", викладачів ВНЗ, учителів-предметників загальноосвітніх шкіл, ліцеїв, коледжів, гімназій, слухачів курсів, а також для самоосвіти.

УДК 004.738.1:WWW(075.8)
ББК 32.973.202 - 018я73

Рекомендовано до друку навчально-методичною радою
Луганського національного університету
імені Тараса Шевченка
(протокол № 5 від 19 січня 2011 року)

Зміст

Передмова

Сучасне суспільство неможливе без Internet і Internet-технологій, тому кожен студент повинен, у тому або іншому ступені, мати уявлення про них. Більшою мірою це стосується майбутніх учених і керівників, а також, фахівців у галузі інформаційних технологій.

Мета дисципліни – набуття студентами знань про Web-програмування, опанування можливостей мови PHP для програмування Web-сайтів і Web-інтерфейсів.

Основні завдання дисципліни:

- формування уявлень про сучасні Інтернет–технології, протоколи передачі даних, клієнт-серверну структуру веб–додатків, URL–адресу, CGI–сценарії, веб-технології;
- ознайомлення з мовою програмування PHP;
- набуття навичок роботи з PHP;
- набуття навичок з розробки Internet-додатків.

У результаті вивчення дисципліни студенти повинні *мати уявлення:*

- про основи роботи в мережі Інтернет;
- про програми, необхідні для роботи з мовою PHP;
- про використання PHP для створення Web-сайтів;

повинні *знати*:

- основи інтерфейсу CGI;
- протоколи обміну інформацією Web-серверів і клієнтських браузерів;
- принципи розробки Web-сайтів за допомогою засобів і методів мови PHP.

У результаті вивчення дисципліни студенти повинні *вміти*:

- використовувати основні методи і засоби мови PHP і способи їх застосування для створення Web-сайтів;
- розробляти Web-сайти за допомогою засобів мови PHP.

Цей посібник призначено для опанування студентами спеціальностей „Інформатика", „Комп'ютерна інженерія" мовою PHP для програмування Web-сайтів і Web-інтерфейсів, а також

розробки Web-додатків необхідних для використання спеціальних засобів, які дозволять застосовувати комп'ютер як найсучасніший технічний засіб.

Курс лекцій поділений на модулі й вміщує 9 лекцій, де кожна містить план; теоретичні відомості стосовано дисципліни; поняття та пояснення термінів; багато прикладів, котрі допоможуть чітко уявити механізм роботи мови PHP при розробці web-додатків та контрольні питання для перевірки засвоєння теоретичного матеріалу. Для якіснішого сприйняття навчального матеріалу визначення виділено світлим тонуванням, а пояснення та зауваження темним тонуванням; наведено питання та завдання до модульних і самостійних робіт; довідник з основних функцій, покажчик. Список літератури дозволить розширити отримані знання тим, чия професійна діяльність буде пов'язана з Internet-технологіями.

Модуль 1

Лекція 1. Вступ до вивчення дисципліни технології Web-програмування

План:

1. Веб-технології. Поняття, особливості, застосування.
2. Сайти та сторінки.
3. Веб-сервер. Принципи побудови взаємодії клієнта і сервера. Поняття про клієнтські та серверні застосування.
4. Веб-браузери. Протокол http. Структура протоколу http.
5. Програмування на стороні клієнта
6. Серверні скрипти. Обробка скриптів на серверній мові.

Терміни і поняття

Web, World Wide Web, Всесвітня павутина – глобальний інформаційний простір, заснований на фізичній інфраструктурі Інтернету й протоколі передачі даних HTTP.

Internet, Інтернет, "Инет", Internet, Interconnected Networks, "Всесвітня мережа", "Глобальна мережа", "Всесвітня павутина" – всесвітня система добровільно об'єднаних комп'ютерних мереж, побудована на використанні протоколу IP і маршрутизації пакетів даних. Інтернет утворює всесвітнє (єдину) інформаційне середовище – спосіб організації оцифрованої інформації. Інтернет слугує фізичною основою для Всесвітньої павутини.

У буденній мові нині Web, Веб, Internet, Інтернет–майже синоніми і вживаються без розподілу понять.

Веб-технології (іноді також вживається в однині)– комплекс технічних, комунікативних, програмних методів вирішення завдань організації спільної діяльності користувачів із застосуванням мережі Інтернет.

Контент – інформація, яка розміщена на веб-сторінках.

URI сторінки– унікальна адреса сторінки в мережі Інтернет.

Приклад: `http://www.yandex.ru/all services.html`

URI містить:

1.Метод доступу до ресурсу, тобто протокол доступу (http).

2.Мережний адрес ресурсу (www.yandex.ru).

3. Повний шлях до файлу на сервері (all_services.html).

Web-технології

Особливості

Web-технології є концепцією роботи з інформацією.

Вона відрізняється такими особливостями:

- технічна основа Web-технологій – локальні й глобальні мережі, часто Інтернет;
- застосування особливого типу тонких клієнтів: web-браузерів (типи й історія, сучасний стан браузерів віддаються на самостійне вивчення);
- переважно текстова і статично-графічна подача інформації (послаблення цієї тенденції пов'язане з розвитком технологій зв'язку і ПО, експансією медіаконтенту);
- зміни в інформаційних джерелах миттєво відбиваються в публікаціях;
- число споживачів інформації практично не обмежене: публікатор сам може задати особливі умови на доступ до публікованої інформації;
- у публікаціях можуть знаходитися посилання на інші публікації без обмеження на місце розташування і джерела матеріалів;
- активна робота пошукових машин (історія, сучасний стан і роль пошукових машин віддаються на самостійне вивчення);
- доставлення та тиражування контенту практично безкоштовні.

Переваги

Привабливість Web-технологій як засоби передачі й надання інформації багато в чому визначає універсальний інтерфейс між людиною і комп'ютером. Кожній людині зрозумілі написи, заголовки, посилання, картинки. Веб-інтерфейс як засіб доступу до інформації інтуїтивно зрозумілий.

Наслідком простоти веб-інтерфейсу є широке використання Інтернету як каналу комунікації.

Браузер (англ. *browser*-програма для перегляду веб-сторінок і роботи з інформацією у веб-інтерфейсі.

Браузери*-програми, якими забезпечені всі сучасні комп'ютери, велике число т.зв. „гаджетів"*.

Теоретично всі браузери повинні відображувати усі сайти, зроблені за стандартами, однаково. Практично є безліч тонкощів і складнощів. Найбільш популярні браузери: Internet Explorer, FireFox, Opera, Safari, Chrome.

Інтеграційна роль веб-технологій.

Значення Web-технології як для розробників програмного забезпечення, так і для звичайних користувачів багато в чому визначається тим, що це, передусім-інтеграційна технологія.

Веб-технології дозволяють створювати прості для освоєння, легкодоступні, украй дешеві, швидко оновлюванні інформаційні, діалогові, довідкові системи та спеціальні сервіси.

Сайти і сторінки, сервіси, портали

Web-сторінка *(гіпертекстовий документ)*-це документ, описаний мовою HTML. Основна відмінність їх від текстових документів полягає в тому, що вони можуть містити посилання на інші аналогічні документи. Сторінкою називають те, що показує браузер при введенні адреси сторінки або при переході до посилання.

Сайтом (веб-сайтом) називають сукупність сторінок, створених із застосуванням програмного забезпечення і такі що

утворюють єдине ціле в технічному, інформаційному й навігаційному аспекті. Найчастіше усі сторінки сайту мають загальне доменне ім'я (поняття доменного імені, системи DNS-серверів, структура www віддаються на самостійне вивчення).

Інтерфе́йс-сукупність засобів і методів взаємодії між елементами системи. Інтерфейси є основою взаємодії усіх сучасних інформаційних систем.

Інтерфейс користувача, призначений для користувача інтерфейс (UI-англ. *user interface*)-різноманітність інтерфейсів, у якому одна сторона представлена людиною (користувачем), інша-машиною/пристроєм. Є сукупність засобів і методів, за допомогою яких користувач взаємодіє з різними, найчастіше складними, машинами і пристроями.

Оскільки інтерфейс є сукупністю, то він складається з елементів, які, самі по собі також можуть складатися з елементів (так, вікно дисплея може містити в собі інші вікна, які, у свою чергу, можуть містити панелі, кнопки й інші інтерфейсні елементи).

Особлива і окрема увага в інтерфейсі користувача традиційно приділяється його *ефективності та зручності користування* (юзабельності).

Юзабіліті (англ. *usability*-дослівно „можливість використання", „здатність бути використаним", „корисність")-поняття в мікроергономіці, що означає підсумковий рівень зручності предмета для використання в заявлених цілях. Термін має зв'язок з поняттям „**ергономічність"**, але, на відміну від останнього менше асоціюється з технічною естетикою, із зовнішнім виглядом і більш прив'язаний до утилітарності „юзабільного" об'єкта.

Міжнародний стандарт ISO 9241-11 визначає юзабіліті як „міру, з якою продукт може бути використаний певними користувачами при певному контексті використання для досягнення певної мети з належною ефективністю, продуктивністю і задоволеністю".

Сервіс (веб-сервіс)-спеціалізований сайт для вирішення потрібних відвідувачам досить вузьких завдань.

Приклад *сервісу-mail.ru або facebook.com.*

Веб-служба, веб-сервіс (англ. *web service*)-програмна система, що ідентифікується рядком URI, чиї загальнодоступні інтерфейси визначені мовою XML. Опис цієї програмної системи може бути знайдений іншими програмними системами, які можуть взаємодіяти з нею згідно з цим описом за допомогою повідомлень, заснованих на XML, і переданих за допомогою інтернет-протоколів. Веб-служба є одиницею модульності при використанні сервіс-орієнтованій архітектури додатку.

Сервер-програмний компонент обчислювальної системи, що виконує сервісні (обслуговуючі) функції по запиту клієнта, надаючи йому доступ до певних ресурсів або послуг.

Веб-сервер-це сервер, приймаючий HTTP-запити від клієнтів, зазвичай веб-браузерів, і що видає їм HTTP-відповіді, зазвичай разом з HTML-сторінкою, зображенням, файлом, медіапотоком або іншими даними. Веб-сервери - основа Всесвітньої павутини.

Веб-сервером називають таке програмне забезпечення, що виконує функції веб-сервера, так і безпосередньо комп'ютер, на якому це програмне забезпечення працює.

Єдиний покажчик ресурсів (англ. ***URL*** – ***Uniform Resource Locator***) –однаковий локатор (визначник місцезнаходження) ресурсу. Раніше називався **Universal Resource Locator**– універсальний локатор ресурсу. URL– це стандартизований спосіб запису адреси ресурсу в мережі Інтернет.

Клієнт, яким зазвичай є веб-браузер, передає веб-серверу запити на отримання ресурсів, позначених URL–адресами. Ресурси – це HTML–сторінки, зображення, файли, медіа-потоки або інші дані, які потрібні клієнтові. У відповідь веб-сервер передає клієнтові запитані дані. Цей обмін відбувається за протоколом HTTP.

Іншими словами, **Web–сервер** (точніше – *http-сервер*) – спеціальна програма, встановлена на загальнодоступному комп'ютері.

Портал. Виділяють два значення.

1. Великий тематичний сайт, що активно розвивається і відвідуваний, з чітко певною і зростаючою аудиторією і поступово поповнюваний сервісами.

Приклади: *www.oboz.ua*, *www.uapouisk.net.*

2. Сайт без вираженої тематичної спрямованості або з широкою тематикою (наприклад, географічною), що має у своєму складі безліч сервісів, служб і напрямів взаємодії.

Приклади: *meta.ua.*

Веб-сервер і браузер. Взаємодія

Усі сайти відкриваються браузерами як html-документи. HTML-документ може містити зображення, відеофрагменти, flash-ролики і звуки.

Усі сучасні сайти мають динамічні елементи, тобто фрагменти контенту, які змінюються в часі, а так само залежно від того, хто саме зайшов на сторінку, і навіть можуть редагуватися самими відвідувачами.

Для того, щоб ці функціональні елементи працювали, необхідно, щоб сервер, що видає сторінки, мав спеціалізоване програмне забезпечення, що працює з базою даних і створює сторінки динамічно. (Уявлення про те, що усі сторінки усіх сайтів, аж до видачі пошукових машин, формуються в html вручну, є примітивною, але украй поширеною помилкою.)

Браузер і веб-сервер взаємодіють за технологією клієнт-сервер. Після введення адреси в рядок адреси браузер формує запит до сервера. Сервер формує сторінку і передає її браузеру. Браузер виводить сторінку користувачеві, який своїми діями формує новий запит.

Для того, щоб HTML-сторінка стала динамічною, тобто могла залежати від поведінки людини і/або зовнішніх подій, існує кілька технологій. Перший поділ пов'язаний з місцем здійснення цього пожвавлення: на сервері або на клієнтові. Далі по-

діл ведеться за методом програмування: з використанням скриптів, що інтерпретуються, або програм, що відкомпілювалися.

Сторона клієнта		Сторона сервера	
Скрипти	Програми	Скрипти	Програми
JavaScript	Plug – ins	Ruby	окремі CGI - програми
JScript	ActiveX	Perl	модулі серверу
VBScript	Java applets	PHP	ISAPI/NSAPI
		Shell-код	Java servlets

Веб-браузери

Сучасний браузер виконує такі функції:

- забезпечення передачі даних по протоколах *http і https*;
- обробка контенту в-сторінки (*css, JavaScript, медіа-файли);*

Практично всі популярні браузери поширюються безкоштовно або „в комплекті" з іншими додатками. Браузери класифікуються:

1) **За-типом пристрою для якого написані:**

- браузери *(Internet Explorer, Firefox, Safari, Opera*);
- браузери для портативних пристроїв (*Internet Explorer Mobile, Opera Mini, Safari для Apple iPhone).*

2) **За-типом інтерфейсу :**

- графічні браузери (усі вище названі);
- текстові браузери *(Lynx, Links, W3M, Netrik, Elinks, Internet Browser).*

3) **За-режимом доступу :**

- online-браузери (все вище перераховані)-браузер намагається отримати сторінки з веб-сервера;
- offline-браузери *(HTTrack, Offline Explorer, Teleport Pro, WebZip)* - переглядати можна тільки збережені копії веб-сайтів або їх частини.

При відвідуванні веб-сайту клієнтський додаток (такий як браузер, пошукові роботи) зазвичай посилає веб-серверу інформацію про себе. Це текстовий рядок, що є частиною HTTP запиту, починається з *User-Agent*:, і що зазвичай містить таку інформацію, як назва і версія застосування, операційна система комп'ютера та мова. У „пошукових роботів" цей рядок часто

містить *URL* та *e-mail* адреса, за якими веб-майстер може зв'язатися з оператором „робота".

Протокол HTTP

HTTP (англ. *HyperText Transfer Protocol*-„протокол передачі гіпертексту")-протокол прикладного рівня передачі даних. Основою HTTP є технологія „клієнт-сервер", тобто передбачається існування споживачів (клієнтів), які ініціюють з'єднання і посилають запит, і постачальників (серверів), які чекають з'єднання для отримання запиту, виконують необхідні дії і повертають назад сполучення з результатом. HTTP нині повсюдно використовується у Всесвітній павутині для отримання інформації з веб-сайтів.

Основним об'єктом маніпуляції в HTTP є ресурс, на який вказує URI (англ. *Uniform Resource Identifier*) у запиті клієнта.

HTTP-протокол прикладного рівня. Обмін повідомленнями відбувається за звичайною схемою „запит-відповідь".

На відміну від багатьох інших протоколів, HTTP не зберігає свого стану. Це означає відсутність збереження проміжного стану між парами „запит-відповідь".

Компоненти, використовуючи HTTP (браузер і сервер), можуть самостійно здійснювати збереження інформації про стан, пов'язаної з останніми запитами і відповідями. Браузер, що надсилає запити, може відстежувати затримки відповідей. Сервер може зберігати IP-адреси і заголовки запитів останніх клієнтів. Проте сам протокол не обізнаний про попередні запити і відповіді, у ньому не передбачена внутрішня підтримка стану, до нього не висуваються такі вимоги.

Структура протоколу HTTP

Кожне HTTP-повідомлення складається з трьох частин, які передаються у зазначеному порядку:

- стартовий рядок (англ. *Starting line*)-визначає тип повідомлення;
- заголовки (англ. *Headers*)-характеризують тіло повідомлення, параметри передачі й інші відомості;

•тіло повідомлення (англ. *Message Body*)-безпосередньо ці повідомлення. Обов'язково повинно відділяти від заголовків порожнім рядком.

Заголовки й тіло повідомлення можуть бути відсутніми, але стартовий рядок є обов'язковим елементом, оскільки вказує на тип запиту/відповіді.

Стартові рядки розрізняються для запиту і відповіді.

Рядок запиту виглядає так:

GET URI-для версії протоколу 0.9. Метод URI HTTP/Версія-для інших версій.

Тут:

Метод (англ. *Method*)-назва запиту, одне слово заголовними буквами. У версії HTTP 0.9 використовувався тільки метод GET, список запитів для версії 1.1 представлено нижче.

URI визначає шлях до запрошуваного документа.

Стартовий рядок відповіді сервера має такий формат:

HTTP/Версія КодСтану Пояснення

Тут:

Версія-пара розділених точкою арабських цифр, як у запиті.

КодСтану (англ. *Status Code*)-три арабські цифри. За кодом статусу визначено подальший вміст повідомлення та поведінка клієнта.

Пояснення (англ. *Reason Phrase*)-текстове коротке пояснення до коду відповіді для користувача. Ніяк не впливає на повідомлення і є необов'язковим.

У разі успішного виконання запиту сервер дасть відповідь рядком:

```
HTTP/1.0 200
Ok
```

Методи запитів протоколу HTTP

Метод HTTP (англ. *HTTP Method*)-послідовність з будь-яких символів, окрім керівників і роздільників, що вказує на основну операцію над ресурсом. Зазвичай метод є коротким англійським словом, записаний заголовними буквами.

Назва методу чутлива до регістра. Якщо метод серверу невідомий, від відповідає помилкою 501 (*Method not implemented*). Якщо серверу метод відомий, але він не застосований до конкретного ресурсу, то повертається сполучення з кодом 405 (*Method Not Allowed*).

Найбільш часто використовувані методи це **GET, HEAD і POST**.

Метод GET. Використовується для запиту утримуваного вказаного ресурсу.

Клієнт може передавати параметри виконання запиту в URI цільового ресурсу після символу ""?:

```
GET /path/resource?param1=value1&param2=value2
HTTP/1.1
```

Згідно зі стандартом HTTP, багаторазове повторення одного й того ж запиту GET повинно приводити до однакових результатів. Це дозволяє кешірувати відповіді на запити GET.

Особливістю цього методу є те, що передача змінних здійснюється в адресному рядку.

Цей метод раціонально використовувати, коли необхідно на сайті передавати змінні для подальшого виведення тих або інших статей.

Особливості використання

1. Передача невеликих осягів даних.
2. Передача змінних відбувається в адресному рядку й тому видно інформацію про змінні та їх значення.
3. Адреса сторінки зберігається, наприклад, в закладках, і в подальшому її можна використовувати такою, якою вона є.

Метод HEAD. Аналогічний до методу GET, за винятком того, що у відповіді сервера відсутнє тіло.

Запит HEAD зазвичай застосовується для витягання метаданих, перевірки наявності ресурсу (валідація URL) і для того щоб дізнатися, чи не змінився він з моменту останнього звернення.

Метод POST. Застосовується для передачі призначених для користувача даних заданому ресурсу. Наприклад, у блогах відвідувачі зазвичай можуть уводити свої коментарі до записів в HTML-форму, після чого вони передаються серверу методом

POST і він уміщує їх на сторінку. При цьому передаванні дані (у прикладі з блогами-текст коментаря) включаються в тіло запиту. Аналогічно за допомогою методу POST зазвичай завантажуються файли.

На відміну від методу GET, для методу POST багатократне повторення одних і тих же запитів POST може повертати різні результати (наприклад, після кожної відправки коментаря з'являтиметься одна копія цього коментаря). Повідомлення відповіді сервера на виконання методу POST не кеширується.

На відміну від методу GET, методом POST можна передавати значно більшіобсяги інформації, навіть файли. Особливістю є те, що дані передаються не в адресному рядку, а усередині тіла запиту.

Особливості використання

1. Працює повільніше, ніж GET, тому що аналізується тіло запиту.
2. Сторінку, сгенерерировану методом POST не можна зберегти у закладку.

Існують також методи: **PUT, PATCH, DELETE, TRACE, CONNECT, LINK, UNLINK**

Коди стану протоколу HTTP

Код стану протоколу HTTP-числове значення стану протоколу, яке визначається результатом обробки останнього запиту.

Нині виділено п'ять класів кодів стану:

- **1xx Informational (Інформаційний);**
- **2xx Success (Успішно).**

Повідомлення цього класу інформують про випадки успішного прийняття й обробки запиту клієнта. Залежно від статусу сервер може ще передати заголовки й тіло повідомлення;

- **3xx Redirection (Перенаправлення).**

Коди статусу класу 3xx повідомляють клієнта, що для успішного виконання операції треба провести подальший запит до іншого URI. В більшості випадків нова адреса вказується в полі *Location* заголовка. Клієнт у цьому випадку повинен, як правило, провести автоматичний перехід;

•4xx Client Error (Помилка клієнта).

Клас кодів 4xx призначений для вказівки помилок з боку клієнта. При використанні всіх методів окрім HEAD сервер повинен повернути в тілі повідомлення гіпертекстове пояснення для користувача. ***Наприклад***, помилка 404 (відсутність ресурсу);

•5xx Server Error (Помилка сервера).

Коди 5xx виділені під випадки невдалого виконання операції з вини сервера. Для всіх ситуацій, окрім використання методу HEAD, сервер повинен містити у тілі повідомлення пояснення, яке клієнт відобразить користувачеві.

Приклад діалогу за протоколом HTTP

Запит:

```
    GET /wiki/HTTP HTTP/1.1 Host: ru.wikipedia.org
    User - Agent: Mozilla/5.0 (X11; U; Linux i686; ru; rv :1.9b5) Gecko/2008050509 Firefox/3.0b5
    Accept: text/html
    Connection: close
```

Відповідь:

```
    HTTP/1.0 200 OK Server : nginx/0.6.31 Content - Language: ru Content - Type: text/html; charset=utf - 8 Content - Length: 1234 Connection: close
```

<вміст запитаної сторінки>.

Програмування на стороні клієнта

Скри́птова мова (*scripting language*, у російськомовній літературі прийнята назва **мова сценаріїв**)-мова програмування, розроблена для запису **„**сценаріїв", послідовностей операцій, які користувач може виконувати на комп'ютері. Прості скриптові мови раніше часто називали мовами пакетної обробки (*batch languages або job control languages*). Сценарії зазвичай інтерпретуються, а не компілюються (хоча часто сценарії компілюються кожного разу перед запуском).

У прикладній програмі, **сценарій (скрипт)**-це програма, яка автоматизує певне завдання, яке без сценарію користувач

робив би вручну, використовуючи інтерфейс програми. Скриптами також називають програми, написані за допомогою спеціальних скриптових мов програмування

Клієнтські мови програмування обробляються на стороні клієнта, тобто програми на клієнтській мові обробляє браузер.

Проблема в тому, що обробка скрипту залежить від браузера користувача, і користувач має повноваження настроїти свій браузер так, щоб він взагалі ігнорував написані скрипти. При цьому браузер взагалі може не підтримувати ту або іншу мову або версію мови.

Завдання: самостійно вивчити інформацію про "війни браузерів"

Також код клієнтського скрипта може подивитися кожний, хто відкриє сторінку зі скриптом.

Перевага ж клієнтської мови полягає в тому, що обробка скриптів такою мовою може виконуватися без відправлення документа на сервер.

Приклад: треба перевірити, чи правильно користувач увів e-mail; щоб це зробити користувачеві, потрібно було б відправити форму із заповненими даними, потім дочекатися, поки вона обробиться, і лише після цього одержати повідомлення про помилку (якщо вона, зрозуміло, наявна). Це неприпустимо з точки зору зручності для користувача і витрат ресурсів.

З клієнтською ж мовою програма відразу скоригує правильне заповнення форми перед відправленням, і, якщо необхідно, виведе помилку. Клієнтські мови є обмежувальними в роботі з файловою структурою і не можуть працювати на стороні сервера.

Найпоширенішим з мов клієнтських скриптів є *JavaScript*.

Серверні скрипти. Обробка скриптів серверною мовою

Коли користувач дає запит на яку-небудь сторінку (переходить на неї по посиланню, або вводить адресу в адресному рядку свого браузера), то викликана сторінка спочатку обробляється на сервері, тобто виконуються усі скрипти, пов’язані зі сторінкою, і тільки потім повертається до відвідувача у вигляді

простого HTML-документа (тобто відвідувач уже ніяк не зможе побачити код серверного скрипта).

Робота серверних скриптів залежить від серверу, на якому розташований сайт, і від того, які технології підтримуються сервером.

Серверні мови програмування відкривають перед програмістом великі функціональні можливості. Сучасні сайти частенько є надзвичайно складними програмно-інформаційними системами, які вирішують значну кількість задач бізнес-логіки та теоретично можуть дублювати функції більшості бізнес-додатків.

Серверні скрипти, як правило, взаємодіють з файловою системою і базами даних. Найчастіше використовуються СУБД MySQL, PostreSQL, MS SQL Server, Oracle.

Серед мов серверних скриптів найбільш поширеними є PHP, Perl, Asp.net.

Контрольні питання:

1.Що таке World Wide Web?
2.Для чого існують веб-технології?
3.Що URI містить у мережі Інтернет?
4.Особливості концепції роботи веб-технології з інформацією.
5.Привабливість Web-технологій як засобу передачі й надання інформації.
6.Назвати програми для перегляду веб-сторінок і роботи з інформацією у веб-інтерфейсі.
7.Сформувати поняття Web–сторінки, сайта, інтерфейсу користувача.
8.З яких елементів складається інтерфейс користувача.
9.За якою технологією взаємодіють браузер та веб-сервер ?
10. Які функції виконує сучасний браузер?
11.Назвати класифікацію популярних браузерів.
12.Опишіть структуру протоколу HTTP.
13.Які існують методи запитів протоколу HTTP? Назвати особливості їх використання.
14.Назвіть класифікації кодів стану протоколу HTTP.
15.Від чого залежить робота серверних скриптів?

Лекція 2. Алфавіт мови PHP

План:

1. Алфавіт мови програмування PHP. Інтеграція мови з мовою гіпертекстової розмітки. Базовий синтаксис PHP.
2. Змінні.
3. Коментарі.
4. Базові типи даних PHP.
5. Перетворення типів.
6. Привласнення.
7. Змінні в змінних.
8. Константи.
9. Математичні операції і константи. Вирази, оператори.

PHP є потужною мовою написання сценаріїв для Web, який продовжує розвиватися, починаючи з версії PHP 3, випущеної в 1997 році. З точки зору розробника PHP підтримує неймовірний великий діапазон Internet-технологій, що робить його на сьогодні провідною мовою сценаріїв для Web.

Перш, ніж писати PHP-сценарії, важливо взагалі зрозуміти, як виконується розробка PHP-сценаріїв. Щоб розібратися з цим, ви повинні спочатку отримати уявлення про взаємодію між клієнтом (наприклад, Web-браузером) і Web-сервером. Коли клієнт запрошує документ з Web-сервера, то, як правило, Web-сервер витягує документ і відправляє клієнтові. В більшості випадку цей документ є HTML-файл, графічний образ або щось подібне. Клієнт обробляє його і відображує у вікні браузеру. На відміну від цього, при використанні PHP-сценарію додається ще одна проміжна стадія-*попередня обробка*. На цій стадії інтерпретатор PHP обробляє запит PHP-сценарію, виконує код, що міститься у ньому, і посилає виведення назад Web-серверу, щоб той відправив його клієнтові. Незважаючи на те, що головна мета PHP-сценарію полягає в генеруванні HTML-змісту, під час його виконання може відбуватися усе, що завгідно-від доступу до бази даних до відправки поштових повідомлень. Одна істотна відмінність цієї мови програмування полягає в тому, що увесь код виконується на сервері.

Незважаючи на те, що це найбільш поширений спосіб використання PHP, він також може використовуватися на клієнтській стороні для розробки застосувань, як у середовищі Windows, так і в Unix.

Базовий синтаксис PHP

PHP-сценарії пишуться у вигляді блоків коду. Ці блоки за необхідності можуть бути вбудовано в HTML, і зазвичай визначено за допомогою рядка `<?php` на початку і `?>`-у кінці. Усе, що поза цими ідентифікаторами блоку, інтерпретатор PHP ігнорує і передає назад Web-серверу для відображення на стороні клієнта.

У лістингу 2.1 показано приклад простого PHP-сценарію, що виводить вітальний рядок.

Лістинг 2.1. Простий сценарій на PHP

```
<HTML>
<head>
<TITLE>Перший PHP-сценарій</TITLE>
</HEAD>
<BODY bgcolor="aqua" >
<h1 align="center">Починаємо роботу з PHP</h1>
<?php
echo "Ласкаво просимо, користувач"!;
?>
</BODY>
</HTML>.
```

Найчастіше використовуються дескриптори <?php і ?>, як роздільники блоків коду також можуть застосовуватися такі конструкції, коли включена відповідна директива конфігурації в php. ini.:	
<? ... ?>	Скорочена версія <?php і ?>.
<% ... %>	Стиль ASP.
<SCRIPT LANGUAGE="PHP"> </SCRIPT>	Синтаксис, сумісний з редакторами HTML.

Змінні

Змінна є іменованою областю пам'яті, що містить дані, з якими можна виконувати операції під час виконання програми.

У PHP імена змінних завжди починаються з символу $ і містять довільну комбінацію символів, за умови, що перший символ потім $ буде буквою або знаком підкреслення. До числа допустимих символів входять заголовні і прописні латинські букви, а також символи з ASCII-кодами в діапазоні від 127 до 255 (символи, не використовувані в американському англійському). Змінні в PHP можуть бути визначені або привласненням їм значення, або за допомогою оператора var.

Змінні в PHP, як і в мові Perl, не вимагають спеціального оголошення. Замість цього змінна оголошується при першому її використанні в програмі, тип змінної побічно визначається за типом тих, що зберігаються в ній даних.

Приклад:

```
$sent = "This is a sentence".; // $sent
```

інтерпретується як рядок

```
$price = 42.99: // $price
```

інтерпретується як дійсне число.

Змінні можуть оголошуватися в будь-якій точці сценарію PHP, проте від розташування оголошення залежить те, звідки можна звертатися до цієї змінної**.**

Зона видимості змінних

Зона видимості (*scope*) визначається як область доступності змінної в тій програмі, у якій вона була оголошена. Залежно від зони видимості змінні PHP діляться на чотири типи:

- локальні змінні;
- параметри функцій;
- глобальні змінні;
- статичні змінні.

Локальні змінні

Змінна, оголошена всередині функції, вважається локальною-на неї можна посилатися тільки в цій функції. При будь-якому привласненні поза функцією використовуватиметься абсолютно інша змінна, яка не має нічого спільного (окрім імені) зі змінною, оголошеною усередині функції. При виході з функції, у якій була оголошена локальна змінна, ця змінна і її значення знищуються.

Основна перевага локальних змінних-відсутність непередбачених побічних ефектів, пов'язаних з випадковою або навмисною модифікацією глобальної змінної.

Параметри функцій

У PHP параметри, передавані функції при виклику мають бути оголошені в заголовку функції. Хоча параметрам привласнюються аргументи, передані ззовні, після виходу з функції вони стають недоступними.

Параметри оголошуються в круглих дужках після імені функції. Оголошення параметрів практично не відрізняється від оголошення типової змінної :

```
<?php
function mult ($val)
{$val = $val * 10; // Функція множить передане значення на 10
return $val;};
$b=mult(100);
echo "результат = $b;";
?>
```

Хоча ви можете звертатися до параметрів у тій функції, у якій вони були оголошені, і виконувати з ними необхідні операції, після завершення функції параметри знищуються.

Глобальні змінні

Глобальні змінні, на відміну від локальних, доступні в будь-якій точці програми.

Щоб змінити значення глобальної змінної, необхідно спеціально оголосити її як глобальну у відповідній функції. Для цього перед ім'ям змінної ставиться ключове слово GLOBAL.

Приклад:

```
<?
$somevar = 15;
function addit()
{ GLOBAL $somevar;
$somevar++;
print "Somevar is $somevar"; }
addit();
?>
```

Буде виведено значення $somevar, рівне 16. Без рядка GLOBAL *$somevar*; змінній буде присвоєно значення 1, оскільки ця змінна вважатиметься локальною стосовано до функції *addit ( )*.

Альтернативний спосіб оголошення глобальних змінних пов'язаний з використанням масиву PHP $GLOBALS( ).

Приклад:

```
<?php
$somevar = 15;
function addit()
{ $GLOBALS ["somevar"];
$somevar++;
return $somevar;};
$rez=addit();
print "Addit is $rez <br>";
print "Значення змінної somevar рівне $somevar";
?>
```

Виводить значення:

```
Addit is 1
```

Значення змінної somevar дорівнює 15.

Глобальні змінні дуже зручні, але при їх використанні потрібна помірність.

Статичні змінні

Останній тип видимості змінних називається статичним. На відміну від змінних, оголошених параметрами і знищуваних при виході з функції, статична змінна зберігає своє значення при повторному виклику. Для оголошення статичної змінної перед її ім'ям ставиться ключове слово STATIC :

STATIC $somevar;

Приклад:

```
<?
function keep_track()
{ STATIC $count = 0;
$count++;
print $count;
print "<br>";}
keep_track(); keep_track(); keep_track();
?>
```

Змінна *$count* є статичною, тому при кожному виклику функції зберігатиметься її попереднє значення і результат буде таким:

1
2
3

Незважаючи на те, що необхідність в явному знищенні змінних з метою звільнення ресурсів відсутня, іноді вимагається примусово знищити змінну. Для цих цілей передбачено функцію *unset ()*. Ця функція може бути використана тільки з допустимими змінними, зокрема й елементами масивів. У лістингу 1.2 демонструється використання функції *unset ()* для знищення існуючої змінної PHP.

Приклад. Використання функції *unset ()*

```
<?php
$myvar = "Рядок";
unset($myvar); // Знищення змінної
?>
```

З точки зору типізації змінних PHP можна класифікувати як мову, що вільно типізується. Коли змінній привласнюється значення, PHP трактує її відповідним чином залежно від контексту, у якому вона застосовується. У PHP існують три базові типи змінних (цілі, строкові, дійсні з плаваючою точкою) і два складні типи (об'єкти й масиви).

Коментарі

У PHP усе, що знаходиться між /* і */, трактується як коментар, використовуваний для пояснень у тілі сценарію, й ігнорується інтерпретатором. Для однорядкових коментарів

можуть застосовуватися або //, або #, що поміщає в коментар залишок рядка.

```
<?php
$var = "foo"; // це ігнорується
$var = "bar"; # це теж
?>
```

Базові типи даних PHP

Цілі числа-фундаментальний числовий тип PHP, що представляє значення зі знаком. На практиці PHP сприймає цілі значення з використанням трьох математичних представлень: десяткові (на базі 10), вісімкові (на базі 8) і шістнадцятирічні (на базі 16). У більшості ситуацій PHP-сценарії пишуться в десятковій нотації.

Приклад. Зберігання цілих значень в PHP

```
<?php
$my_int = 50;  /* Стандартна десяткова нотація */
$my_int = 062; /*То ж число у вісімковій нотації
(починається з букви 'Oz') */
$my_int = 0x32;/* Шістнадцятирична нотація */
?>
```

Коли PHP працює з **дробними** числами, він представляє значення у вигляді типу даних з плаваючою точкою.

І останній базовий тип даних-**строковий**. Існує два типи рядків : розібрані *(parsed*) і нерозібрані (*unparsed*).

Розібрані рядки визначаються з використанням подвійних лапок і розбираються PHP, тоді як нерозібрані рядки представляються в одинарних лапках і використовуються такими, як вони є.

Коли рядок визначений у подвійних лапках, то будь-яке посилання на змінну всередині такого рядка автоматично замінюється її значенням, тоді як для нерозібраних рядків така підстановка не виконується.

У лістингу 2.4. представлено приклад обох типів рядків.

Лістинг 2.4. Розібрані і нерозібрані рядки в PHP

```
<?php
$my_int = 50;
$string_one   =   "Значення   змінної   рівне
```

```
$my_int<BR>";
$string_two    =    'Значення    змінної    рівне
$my_int<BR>';
  echo string_one;
  echo $string_two;
  ?>
```

Висновок буде таким:

```
Значення змінної рівне 50
Значення змінної рівне my_int
```

Разом з можливістю замінювати посилання змінних, розібрані рядки дозволяють працювати з тим, що називається **скасованими** (*escaped*), або **провідними**, символами. Цей спеціальний формат використовується для представлення символів, які зазвичай було б важко або взагалі неможливо включити в рядки за допомогою стандартної клавіатури. Перелік доступних у PHP првідних символів представлено в таблиці. 2.1.

Таблиця 2.1. **Символи php, що керують**

Рядок символів, що керують	***Результуючий символ***
\n	Символ переведення рядка
\r	Символ повернення каретки
\t	Символ горизонтальної табуляції
\\	Символ зворотного слеша
\$	Символ $
\'	Символ одинарної лапки
\"	Символ подвійної лапки
\###	ASCII-символ (вісімковий)
\x##	ASCII-символ (шістнадцятиричний)

Перемикання типів

Іноді буває зручно використовувати змінні засоби, не передбачені при їх створенні. Тип змінних PHP може змінюватися і без використання механізму явного перетворення. Цей процес, незалежно від того, чи виконується він прямо або побічно, називається перемиканням (*juggling*) типів.

Припустимо, ви підсумовуєте дві величини-рядок і ціле число. Результат залежить від вмісту рядка. ***Наприклад***, при пі-

дсумовуванні цілого числа зі строковим представленням числа буде отримано ціле число:

```
$variablel = 1;
$variable2 = "1";
$variable3 = $variablel + $variable2;
//$variable3 привласнюється 4.
```

Інший приклад перемикання типів-підсумовування цілого числа з дійсним. При цьому ціле число перетвориться до дійсного типу, щоб уникнути втрати точності:

Слід згадати про певні маловідомі особливості перемикання типів. Лексичний аналізатор PHP визначає тип по початку рядка. Припустимо, ми привели змінну *$variable2* до виду "*There are 100 bottles of beer on the wall"*. Оскільки алфавітні символи важко інтерпретувати як ціле число, рядок інтерпретується як 0, і змінною *$variable3* привласнюється 5.

Приклад:

```
$variablel = 5;
$variable2 = "100 bottles of beer on the wall";
$variable3 = $variable1 + $variable2;
// $variable3 привласнюється 105
```

У наслідку змінної *$variable3* привласнюється значення 105.

Хоча в більшості випадків перемикання типів забезпечує бажаний результат, існує спосіб явного приведення змінних до конкретного типу.

Перетворення типів

Явне приведення змінної до типу називається перетворенням (*casting*) типу. Зміна типу може бути як тимчасовою, одноразовою, так і постійною.

Щоб тимчасово привести змінну до іншого типу, досить скористатися оператором перетворення типу-зазначити потрібний тип перед ім'ям змінної в круглих дужках (таблиця. 2.2).

Таблиця 2.2. **Оператори перетворення типу змінних**

Оператор перетворення типу	Новий тип
(int) чи (integer)	Ціле число
(real), (double) або (float)	Дійсне число

(string)	Рядок
(array)	Масив
(object)	Об’єкт

Приклад перетворення типів :

```
$variable1= 13; // $variable1 привласнюється ціле число 13
$variable2 = (double) $variable1; // $variable2 привласнюється 13.0
```

Перетворення дійсного типу до цілого завжди супроводжується округленням.

Рядок або змінну іншого типу також можна перетворити в елемент масиву. У цьому випадку перетворена змінна стає першим елементом масиву.

Нарешті, будь-який тип даних можна перетворити на об’єкт. Змінна стає атрибутом об’єкта, і їй привласнюється ім’я *scalar* :

```
$model = "Toyota";
$new_obj = (object) $model;
```

Посилання на початкове строкове значення виглядає так:

```
print $new_obj ->scalar;
```

Привласнення

Існують 2 види привласнення: за значенням і за посиланням.

Привласнення за значенням

Привласнення за значенням-найпоширеніший спосіб привласнення, при якому значення просто заноситься в область пам’яті, представлену ім’ям змінної.

Приклади привласнення за значенням:

```
$vehicle = "car";
$amount =10.23;
```

Привласнення за значенням також може виконуватися в результаті виконання команди *return* у функціях:

```
function simple ()
{ return 5;}
```

```
$znach = simple();
```

Функція *simple()* повертає значення 5, яке привласнюється деякій змінній.

Привласнення за посиланням

Інший спосіб полягає в привласненні змінного посилання на область пам'яті, займану іншою змінною. Замість конкретного значення змінна-приймач зв'язується з покажчиком (чи посиланням) на область пам'яті, тому фактичне копіювання не виконується.

Щоб присвоїти значення за посиланням, зазначте перед ім'ям змінної-джерела символ & (амперсанд) :

```
$dessert = " cookies ";
$dessert2 = &$dessert;
$dessert2 = "cookies";
print "$dessert2 <br>"; // Виводиться рядок cookies
print $dessert; // Знову виводиться рядок cookies
```

З наведеного фрагменту видно, що після зв'язування змінною *$dessert2* з посиланням на область пам'яті, займану змінною *$dessert*, будь-які зміни *$dessert2* приводять до автоматичної модифікації *$dessert* (і всіх інших змінних, що посилаються на цю ж область пам'яті).

Змінні у змінних

В окремих ситуаціях буває зручно використовувати змінні, вміст яких може динамічно інтерпретуватися як ім'я іншої змінної. Розглянемо типовий випадок привласнення:

```
$recipe = "spaghetti";
```

Рядок "*spaghetti*" можна інтерпретувати як ім'я змінної-для цього в команді привласнення перед ім'ям початкової змінної ставиться другий знак $.

```
$recipe = "& meatballs";
```

Ця команда привласнює рядок *"& meatballs*" змінної з ім'ям "*spaghetti"*. Отже, наступні дві команди виводять однакові результати:

```
print $recipe $spaghetti;
print $recipe $($recipe);
```

В обох випадках буде виведено рядок "*spaghetti & meatballs*".

Стандартні змінні

Щоб отримати повний перелік змінних web-сервера, оточення і PHP, визначених для вашої конфігурації системи, досить виконати такий фрагмент:

```
while (list($var,$value) = each($GLOBALS)) :
echo "<BR>$var => $value";
endwhile;
```

В результаті виводиться перелік стандартних глобальних змінних.

Стандартні змінні містять різноманітні відомості.

Наприклад, наступна команда виводить IP-адрес користувача :

```
print "Hi! Your IP address is: $REMOTE_ADDR";
```

IP-адрес виводиться в числовій формі (наприклад, 208.247.106.187).

Крім того, стандартні змінні можуть використовуватися для збору інформації про браузер і операційну систему користувача. Команда

```
print "Your browser is: $HTTP_USER_AGENT";
```

повертає інформацію наступного виду :

```
Your browser is: Mozina/4.0 (compatible: MSIE 5.0; Windows 98: CNETHomeBuild051099).
```

Інформація про браузер і операційну систему, у якій він працює, може згодитися при побудові сторінок, розрахованих на специфічні формати конкретних браузерів.

Для роботи з масивами стандартних змінних необхідно включити директиву *track_vars* до файлу php.ini. У PHP версії 4.0.3 директива *track_vars* включена постійно.

Константи

Константою називається іменована величина, яка не змінюється в процесі виконання програми. Константи особливо

зручні при роботі зі свідомо постійними величинами-наприклад, числом π (3,141592).

У PHP константи визначаються функцією *define()*. Після того, як константа буде визначена, ви не зможете її змінити (чи перевизначити) в цій програмі.

Наприклад, визначення числа пі в сценарії PHP може виглядати так:

```
define("'PI", "3.141592");
```

Певну константу можна використовувати в програмі:

```
print "The value of pi is". PI".<br>";
$pi2 = 2 * PI:
print "Pi doubled equals $pi2".
```

Результат роботи цього фрагмента буде таким:

```
The value of pi is 3.141592.
Pi doubled equals 6.283184.
```

В іменах констант не зазначено знак долара. Константу неможливо модифікувати, якщо константа використовується в обчисленнях, то результат доводиться зберігати в іншій змінній.

Вирази, оператори

Вирази

Вираження описує певну дію, що виконується в програмі. Кожне вираження полягає, принаймні, з одного операнда й одного або кількох операторів.

Операнди

Операнд є деякою величиною, що обробляється в програмі. Операнди можуть належати до будь-якого типу даних, представленого в PHP.

Приклади *операндів :*

```
$a++; // $a – операнд.
$sum = $val1 + $val2; // $sum. $val1 і $val2 – операнди.
```

Оператори

Оператор є символічним позначенням певної дії, що виконується з операндами у вираженні. PHP виконує автоматичне перетворення типів на підставі типу оператора, що об'єднує два операнди, - в інших мовах програмування це відбувається не завжди.

Операції

Таблиця 2.3. **Арифметичні операції**

Приклад	Назва	Результат
$a + $b	Складання	Сума $a і $b
$a - $b	Віднімання	Різниця $a і $b
$a * $b	Множення	Твір $a і $b
$a / $b	Ділення	Частка від ділення $a на $b
$a % $b	Modulus	Цілочисельний залишок від ділення $a на $b

Операція ділення ("/") завжди повертає число з плаваючою точкою.

Операції привласнення, бітові, порівняння.

Базова операція привласнення це "=".

Значенням вираження привласнення є присвоєне значення. Тобто значення "$a = 3" дорівнює 3.

```
$a = ($b = 4) + 5; // $a зараз дорівнює 9, а $b має значення 4
```

Окрім базової операції привласнення, є "комбіновані операції" для усіх бінарних, арифметичних і строкових операцій, які дозволяють використовувати значення у вираженні, а потім встановити його значення в результат цього виразу.

Наприклад:

```
$a = 3;
$a += 5; // встановлює в $a 8,
аналогічно a = $a + 5;
```

Таблиця 2.4. Бітові операції.

Приклад	Ім'я	Результат
$a & $b	And	Встановлюються біти, які встановлені і в $a, і в $b

$a | $b	Or	Встановлюються біти, які встановлені в $a або в $b
$a ^ $b	Xor	Встановлюються біти, які встановлені в $a або $b, але не в обох
~ $a	Not	Встановлюються біти, які в $a не встановлені, і навпаки
$a << $b	Зрушення вліво	Зрушує біти змінної $a на $b кроків вліво (кожен крок/зміщення означає "помножити на 2")
$a >> $b	Зрушення управо	Зрушує біти змінної $a на $b кроків управо (кожен крок/зміщення означає "розділити на 2")

Таблиця 2.5. Операції порівняння

Приклад	Назва	Результат
$a = = $b	рівно	**TRUE**, якщо $a рівне $b
$a = = = $b	ідентично	**TRUE**, якщо $a рівне $b і вони одного типу. (тільки у PHP 4)
$a != $b	не рівно	**TRUE**, якщо $a не рівне $b
$a <> $b	не рівно	**TRUE**, якщо $a не рівне $b
$a != = $b	не ідентично	**TRUE**, якщо $a не рівне $b або вони різних типів. (тільки у PHP 4)
$a < $b	менше	**TRUE,** якщо $a строго менше $b
$a > $b	більше	**TRUE,** якщо $a строго більше $b
$a <= $b	менше або рівно	**TRUE,** якщо $a менше або рівне $b
$a >= $b	більше або рівно	**TRUE**, якщо $a більше або рівне $b

Операторы порівняння призначені для роботи тільки з числовими значеннями. У PHP існують стандартні функції для порівняння строкових величин.

Операції управління помилками

PHP підтримує одну операцію управління помилками: знак (@). Якщо він уставлений як префікс вираження PHP, будь-які помилки, які можуть генеруватися цим виразом, пригнічуються.

Якщо включено *track_errors*, будь-які повідомлення про помилки, генеровані цим виразом, зберігатимуться в глобальній змінній *$php_errormsg*. Ця змінна перезаписуватиметься при виникненні кожної нової помилки, тому перевіряйте її відразу, якщо необхідно.

```
<?php  /* Передбачається файлова помилка */
```

```
$my_file = @file ('non_existent_file') or die
("Failed opening file: error was '$php_errormsg'");
// працює для усіх виразів, а не тільки у функціях
$value = @$cache[$key];
//повідомлення не буде, якщо індекс $key не існує.
  ?>
```

Примітка: @-операция працює тільки у виразах. Основне правило: якщо ви можете набути значення чого-небудь, ви можете поставити як префікс операцію @. ***Наприклад,*** ви можете поставити її до змінних, функцій і викликів **include(),** констант і тощо. Ви не можете підставити її до визначення функції або класу або структур управління, таким як **if** і **foreach**, і т. ін..

Оператори привласнення, строкові оператори.

Операторы привласнення задають нове значення змінної. У простому варіанті оператор привласнення обмежується зміною величини, в інших варіантах (званих скороченими операторами привласнення) перед привласненням виконується певна операція. Приклади операторів наведені в таблицю 2.6.

Таблиця 2.6. Оператори привласнення

Приклад	**Назва**	**Результат**
$a = 5	Привласнення	Змінна $a рівна 5
$a += 5	Складання з привласненням	Змінна $a дорівнює сумі $a і 5
$a *= 5	Множення з привласненням	Змінна $a дорівнює твору $a і 5
$a/=5	Ділення з привласненням	Змінна $a дорівнює частці відділення $a на 5
$a .= 5	Конкатенація з привласненням	Змінна $a дорівнює конкатенації $a і 5

Таблиця 2.7. Строкові оператори

Приклад	**Назва**	**Результат**
$a = "abc"".def"	Конкатенація	Змінній $a привласнюється результат конкатенації $a і $b
$a - "ghijkl"	Конкатенація з привласненням	Змінній $a привласнюється результат конкатенації її поточного значення з рядком "ghijkl"

Оператори інкремента і декремента

Операторы інкремента (++) і декремента (--), приведені в таблиці. 2.8, забезпечують укорочений запис для зміни значення змінної на 1.

Таблиця 2.8. Оператори інкремента і декремента

Приклад	**Назва**	**Результат**
++$a, $a++	Інкремент	Змінна $a збільшується на 1
--$a, $a--	Декремент	Змінна $a зменшується на 1

Логічні оператори

Логічні оператори дозволяють управляти порядком виконання команд в програмі і часто використовуються в конструкціях (таких, як умовна команда **if**, а також цикли **for** і **while**), що управляють.

Таблиця 1.9. Логічні оператори

Приклад	**Назва**	**Результат**
$a && $b	Кон'юнкція	Істина, якщо істинні обидва операнди,
$a AND $b	Кон'юнкція	Істина, якщо істинні обидва операнди
$a || $b	Диз'юнкція	Істина, якщо істинний хоч би один з операндів
$a OR $b	Диз'юнкція	Істина, якщо істинний хоч би один з операндів
!$a	Заперечення	Істина, якщо значення $a помилково
NOT $a	Заперечення	Істина, якщо значення $a помилково
$a XOR $b	Диз'юнкція, що виключає	Істина, якщо істинний тільки один з операндів

Логічні оператори часто використовуються для перевірки результату виклику функцій :

```
file_exists("filename.txt") OR print "File does not exist"!;
```

Маніпуляції зі змінними

PHP підтримує усі базові математичні операції, як і будь-яку іншу мову програмування, зокрема і складання та множення, а також широкий діапазон тригонометричних і логарифмічних функцій. Окрім математичних операцій PHP підтримує великий обсяг функцій з рядками.

PHP підтримує усі загальноприйняті математичні стандарти пріоритетів операцій, угрупування і інше - як для цілих, так і для дійсних чисел з плаваючою точкою.

Важливо знати, як обробляються числа з плаваючою точкою у PHP, коли вони конвертуються в цілі. *Наприклад*, значення 0.999999 при перетворенні в ціле може перетворитися на 0, тоді як у інших системах воно може бути зведене до 1, як завжди й очікується. Така різниця в поведінці залежить від системи, на якій виконується PHP, а не від самого PHP.

У PHP також підтримуються скорочені форми запису операцій.

Приклад. Скорочений запис математичних операцій у PHP

```
<?php
$answer = 5;//Привласнення початкового значення
$answer += 2;// Еквівалент $answer = $answer + 2;
?>
```

Для ще більшого спрощення можна використовувати операції інкремента й декремента наступного виду.

Разом з простою математикою PHP також підтримує тригонометричні й логарифмічні операції для складних обчислень, ***наприклад***:

```
<?php
$cos = cos(2 * M_PI); /* косинус 2*PI дорівнює 1 */
?>
```

Математичні операції і константи

Зумовлені константи

Перераховані нижче константи завжди доступні як частина ядра PHP.

Таблиця 2.10. **Math constants**

Constant	Value	Description
M_PI	3.14159265358979323846	Pi
M_E	2.718281828459045235	e
M_LOG2E	1.4426950408889634074	log_2 e
M_LOG10E	0.43429448190325182765	log_10 e
M_LN2	0.69314718055994530942	log_e 2
M_LN10	2.30258509299404568402	log_e 10
M_PI_2	1.57079632679489661923	pi/2
M_PI_4	0.78539816339744830962	pi/4
M_1_PI	0.31830988618379067154	1/pi
M_2_PI	0.63661977236758134308	2/pi
M_SQRTPI	1.77245385090551602729	sqrt(pi)[4.0.2]
M_2_SQRTPI	1.12837916709551257390	2/sqrt(pi)

Constant	Value	Description
M_SQRT2	1.41421356237309504880	sqrt(2)
M_SQRT3	1.73205080756887729352	sqrt(3)[4.0.2]
M_SQRT1_2	0.70710678118654752440	1/sqrt(2)
M_LNPI	1.14472988584940017414	log_e(pi)[40.2]
M_EULER	0.57721566490153286061	Euler constant [4.0.2]

Таблиця 2.11. **Math functions**

abs	Absolute value
acos	Arc cosine
acosh	Inverse hyperbolic cosine
asin	Arc sine
asinh	Inverse hyperbolic sine
atan2	Arc tangent of two variables
atan	Arc tangent
atanh	Inverse hyperbolic tangent
base_convert	Convert a number between arbitrary bases
bindec	Binary to decimal
ceil	Round fractions up
cos	Cosine
cosh	Hyperbolic cosine
decbin	Decimal to binary
dechex	Decimal to hexadecimal
decoct	Decimal to octal
deg2rad	Converts the number in degrees to the radian equivalent
exp	Calculates the exponent of e (the Neperian or Natural logarithm base)
expm1	Returns exp(number) 1, computed in a way that is accurate even when the value of number is close to zero
floor	Round fractions down
fmod	Returns the floating point remainder (modulo) of the division of the arguments
getrandmax	Show largest possible random value
hexdec	Hexadecimal to decimal
hypot	Returns sqrt(num1*num1 + num2*num2)
is_finite	Finds whether a value is a legal finite number
is_infinite	Finds whether a value is infinite
is_nan	Finds whether a value is not a number
lcg_value	Combined linear congruential generator
log10	Base
10 logarithm	
log1p	Returns log(1 + number), computed in a way that is accurate even when the value of number is close to zero
log	Natural logarithm
max	Find highest value
min	Find lowest value
mt_getrandmax	Show largest possible random value

mt_rand	Generate a better random value
mt_srand	Seed the better random number generator
octdec	Octal to decimal
pi	Get value of pi
pow	Exponential expression
rad2deg	Converts the radian number to the equivalent number in degrees
rand	Generate a random integer
round	Rounds a float
sin	Sine
sinh	Hyperbolic sine
sqrt	Square root
srand	Seed the random number generator
tan	Tangent
tanh	Hyperbolic tangent

Контрольні питання:

1. У чому полягає головна мета PHP-сценарію, під час його виконання?
2. Як виконується розробка PHP-сценаріїв? У якому вигляді пишуться PHP-сценарії?
3. За допомогою якого дескриптора можуть бути вбудовані PHP-сценарії в HTML?
4. З якого символу завжди починаються імена змінних у PHP?
5. Які символи належать до числа допустимих змінних у PHP?
6. Назвіть чотири типи змінних PHP від зони видимості.
7. Яка змінна вважається локальною? Основні переваги локальних змінних.
8. Як оголошуються параметри у PHP?
9. Яка змінна вважається глобальною? За допомогою якого ключового слова оголошується глобальна змінна?
10. Яка змінна вважається статичною? За допомогою якого ключового слова оголошується статична змінна?
11. Для чого застосовуються коментарі?
12. Які існують базові типи даних у PHP?
13. У якій нотації пишуться ситуації PHP–сценаріїв?
14. Назвіть перелік доступних у PHP символів, що керують.
15. Яке буде отримано число, при підсумовуванні цілого числа зі строковим представленням числа.
16. Яке приведення змінної до типу називається перетворенням (casting) типу?
17. Урезультаті виконання якої команди може виконуватися Привласнення за значенням?

18. Який спосіб полягає у привласненні змінного посилання на область пам’яті, займану іншою змінною?
19. У яких ситуаціях буває зручно використовувати змінні, вміст яких може динамічно інтерпретуватися як ім’я іншої змінної?
20. У результаті чого виводиться список стандартних глобальних змінних?
21. Що називається константою? Якою функцією визначаються у PHP?
22. Навести приклади операндів та операторів у PHP.
23. Назвати арифметичні операції.
24. Операції привласнення, бітові, порівняння.
25. Операції управління помилками, привласненя, строкові оператори.
26. Навести приклади операторів інкремента й декремента.
27. Які оператори дозволяють управляти порядком виконання команд у програмі і часто використовуються в конструкціях PHP-сценаріїв?
28. Як обробляються числа з плаваючою точкою у PHP?

Лекція 3. Структури, що керують. Функції.

План:

1. Вивчення основних кострукций мови.
2. Умовна операція.
3. Цикли.
4. Вбудовування структур, що керують.
5. Функції, визначувані користувачем.
6. Час життя змінної.
7. Рекурсія.

Логічні структури, що керують

Структури, що керують,–це засоби, які дозволяють управляти поведінкою програм. Структури, що управляють, дозволяють вказувати умови, при яких повинен виконуватися той або інший фрагмент коду, що зазвичай базуються на поточному стані сценарію. Часто вони навіть можуть транслюватися із звичайних тверджень природною мовою.

Умовний оператор

Оператор **if** має таку загальну форму:

if(умова){Код, якщо умова істинна} [**else**{Код, якщо умова помилкова}]

Умова–це будь-яке вираження, що повертає булеве значення.

Коли описується загальний синтаксис функції, квадратні дужки навколо якої-небудь його частини (такі, як навколо частини **else** наведеної вище конструкції) слугують для позначення необов'язкової частини, яка може бути пропущена при практичному використанні.

Однорядкова версія оператора **if** також зустрічається в такому вигляді:

If (умова){Код, якщо умова істинна}

Оператор **if** в PHP–найбільш фундаментальна структура, що управляє, призначена для виконання того, що називається блоком коду, у тому й тільки у тому випадку, коли умовне вираження повертає булеве значення *„істина”*.

Незважаючи дивлячись на те, що в загальному випадку умовна частина оператора **if** має дорівнювати зумовленим значенням true або false, цілі значення більше нуля також трактуючи як true, тоді як 0 трактується false. Проте, настійно рекомендується використовувати умовні вирази, що повертають булеві значення.

Операції порівняння

Приклад операції *Дія*

`$foo == $bartrue,` якщо `$foo` рівне `$bar.`

`$foo === $bartrue,` якщо `$foo` рівне `$bar` і обоє змінних відносяться до одного типу.

`$foo >= $bartrue,` якщо `$foo` більше або рівно `$bar.`

Лістинг 3.1. Використання операцій порівняння в операторові **if:**

```
<?php
$usl = 14;
if ($usl < 15){echo "Змінна задовольняє умові<BR> ";} else {echo " змінна більша<BR>";}
?>
```

Умови можуть бути вкладеними.

Лістинг 3.1. Вкладені умови

```
<?php
if ($value > 0){if ($value <= 10){echo 'Значення змінної $value знаходиться між 1 і 10.';} else {if ($value <= 20){echo 'Значення змінної $value знаходиться між 1 і 20.';} else {echo 'Значення змінної $value більше 20.';}}
?>
```

До блоку коду може бути включено все що завгодно, зокрема й інший блок. Немає обмежень на "глибину" вкладеності, принаймні хорошою практикою є обмеження міри вкладеності блоків для збереження читабельності коду.

Хоча це і працює, існує кращий спосіб. Коли потребує перевірка великої кількості умов, у PHP існують логічні операції для комбінації безлічі умов у єдине булеве значення.

Умовна частина пропозиції є вираженням, що повертає при обчисленні булеве значення. Тобто для створення складеного оператора **if** усе, що знадобиться-це набір булевих операцій. Ці операції, звані також логічними операціями, перераховані в таблиці. 3.1.

Таблиця 3.1. **Логічні операції PHP**

Операція	**Дія**
$foo and $bar	**true**, якщо $foo і $bar істинні.
$foo or $bar	**true**, якщо $foo або $bar істинні.
$foo xor $bar	**true**, якщо $foo або $bar істинно (але не обоє відразу).
!$foo	**true**, якщо $f оо помилково.
$foo && $bar	**true**, якщо $foo і $bar істинні.
$foo || $bar	**true**, якщо $foo або $bar істинні.

Незважаючи на те, що вони виглядають ідентичними, операції **and** та **or** – не те ж саме, що і **&** та **||**. У PHP операції **and** та **or** виконуються до операцій **&&** та **||**. Через це настійно рекомендовано використовувати дужки.

Приклад. Визначення стану води залежно від температури:

Лістинг 3.2. Складені оператори if

```
<?php
$gidkost = 'вода'; $temp = 100;
if ($gidkost) || $temp<0){ echo "твердий стан
<BR>"; }
 else { if ($gidkost) || $temp<0){echo "ріди-
на <BR>"; } else{ echo "газ <BR>"; } ?>
```

Принаймні це вже відмінно, існує можливість ще удосконалити частину логіки **else**.

У ситуаціях, коли ви використовуєте умови для визначення значення змінної, може застосовувано такий синтаксис: `$variable = (умова) ? /* істина */ : /* брехня */;$variable` буде присвоєне значення з першого сегмента, якщо умова істинна, і з другого – якщо помилка.

У подібних ситуаціях використаний код може бути в певною мірою спрощений за рахунок застосування пропозиції **elseif**. Ця пропозиція замінює **else** в умові **if**, як показано нижче:

`if(умова) {/* Блок коду, що підлягає виконанню, якщо умова істинна */ 1 elseif (умова) {/* Блок коду, що підлягає виконанню, якщо перша умова помилкова, а друга істинна */ } else {/* Блок коду, що підлягає виконанню, якщо обидві умови помилкові */ }.`

Слід зазначити, що таким чином можна сполучати разом стільки конструкцій **elseif**, скільки знадобиться, проте ефективнішим і читабельним код буде при використанні логічних виразів. Цей спосіб повинен використовуватися тільки тоді коли існує безліч складних умов, які потрібно перевіряти при виконанні сценарію.

При нагоді роботи з деякими фіксованими значеннями використовують конструкцію, що управляє, – оператор **switch**.

Як і усе, окрім найбільш фундаментальних конструкцій, що управляють, оператор **switch** – це спрощений спосіб виконання завдань, які можна виконати за допомогою оператора **if**. Призначення **switch** – дозволити розробникові привласнювати блок коду кожному з безлічі різних можливих випадків, які може приймати змінна, що керує. Загальна форма оператора switch виглядає таким чином:

```
switch($variable)
{
[case <константа>:] /* код, що виконується, коли $variable дорівнює 1 */
[break;]
[case <константа>:] /* код, що виконується, коли $variable дорівнює 2 */
[break;]
.....інші випадки [default :]
/*/*код, що виконується, якщо не було збігу ні з одним із випадків*/}
```

Константи **case** не обмежені цілими величинами, як в інших мовах, подібних С. У PHP можуть використовуватися будь-які значення констант, зокрема й рядки та числа з плаваючою точкою.

При використанні операторові **switch** представлено єдину змінну, значення якої порівняно з тими, які вказано в кожній індивідуальній пропозиції **case**. Фактично оператор switch схожий на серію операторів **if.**

Слово **break** у кінці кожного блоку є не обов'язковим. Якщо воно не вказане, PHP продовжить обробку сценарію з поточного місця, виконуючи кожен блок **case** до тих пір, поки не зустріне **break** або не досягне кінця оператора **switch**.

Приклад:

```
<?php
switch ($a)
{
case 1: echo 'Перший випадок<BR>';
case 2: echo 'Другий випадок<BR>'; break;
default:
echo 'Випадок за умовчанням';
}
?>
```

Якщо змінна **$a** має значення 1, будуть виконані й перший і другий блоки коду, оскільки **break** зустрічається тільки в кінці **case 2**.

Якщо змінна **$a** має значення 2, буде виконаний тільки другий блок коду.

Умовна операція

Умовною операцією є операція **"?:"** (тернарна), яка оперує так само, як у мові С:

```
((expr1) ? (expr2) : (expr3);
```

Цей вираз обчислюється в *expr2*, якщо *expr1* обчислюється в **TRUE**, і в *expr3*, якщо *expr1* обчислюється в **FALSE**.

Структури операторів повторення

Ще одна фундаментальна програмна концепція – ітерації блоку коду, що повторюються. Ви можете використовувати різні методи для виконання ітерацій (званих циклами), що повторюються, у сценаріях PHP.

Цикл while.

Незважаючи на те, що його функціональне призначення відрізняється, PHP-оператор **while** схожий на оператор **if,** включаючи підтримку безлічі умов з використанням логічних операцій.

Загальний синтаксис оператора **while**:

while (умова) {/* Код для повторного виконання, поки вказана умова істинна */ }

Чи в однорядковій формі:

while (умова) /* Код для повторного виконання */

За допомогою цього оператора можна вирішити завдання, виконати які було неможливо раніше – написання сценарію, що виконує підрахунок. ***Приклад***. Відобразити кожне число, яке ділиться на 3, у діапазоні від 1 до 300 і надрукувати з них усі непарні:

```
<?php
$count = 1;
while ($count < 300)
{ if (($count%3)==0){echo  "$count ділиться на 3!<BR>" }
  $count++; }
?>
```

Зверніть увагу, що в операторі **while** збільшується значення змінної **$count.** Без цього рядка коду умова, за якою оператор **while** припиниться, ніколи не буде задоволена. Подібна ситуація називається *нескінченним циклом.*

Завжди треба стежити, щоб усередині будь-якого циклу в PHP було передбачено умову виходу з нього, щоб запобігти виникненню нескінченних циклів.

Як і в більшості структур PHP, що керують, існує кілька варіацій циклу **while,** кожен з яких має синтаксис, що злегка відрізняється, і поведінку.

Цикл do/while.

Загальний синтаксис циклу do/while :

```
do
{/* Виконуваний код */ }
while(умова);
```

На відміну від **while**, оператор **do/while** завжди виконує блок коду, мінімум, один раз. Багаторазове виконання коду визначається умовою, що міститься в його частині **while**. Якщо ця умова повертає значення **true** (істина), код виконується знову, і так до тих пір, поки умова не поверне значення **false** (брехня). Хоч оператор **do/while** використовується винятково, в окремих випадках він може виявитися корисним.

Незважаючи на те, що цикли **do/while** або **while** концептуально можуть впоратися з будь-якою ситуацією, у якій узагалі потрібні цикли, у PHP існує безліч спеціалізованих типів циклів. Один з найчастіше використовуваних типів циклів PHP – цикл **for** – застосовується у випадках, коли потрібна змінна-лічильник.

Цикл for.

Синтаксис циклу **for** виглядає так:

```
for (ініціалізація; умова; пост-обробка)
{
/* /* Код, що підлягає виконанню, поки умова істинна */
}
```

Параметри оператора **for** складніші, ніж у оператора **while**.

Особливість циклу **for** полягає в тому, що параметр розділено на три незалежні сегменти.

Перший сегмент, званий сегментом ініціалізації, виконується негайно після досягнення потоком управління оператора **for** і слугує для ініціалізації будь-яких змінних, використаних у циклі.

Другий сегмент, званий сегментом умови, визначає, з якої умови цикл **for** повинен припинити виконання свого блоку коду.

Третій сегмент, званий сегментом пост-обробки, виконується негайно після блоку коду, що знаходиться усередині циклу **for.**

Технічно в будь-якому з трьох сегментів параметра можуть бути розміщені будь-які коректні PHP-оператори. Проте, як уже згадувалося, оператор **for** призначено для обставин, коли треба працювати з лічильником.

Приклад.

```
<?php
for ($count = 1; $count <= 300; $count++)
{
 if (($count%3) == 0) {echo "$count ділиться
на 3!<BR>" }
}
?>
```

Як ви можете бачити, перший сегмент параметра циклу (сегмент ініціалізації) використовувано для того, щоб присвоїти **$count** початкове значення 1. Потім виконується блок коду і третій сегмент параметра циклу (сегмент пост-обробки) збільшує значення змінної **$count** на 1 до тих пір, поки другий сегмент (умова) не перестане повертати **true**. Цей код виконує те ж саме, що робив попередній приклад з циклом **while**.

Убудовування структур, що керують

Розглянемо, як структури, що керують, можуть бути використано найефективніше для генерації HTML-дескрипторів (чи будь-якого іншого виводу).

Давайте розглянемо приклад, у якому здійснюється спроба відобразити картинку в HTML-документе у разі, коли значення змінної **$display** установлене в **true**. Найбільш загальне рішення має вигляд:

```
<?php
if ($display){echo "<IMG SRC=\"/gfx/mypicture.jpg\
">";}
?>
```

Хоча це рішення працездатне, напевно, що воно недостатньо витончене. Щоб виправити положення, існує альтернативний синтаксис структур PHP, що керують, який дозволяє вам вийти з інтерпретатора PHP і передати вивід безпосередньо „крізь” PHP.

Для цього оператора **if** цей синтаксис має вигляд:

```
<?php  .. if (умова) : ?>
Текст для виводу без інтерпретації PHP
<?php endif; ?>
```

У разі попереднього прикладу цей синтаксис може бути застосовано так, щоб дати той же результат:

```
<?php if($display) :   ?>
<IMG SRC="/gfx/mypicture.jpg">
<?php endif; ?>
```

Цей альтернативний синтаксис допустимий для кожної структури PHP, що керує. Замість фігурної дужки {() зазначається двокрапка (:) для позначення початку структури, що керує, і кожна із структур, що керують, завершується відповідною пропозицією (**endif, endwhile,** закінчується **endструктура: endfor** і т. ін.).

В основному цей альтернативний синтаксис застосовується, коли ви хочете відобразити PHP-код, що не інтерпретується, проте він може використовуватися в будь-якому місці всередині PHP-сценарію. До того ж, цей синтаксис не обов'язковий для запобігання інтерпретації окремого сегмента в документі. Наступний варіант також допустимий, хоча й може спантеличити:

```
<?php
 if($display){
?>
 <IMG SRC="/gfx/mypicture.jpg">
<?php
              }
?>
```

Слід зазначити, що відключення інтерпретатора PHP не обмежується структурами, що керують. У будь-якій точці всередині сценарію інтерпретатор може бути відключено із застосуванням допустимого дескриптора закриття PHP і включено назад за допомогою допустимого дескриптора відкриття PHP.

Функції, визначувані користувачем

У PHP функції визначаються таким чином:

```
function func_name ([variable [= constant], .])
{
  /* /* будь-який допустимий код PHP */
}
```

Ім’я функції (помічене тут як **func_name**) – це довільне ім’я, що підпорядеовується тим же правилам, що й імена змінних PHP, з наступним набором параметрів. Кількість параметрів, їх значення за умовчанням (якщо такі є) й імена задаються розробником. Функції також можуть "повертати" значення за допомогою оператора **return**.

Чи іншими словами:

Функція оголошується за допомогою ключового слова **function,** після якого у фігурних дужках записуються різні оператори, що становлять тіло функції, :

```
function MyFunction()
{
 //  // оператори
}
```

Якщо функція приймає аргументи, то вони записуються як змінні в оголошенні функції.

Аргумент функції є змінним, передаваним у тіло функції для подальшого використання в операціях. У разі, коли функція приймає більше одного аргументу, ці змінні розділяються комами:

```
function MyFunction($var, $var1, $var2)
```

Якщо функція повертає яке-небудь значення, у тілі функції обов'язково має бути наявним оператор return :

```
function MyFunction()
{
   return $ret; // повертається значення змінної $ret
}
```

Приклад.

```
    function get_sum()
{
$var = 5;
$var1 = 10;
$sum = $var + $var1;
return $sum;
}
echo(get_sum()); // виводить 15
?>
```

У цьому прикладі показана функція, що обчислює суму двох чисел. Ця функція не набуває жодного аргументу, а просто обчислює суму й повертає отриманий результат. Після цього вона викликається в тілі оператора **echo** для виведення результату в браузер. Модифікуємо цю функцію так, щоб вона не повертала отриманий результат, а виводила його в браузер. Для цього досить внести оператор **echo** в тіло функції :

```
<?
function get_sum()
{
$var = 5;
$var1 = 10;
$sum = $var + $var1;
echo $sum;
}
get_sum();
?>
```

Змінні **$var** і **$var1** ми можемо оголосити як аргументи й у цьому випадку в тілі функції їх визначати не потрібно:

```
<?
function get_sum($var, $var1)
{$sum = $var + $var1;
echo $sum;}
get_sum(5,2); // виводить 7
?>
```

Змінна, яка, містить значення, що передане через аргумент, називається **параметром** функції.

У розглянутих прикладах аргументи функції передаються за значенням, тобто значення параметрів змінюється тільки всередині функції, і ці зміни не впливають на значення змінних за межами функції:

```
<?
function get_sum($var) // аргумент передається за значенням
{$var = $var + 5;
return $var;}
$new_var = 20;
echo(get_sum($new_var)); // виводить 25
echo("<br>$new_var"); // виводить 20
?>
```

Нижче представлений приклад функції PHP, яка визначає, чи є цей рік високосним.

Приклад. Призначена для користувача функція для визначення високосного року. Рік вважається високосним, якщо: Ділиться на 4, але не на 100.Ділиться на 4 і на 400.

```
<?php
function is_leapyear($year = 2004)
{ $is_leap = (!($year % 4) && (($year % 100) ||
!($year % 400)));
return $is_leap;}
?>
```

Приклад. Виклик призначеної для користувача функції

```
<?php
$answer = is_leapyear(2005);
if ($answer){echo "2005-й рік високосний<BR>" }
else {echo  "2005-й рік не високосний<BR>"}
/* Застосування значення за умовчанням для парамет-
ру */
$answer = is_leapyear ();
if ($answer){echo  "2004-й рік високосний<BR>" }
else {echo "2004-й рік не високосний<BR>" }
?>
```

Слід звернути увагу на зону видимості змінних.

Термін **зона видимості** (*scope)* змінної вказує на те, як PHP вирішує, які з оголошених змінних у якому місці сценарію можуть бути використано.

До теперішнього часу всі змінні в нас належали до так званої *глобальної зони видимості*. Проте змінні, оголошені всередині функції, є частиною *локальної області функції*. Більше того, будь-які змінні, визначені поза функцією, також не доступні усередині неї.

Незважаючи на те, що концепція зони видимості надзвичайно зручна і значно спрощує розробку, існують випадки, коли бажано мати доступ до змінних із зовнішньої по відношенню до функції області видимості. Щоб дозволити це, у PHP включений оператор **global**. Це слово змінює зону видимості цієї змінної з локальної зони видимості на глобальну.

Синтаксис **global** виглядає таким чином:

global $varl [, $var2 [, $var3 [, ..]]];

У PHP змінна, яка передається оператору **global**, не має бути раніше оголошена ні в якій зоні видимості. Зокрема, це зручно тоді, коли ви хочете спроектувати функцію, яка „створює" змінну у глобальному контексті.

Для того, щоб змінні передані функції зберігали своє значення при виході з неї, застосовано передачу параметрів по посиланню. Для цього перед ім'ям змінної необхідно помістити амперсанд (&) :

function get_sum($var, $var1, &$var2)

У цьому випадку змінні **$var** і **$var1** будуть передані за значенням, а змінна **$var2** – за посиланням. У разі, якщо аргумент передається за посиланням, при будь-якій зміні значення параметра відбувається переміна змінної-аргументу.

Приклад.

```
<?
function get_sum(&$var) // аргумент передається по посиланню
{
$var = $var + 5;
return$var;
}
$new_var = 20;
echo(get_sum($new_var)); // виводить 25
echo("<br>$new_var"); // виводить 25
?>
```

Зона видимості змінних

Змінні у функціях мають локальну зону видимості.

```
<?
function get_sum()
{ $var = 5; // локальна змінна
echo $var; }
$var = 10; // глобальна змінна
get_sum(); // виводить 5 (локальна змінна)
echo("<br>$var"); // виводить 10 (глобальна змінна)
?>
```

Локальну змінну можна зробити глобальною, якщо перед її ім'ям зазначити ключове слово **global**. Якщо зовнішня змінна оголошена як **global**, то до неї можливий доступ з будь-якої функції:

```
<?
function get_sum()
{ global $var;
$var = 5; // змінюємо глобальну змінну
echo $var;}
$var = 10;
echo("$var<br>"); // виводить 10
get_sum(); // виводить 5 (глобальна змінна змінена)
?>
```

Доступ до глобальних змінних можна отримати також через асоціативний масив **$GLOBALS** :

```
<?
function get_sum()
{ $GLOBALS["var"] = 20;
// змінюємо глобальну змінну $var
echo($GLOBALS["var"]); }
$var = 10;
echo("$var<br>"); // виводить 10
get_sum(); // виводить 20 (глобальна змінна змінена)
?>
```

Масив **$GLOBALS** доступний у зоні видимості будь-якої функції і містить усі глобальні змінні, які використовуються у програмі.

Час життя змінної

Часом життя змінної називається інтервал виконання програми, протягом якого вона існує. Оскільки локальні змінні мають своєю зоною видимості функцію, той час життя локальної змінної визначається часом виконання функції, у якій вона оголошена. Це означає, що в різних функціях абсолютно незалежно один від одного можуть використовуватися змінні з однаковими іменами.

Для того, щоб локальна змінна зберігала своє попереднє значення при нових викликах функції, її можна оголосити статичною за допомогою ключового слова **static**:

```
<?php
function counter()
{
  static $counter = 0;
  return ++$counter;
}
echo counter();
echo("<br>");
echo counter();
?>
```

Часом життя статичних змінних є час виконання сценарію. Тобто, якщо користувач перезавантажує сторінку, що призводить до нового виконання сценарію, змінна **$counter** у цьому випадку ініціалізувалася спочатку.

Рекурсія

Рекурсією називається така конструкція, при якій функція викликає саму себе. Розрізняють пряму й непряму рекурсії. Функція називається прямо рекурсивною, якщо містить у своєму тілі виклик самій себе. Якщо ж функція викликає іншу функцію, яка у свою чергу викликає першу, то така функція називається побічно рекурсивною.

Розглянемо класичні приклади використання рекурсії - реалізацію операції піднесення до ступеня та обчислення факторіалу числа.

```
<?php
function degree($x, $y)
{
  if($y){return $x*degree($x,$y - 1);}
return 1;
}
echo(degree(2,4)); // виводить 16
?>
```

Цей приклад заснований на тому, що ***xy*** еквівалентно ***x*x(y - 1)***. У цьому коді завдання обчислення **24** розбивається на

обчислення **2*2³**. Потім **2*2³** розбивається на **2*2²** і так до тих пір, поки показник не стане дорівнювати нулю.

```
<?php
function fact($x)
{
  if ($x < 0) return 0;
  if ($x == 0) return 1;
return $x * fact($x - 1);
}
echo (fact(3)); // виводить 6
?>
```

Для негативного аргументу функція повертає нульове значення, оскільки факторіал негативного числа не існує за визначенням. Для нульового параметра функція повертає значення 1, оскільки 0! = 1. В інших випадках викликається та ж функція із зменшеним на 1 значенням параметра, після чого результат множиться на поточне значення параметра. Тобто відбувається обчислення добутку :

```
k(k - 2) * .. * 3 * 2 * 1 * 1
```

Послідовність рекурсивних викликів уривається тільки при виклику **fact(0),** який і призводить до останнього значення 1 у добутку, оскільки останнє вираження, з якого викликається функція, має вигляд `1 * fact(1 - 1).`

Лістинг 3.4. Робота із зоною видимості змінної в PHP.

```
<?php
function createglobal()
{
 global  $my_global;
 $my_global = 10;
}
echo "Значення \$my_global рівне '$my_global'<BR>";
createglobal() ;
echo "Значення \$my_global рівне '$my_global'<BR>";
?>
```

Результуючий вивід має такий вигляд:

Значення `$my_global рівне ''`

Значення `$my_global дорівнює '10'`

Як бачите, незважаючи на те, що **$my_global** ніде не ініціалізувався у глобальній області, за допомогою оператора **global**

вона створюється усередині функції **createglobal()**. Аналогічним чином змінні, які існують у глобальному контексті, можуть також бути перенесені в локальний контекст функції (див. лістинг 3.5).

Лістинг 3.5. Ще про зони видимості змінних PHP.

```
<?php
<?php
function getglobal()
{
 global $my_global;
 echo "Значення \$foobar рівне '$foobar'<BR>";
}
$my global = 20;
getglobal();
?>
```

Це видасть такий результат:

Значення **$my_global** дорівнює **'20'**

Не дивлячись на те, що локальні змінні тісно пов'язані з концепцією зони видимості, деякі змінні, створені PHP, а саме - `$_SERVER, $_GET, $_POST, $_REQUEST, $_GLOBALS, $_COOKIE, $_ENV, $_SESSION` і `$_FILES` доступні завжди незалежно від поточного контексту (імена чутливі до регістра). Ці змінні, звані **суперглобальними,** під час виконання сценарію PHP доступні постійно.

У більшості випадків, коли виконується функція PHP, будь-які змінні, які були створені функцією, знищуються після її завершення. Проте, як більшість сучасних мов програмування, PHP підтримує статичні змінні, які не знищуються при завершенні функції.

Для створення статичної змінної у функції застосовується оператор **`static`**, як показано в наступному прикладі:

```
static $varname [ = constant [, $var2 [= constant]] ..];
```

$varname - це ім'я змінної, яка не повинна знищуватися після закінчення роботи функції, а необов'язкова константа (constant) задає початкове значення цієї змінної.

У лістингу 3.6 наведено приклад використання оператора **static.**

Лістинг 3.6. Робота зі статичними змінними у PHP

```
<?php
function statictest ()
{
  static $count=0;
  $count++;
 return $count;
}
statictest();
statictest() ;
$foo = statictest();
echo "Функція statictest() працювала $foo раза.<BR>";
?>
```

Результат роботи цього сценарію: `Функція statictest()` працювала 2 рази.

Контрольні питання:

1. Назвіть основні кострукції мови PHP.
2. Оператор if у PHP. Операції порівняння.
3. У ситуаціях, коли ви використовуєте умови для визначення значення змінної, який може застосовуватися синтаксис?
4. Чим керує, оператор switch?
5. Де використовується слово break?
6. Які використовуються методи для виконання ітерацій?
7. З чого складається загальний синтаксис оператора while?
8. Який з типів циклів PHP застосовується у випадках, коли потрібна змінна-лічильник?
9. У чому полягає особливість циклу for, якщо параметр розділений на три незалежні сегменти?
10. Як структури, що керують, можуть бути використані найефективніше для генерації HTML–дескрипторів?
11. Показати функції, виклик яких призначено для користувача функції.
12. Які змінні у функціях мають локальну зону видимості?
13. Що називається часом життя змінної?
14. Зазначте класичні приклади використання рекурсії.

Лекція 4. Динамічні змінні та функції. Робота з формами в PHP

План:

1. Знайомство з різними типами змінних.
2. Можливості передачі даних з одного документа в інший.
3. Багатофайлові сценарії PHP.
4. Посилання. Робота з формами.
5. Способи доступу до даних.
6. Використання масивів як імена елементів.
7. Керування завантаженням файлів.

Динамічні змінні

Окрім звичайного маніпулювання даними, PHP дозволяє створювати змінні, ідентифікатори яких (наприклад, $foo - ідентифікатор) невідомі до тих пір, поки сценарій не запущений. Ця концепція "змінних змінних" хоча й не застосовується в щоденній розробці, усе ж в окремих випадках абсолютно незамінна, як ви це побачите далі у книзі, коли розглядатимуться форми.

Так виглядає синтаксис, застосований у випадках, коли ви хочете звернутися до певного значення за іменем змінної :

```
${<вираження>}
```

<вираження> може представляти будь-який допустимий вираз PHP, який при обчисленні повертає значення, що відповідає описаним раніше правилам, що регламентують імена змінних.

Розглянемо наступні два рядки коду, кожен з яких маніпулює змінній **$foo** :

```
<?php
$foo = 5;
${"foo"}++;  // Змінна $foo тепер дорівнює 6
$my_var_name = "foo";
${$my_var_name} ++;
?>
```

При виконанні останнього рядка, наведеного фрагмента збільшиться значення змінної **$foo** до **7**. Дивлячись на цей код, можна бачити, що **$my_var_name** містить рядок **"foo".** Коли виконується рядок, що йде за цим, значення $my_var_name обчислюється і результат трактується як ім'я змінної. Тобто змінна **$foo** збільшується на одиницю.

Динамічні функції

Разом з динамічними змінними PHP також може виконувати функції динамічно. Зокрема, це зручно при контролі допустимості даних, уведених у форму.

Щоб виконати функцію, ім'я якої ви не знаєте до часу виконання, просто додайте список параметрів у кінець будь-якої змінної.

Розглянемо такий фрагмент коду:

```
<?php
function test () {echo "Ласкаво просимо в PHP!<BR>";}
$myfunc = "test";
$myfunc ();
?>
```

Коли цей код виконається, результатом, переданим клієнтові, буде „Ласкаво просимо в PHP!". Незважаючи на те, що в цьому випадку функція **test ()** не приймає параметрів і не повертає значення, це також можливе при динамічному виклику функцій.

Говорячи про параметри функцій, давайте поглянемо на концепцію динамічних параметрів. Досі були показані тільки функції із зумовленим числом параметрів.

PHP також підтримує можливість динамічної передачі параметрів, без попереднього визначення їх до моменту запуску функції. Для цього використовуються функції PHP **func_num_args () і func_get_args ().**

Обидві функції не набувають параметрів і можуть викликатися тільки з середини PHP-функций. Як вказує ім'я, **func_num_args()** повертає кількість аргументів, переданих поточній функції. На додаток до цього друга функція – **func_get_args**

() – призначена для повернення індексованого масиву, що містить значення усіх переданих параметрів.

Нижче представлено приклад застосування обох функцій у призначеній для користувача функції, яка може приймати невизначене число параметрів, :

```
<?php
function dynamic_func()
{
echo "Передане " .func_num_args(). " аргументів". "<BR>";
$args = func_get_args();
    for($i = 0; $i < count($args); $i++)
{$sum+=$args[$i];
echo "Передане значення: {$args[$i]} <BR>";}
 echo " их сумма равна $sum";
}
dynamic_func (1,2,3,4,5,6,7,-10);
?>
```

Багатофайлові сценарії PHP

Завжди хорошою практикою є максимально детальне розбиття сценаріїв на модулі, тобто проектування функцій так, щоб їх можна було використовувати в інших PHP-сценаріях. В цьому відношенні, у міру того, як ви накопичуватимете зростаючу бібліотеку функцій, правильна їх організація стає усе більш важливою. У PHP така організація досягається розподілом сценаріїв на безліч файлів та включенням їх по запиту. До того ж, розміщуючи важливу статичну інформацію (таку, як параметри підключення до бази даних) у різних файлах, її можна надійніше захистити від загального доступу, розташовуючи поза деревом каталогів Web-серверу. Незалежно від причин, включення зовнішніх файлів здійснюється PHP-директивами **include**, **include_once, require і require_once**. Як і можна припустити, з цих чотирьох директив тільки **include** та **require** істотно відрізняються один від одного, і на їх відмінностях буде зосереджено увагу.

Єдина різниця між **include/require** і **include_once/require_once** пов'язана з тим, скільки разів цей файл завантажується. Коли використовуються оператори **include_once/ require_once**, файл не може бути завантажений або виконаний безліч разів.

Якщо спробувати повторно завантажити файл одним з цих методів, така спроба буде проігнорована. Оскільки неприпустимо визначати одну й ту ж функцію безліч разів у сценарії, ці директиви дозволять розробникові включати сценарій з потреби, не перевіряючи, чи був він раніше завантажений.

Загальний синтаксис операторів **include** та **require** :

```
include "file_to_load.php"; include_once
"file_to_load.php";
     чи
require "file_to_load.php";
require_once "file_to_load.php";
```

Слід зазначити, що для кожного такого включення ім'я файла може бути задане строковою константою або змінною, яка зберігає ім'я файла.

Якщо пакувальники URL дозволені в PHP, то ім'я файлу в директивах включення можна використовувати адресу з HTTP.

Як вже було сказано, поділяти функції та код, який їх використовує в багатьох сценаріях, – це хороша практика. Наслідуючи цей принцип, передбачається, що існує PHP-файл з ім'ям **library.inc**, який містить функцію **isleapyear()**, визначену в лістингу 4.1. Слід зазначити, що файли, що не містять PHP-код (такі як файли HTML), також можуть бути включені. При цьому вони будуть відправлені на вихід, як передбачається.

Припускаючи, що **library. inc** знаходиться в тому ж каталозі, що й поточний сценарій, ви можете використовувати функцію **is_leapyear (),** яка знаходиться у файлі **library.inc**, як показано в лістингу 4.1.

Лістинг 4.1. Використання **include** для завантаження файлів у PHP.

```
<?php
  include ('library.inc'); // Дужки не обов'язкові
  $leap = is_leapyear(2003);
```

```
?>
```

У більшості реальних ситуацій файли, які включено в PHP-сценарії, знаходяться не в тому ж каталозі, що і сценарій, який їх використовує. Часто усі файли, що включено, розташовуються в каталозі, шляху пошуку файлів PHP, що є частиною. Коли файл запрошується для включення у PHP-сценарій, то PHP спочатку перевіряє поточний каталог, а потім каталоги з шляху пошуку файлів, що включаються.

Аналогічно, оператор **require** також може використовуватися для включення файла, як показано в лістингу 4.2.

Лістинг 4.2. Використання **require** для завантаження файлів у PHP.

```
<?php
  require ('library.inc'); // Дужки не обов'язкові
     $leap = is_leapyear(2003);
?>
```

Існують дві основні відмінності між **include** та **require**: перше – це здатність повертати значення і друге – за яких умов завантажується запитаний файл.

Коли застосовується директива **include**, PHP відкладає момент завантаження запитаного файла до того моменту, коли сценарій досягне точки виконання оператора **include** та замінить його вмістом цього файла. У протилежність цьому в разі застосування оператора **require**, він замінюється вмістом файла, що включається, незалежно від того, чи буде виконаний оператор **required** (і, відповідно, вміст файла) у процесі нормального виконання сценарію.

З'ясуємо, що саме означає „повернення значення" із зовнішнього файла?

Розглянемо код, наведений у лістингу 4.3, який, як передбачається, збережений у файлі test.inc, і пов'язаний з ним сценарій includetest.php.

Лістинг 4.3. Поведінка файлів, включених за допомогою **include.**

Файл test.inc

```
<?php
echo "Усередині файлу, що включається<BR>";
```

```
    return "Повернений рядок";
    echo  "Після повернення усередині включен-
ня<BR>";
    ?>
```

Файл includetest.php

```
    <?php
    echo "Усередині includetest.php <BR>";
    $ret = include ('test.inc');
    echo  "Виконане включення test.inc <BR>";
    echo "Повернене значення рівне   '$ret'"
    ?>
```

Коли includetest.php виконається результат буде таким:

```
Усередині includetest.php
Усередині included file
Виконано включення test.inc
Повернене значення рівно 'Повернений рядок'
```

Як ви можете переконатися, зовнішні файли не лише застосовні для збереження бібліотек функцій загального користування PHP. Коли використовується оператор **include**, вони можуть бути функціями PHP у повному розумінні цього слова. Зверніть увагу, що коли у файлі **includetest.php** зустрічається оператор **return**, виконання частини файла, що залишилася, уривається.

Здатність повертати значення із зовнішніх файлів обмежена тільки операторами **include** та **include_once**. Оператори **require** і **require_once** не дозволяють це робити.

Посилання

Посилання на змінні

Концепція посилань у PHP істотна для представлення розробникам можливостей посилатися на дані, що містяться у змінній, по одному або більше іменах змінних. Це означає щось більше, ніж просто те, що дві змінні мають одне й те ж значення (наприклад, і $a, і $b дорівнюють 5). Коли одна змінна посилається на іншу, будь-які зміни, виконані для однієї з них, змінюють значення іншої, на яку посилається перша змінна.

У PHP посилання створюються додаванням префікса у вигляді амперсанта & до імені змінної або функції.

Приклад. Використання посилань у PHP.

```
<?php
 $myvar = 42;      /* Ініціалізація $myvar */
 $myref = &$myvar;   /* Створюється посилання
$myref на $myvar */
 echo "Значення \$rayref рівне '$myref<BR>";
 echo "Значення \$myvar рівне '$myvar'<BR>";
 $myvar++;
 echo "Значення \$myref рівне '$myref<BR>";
 echo "Значення \$myvar рівне '$myvar'<BR>";
 $myref--;
 echo  "Значення  \$myref рівне  '$myref <BR>
 echo "Значення  \$myvar рівне  '$myvar '<BR>
?>
```

Після виконання цього сценарію виходить наступний висновок:

```
Значення $myref дорівнює'42'
Значення $myvar дорівнює'42'
Значення $myref дорівнює'43'
Значення $myvar дорівнює'43'
Значення $myref дорівнює'42'
Значення $myvar дорівнює'42'
```

Як бачите, змінні **$myvar** і **$mayref** є посиланнями/псевдонімами для одного й того ж елемента даних, то будь-які зміни однієї з них приводять до зміни іншої.

Оскільки обидві змінні – і **$myvar**, і **$myref** – представляють одні і ті ж дані, то якщо ви зруйнуєте будь-яку з них за допомогою PHP-функції **unset ()**, дані не будуть втрачені. Змінна, що залишилася, як і раніше посилатиметься на ті ж дані. І це справедливо незалежно від того, скільки посилань на одну змінну будуть зруйновано. До тих пір, поки хоч би одна зі змінних посилається на елемент даних, він залишається доступним у сценарії через цю змінну.

Посилання, використанні у функціях

Посилання також можуть використовуватися у поєднанні з функціями. Наприклад, розглянемо ситуацію, у якій бажано повертати більш за одне значення з функції. Повернути більш за одне значення за допомогою оператора **return** неможливо, до того ж може бути небажаним використання глобальних змінних. За допомогою посилань ви можете повернути стільки значень, скільки треба, причому у відносно зрозумілій манері.

Щоб визначити параметр функції як посилання, передаватимете імені змінної параметра префіксом & і передавайте функції посилання за викликом, як показано у прикладі.

Приклад. Передача параметра по посиланню у PHP.

```
<?php
function reference_test($var, &$result, &$result2)
{ $result = "Це повертане значення #1";
$result2 = "Ви передали $var як параметр";
return 42; }
$res = reference_test(10, &$res1, &$res2);
echo "Значення \$res рівне '$res'<BR>";
echo "Значення \$resl рівне '$res1'<BR>";
echo "Значення \$res2 рівне '$res2'<BR>";
?>
```

Це видасть такий результат:

```
Значення $res дорівнює '42'
Значення $resl рівне 'Це повертане значення #1'
Значення $res2 рівне 'Ви передали 10 як параметр'
```

По-перше, оголошено, що функція **reference_test ()** набуває трьох параметрів. Перший параметр **$val** – це стандартний параметр PHP, тоді ті, що залишилися два – **$result** та **$result2** – є параметрами-посиланнями. Коли викликається функція **reference_test ()**, їй передається три параметри. Перший – константне значення 10, а інші два – посилання на змінні **$resl** і **$res2**. При виклику встановлюється зв'язок між змінними **$resultl** та **$result2** з одного боку і **$resl** і **$res2** – з іншого (оскільки вони один на одного посилаються). Тому, коли всередині функції змінюються значення **$result** та **$result2,** пов'язані з ними змінні **$resl** та **$res2** також змінюються. Функція як і раніше

повертає цілу константу 42, яка потім, як і очікується, привласнюється змінній **$res**.

Слід зазначити, що ви не повинні передавати параметри за посиланням під час виконання і при оголошенні функції. Розміщення посилань у будь-якому з цих двох випадків матиме один і той же наслідок. Єдина різниця у тому, що якщо параметри-посилання оголошені в самій функції, то при всіх її викликах параметри передаватимуться по посиланню.

Разом з передачею змінних у функції за посиланням PHP підтримує повертані виносні змінні. У прикладі ця концепція застосовується для створення функції, що повертає посилання на базі значень переданих параметрів.

Приклад. Повернення значення по посиланню

```
<?php
  function find_var($one, $two, $three)
{ if(($one > 0) && ($one <= 10)) return $one;
if(($two > 0) && ($two <= 10)) return $two;
if(($three > 0) && ($three <= 10)) return $three; }
$c_one = 'foo'; $c_two = 42; $c_three = 4;
$right_var = &find_var($c_one, $c_two, $c_three);
$right_var++;
echo "Значення  \$c_ three і \$right_var рівні: ";
echo "$c_three і $right_var<BR>\n";
?>
```

Коли цей код виконується, функція **find_var ()** визначає, які з трьох параметрів знаходяться в діапазоні від 1 до 10, і повертає посилання на цю змінну, яка потім може бути прив'язана до змінної **$rightvar**. У результаті, коли **$right_var** збільшується, також буде збільшена тільки та змінна, яка відповідає вимогам (а саме - **$c_three**), : Значення **$c_three** і **$rightvar** рівні: 5 та 5.

Робота з формами

Коли дескриптор <FORM> поміщається до HTML-документа, єдина зміна, яка вноситься до компонування документа, є створення нового абзацу (подібно до HTML-дескриптора <P>). Щоб форма відповідала своєму призначенню та набувала введення користувача, ви повинні включити відповідні HTML-дескриптори елементів форми.

Відправка форм PHP-сценаріям

Тепер, коли ви ознайомилися з усіма основними елементами HTML-форм (або пропустили цей розділ, оскільки добре їх знаєте), настав час приступити до матеріалу цієї глави, що має відношення до PHP. Інша частина глави буде зосереджена на доступі й роботі з формами та їх даними.

Набуття значень форми

Після того, як форма відправлена до Web-серверу, якщо її атрибут **ACTION** є PHP-сценарій, цей сценарій запускається і набуває даних, які були відправлені. Для того, щоб дані з форм стали доступними у PHP-коді, існує безліч методів здобування даних. Залежно від того, який метод використано для відпраалення даних форми та передачі їх PHP-сценарію (**GET** або **POST**), у PHP визначено два суперглобальні масиви, звані, відповідно, **$_GET** та **$_POST**, які можна застосувати для збереження даних. Це асоціативні масиви, що містять список ключів (елементів, що представляють імена, форми, вказані в атрибутах **NAME**) й асоційованих з ними значень. Таким чином, значення змінної **$_GET['mytext']** міститиме значення елементу HTML-форми, атрибут **NAME** якого дорівнює **mytext**:

Елемент HTML-форми:

```
<INPUT TYPE="text" NAME="mytext" VALUE="Це текст"!>
```

Код PHP :

```
<?php echo $_GET['mytext'];?>
```

При відправці форми виходить вивід: `Це текст!`

Термін **суперглобальний** означає, що незалежно від поточного контексту (наприклад, усередині функції), змінні **$_GET** і **$_POST** будуть завжди доступні без необхідності використання оператора **global** для включення їх у поточний контекст. За додатковою інформацією звертайтеся.

При використанні масиву **$_GET** передбачається, що форма відправляється методом **GET**. Якщо ж це робиться методом **POST**, дані будуть збережені в масиві **$_POST**. Для тих випадків, коли жоден з цих двох методів не підходить (чи може бути будьким), передбачений третій суперглобальний масив **$_REQUEST**,

який комбінує **$_GET, $_POST, $_COOKIE** (суперглобальний масив, що містить так звані **cookie**-змінні) та **$_FILES** (описаний далі в лекції).

PHP (версій, що попередні 4.1.0), за умовчанням створює стандартні імена змінних, такі як **$myvalue** для представлення значення, що міститься в суперглобальній змінній **$__GET** ['myvalue']. Хоча як і раніше можна включити цю поведінку за умовчанням (установивши значення директиви **register_globals** рівною **on**), діяти так дуже не рекомендується з міркувань безпеки.

Правильніший підхід (що забезпечує такий же кінцевий результат) припускає використання функції **import_request_variables()**. Ця функція створює глобальні змінні на зразок **$myvalue** для зберігання значень з суперглобальних масивів. *Синтаксис функції*:

import_request_variables ($types [, $pref ix])

При використанні цієї функції **$types** представляє рядок, який вказує тип змінних для імпорту та повинна містити будь-яку комбінацію (незалежну від регістру) букв P, G і C. Ці букви відповідають $_POST, $_GET і $_COOKIE. Другий параметр не обов'язковий, **$prefix**, якщо він вказаний, представляє префікс до імені кожної створюваної змінної.

У прикладі (передбачається, що $_GET['myvalue'] існує) використовано функцію **import_request_variables()** для створення локальної копії **$_GET**['myvalue'].

Приклад.Використання *функції import_request_variables ()*

Файл форми :

```
<form name="f1" method="get" action="4.php">
<input type="text" name="myvar">
<input type="submit">
</form>
```

Файл 4.php:

```
<?php
 import_request_variables("G", "myget_");
 echo "Значення поля 'myvar' рівне:
$myget_myvar";
?>
```

Використання функції **import_request_variables()** рекомендується замість конфігураційної директиви **register_globals**, проте слід зазначити, що вона не захистить вас від ризику, пов'язаного з обробкою призначених для користувача даних. Завжди рекомендується, щоб дані, передані від користувача, були оброблені перед використанням. Важливо також зазначити, що ця функція імпортує змінні тільки в область глобального контексту. Тобто вона ніколи не повинна викликатися усередині тіла функції.

Засоби доступу до даних

Тепер, коли ви отримали уявлення про те, як організувати доступ до зовнішнього введення з PHP-сценарію, безліч невеликих питань можуть виникнути під час застосування цих знань на практиці.

Можливі різні варіанти передачі значення змінної ***a*** з форми в сценарій:

```
<?php
echo ($a);/*Хотя як і раніше можна включити цю поведінку за умовчанням (встановивши значення директиви register_globals рівною on) */
echo('<hr>');
echo($HTTP_GET_VARS[a]);
echo('<hr>');
/*/*При використанні масиву $_GET передбачається, що форма вирушає методом GET. Якщо ж це робиться методом POST, дані будуть збережені в масиві $_POST.
Для тих випадків, коли жоден з цих двох методів не підходить (чи може бути будь-ким), передбачений третій суперглобальний масив $_REQUEST, який комбінує $_GET, $_POST, $_COOKIE (суперглобальний масив, що містить так звані cookie -переменные*/
echo($_GET['a']);
echo ('<hr>');
/*/*Более правильний підхід (що забезпечує той же кінцевий результат) припускає використання функції import_request_variables (). Ця функція створює глобальні змінні на кшталт $myvalue для зберігання
```

```
значень з суперглобальних масивів. Синтаксис функції виглядає таким чином:
     import_request  variables($types  [,  $prefix])
*/
     import_request_variables("G"," my_");
     echo $my_a;
     ?>
```

Розглянемо елементи HTML, імена яких включають точку.

У HTML передбачений відмінний доступ до елементів форми по імені на кшталт myvar.email (значення, установлене в дескрипторі NAME елемента). Проте враховуючи обмеження, що накладаються на імена змінних, таке ім'я у PHP є неприпустимим. Як наслідок, коли PHP виконує відправку форми, яка містить точку в одному або більш іменах елементів, вони автоматично конвертуються в символи підкреслення.

Таким чином, наступне значення елементу :

```
<INPUT TYPE="text" NAME="myform.email">
```

доступно в PHP таким чином:

```
<?php
echo $_GET[ 'myform_email' ] ; ?>
```

Хоча кожна форма може бути спроектована таким чином, що жодному елементу не привласнюється ім'я зі знаком підкреслення, це поведінка у PHP важлива, коли ви маєте справу з елементами зображень, уживаними для відправлення форми. Якщо такий елемент має атрибут NAME, то після клацання на ньому він вирушає як частина даних з координатами X і Y точки, де користувач виконав клацання. Точніше кажучи, елемент зображення відправляє ці значення як змінні AAAAAA.x і AAAAAA.y, де AAAAAA представляє значення атрибуту NAME. То є для доступу до цих значень у PHP точка має бути замінена знаком підкреслення : Розглянемо наступний елемент зображення для відправки форми :

<INPUT TYPE="image" SRC="images/myimagemap.gif" NAME="mymap"> А ось як дістати доступ до його координат з PHP:

```
 <?php
 echo ("Координата X : ($_GET['mymap_x']) <BR>
 Координата Y : ($_GET['mymap_y'] <BR>")
```

```
?>
```

При роботі з елементами зображень у PHP-сценаріях існує одна поширена помилка кодування. У ситуаціях, коли бажано мати кілька елементів відправлення форми, кожному з них має бути присвоєне значення атрибуту NAME, щоб їх можна було розрізняти.

Якщо використовується декілька елементів зображень для відправки форми, часто для визначення точного елементу відправки перевіряються координати X або Y таким чином: if ($_GET['myimagename_x ']).

На жаль, це неправильний метод. Як відомо, елемент зображення для відправки форми повертає значення координат точки X, Y, у якій виконано клацання, PHP-сценарію. Проблема полягає в тому, що представлений вище оператор if не забезпечує коректну перевірку, якщо координата X набуває значення 0 (у текстових браузерах при клацанні завжди повертаються координати 0, 0). Набагато правильніший метод полягає у виклику функції isset (), як показано на прикладі.

Приклад. Коректна перевірка при відпралення форми з використанням елементів зображень

Елементи зображень для відправлення форми:

```
<INPUT TYPE="image" NAME="submit_one"
SRC="/images/buttonl.gif">
<INPUT TYPE="image" NAME="submit_two"
SRC="/images/button2.gif">
```

Коректне визначення у PHP того, на якій із кнопок відправлення було проведено клацання:

```
<?php
if(isset($_GET['submit_one_x'])) {
/* Код, що виконується при клацанні на першій
кнопці відправки */ }
 elseif(isset ($_GET[ *submit _two_x' ])) {
/* Код, що виконується при клацанні на другій
кнопці відправки */ } else {
/* Код, що виконується, якщо відправки не було
*/ } ?>
```

Наприклад

```
<?php
$x=$_GET['submit_one_x'];
```

```
     echo $x;
     echo("-------------------------");
     if              (isset($_GET['submit_one_x']))
{echo("перша");}
     elseif(isset($_GET['submit_two_x']))
{echo("друга");}
     else { echo("відправки не було");}
      ?>
```

Зверніть увагу в лістингу 4.2 на включений третій випадок (остання пропозиція else). Якщо існують тільки дві кнопки відправлення (як у цьому прикладі), і обидві вони перевіряються, чи буде коли-небудь виконаний код за останнім else? Так! У Microsoft Internet Explorer натиснення клавіші <Enter>, коли фокус знаходиться на певних елементах (таких, як текстове поле), викликає відправлення форми без використання якого-небудь з елементів відправлення.

Використання масивів як імена елементів

Як було описано в першій лекції, елемент <SELECT> потенційно може використовуватися для вибору кількох елементів і тому передасть декілька значень PHP-сценарію. На жаль, установка атрибута NAME у значення ніби myselect створить змінну $_GET['myselect'], яка міститиме тільки останній обраний елемент списку. Очевидно, це не бажаний результат, отже, треба використовувати інший метод. Щоб розв'язати цю проблему, PHP дозволяє створювати масиви динамічно, на базі відправлення форми, за рахунок додавання квадратних дужок у кінець імені елементу. Тобто, myselect стане myselect[], змушуючи PHP додавати елементи в масив $_GET['myselect'] замість того, щоб перезаписувати попереднє значення. Ця концепція проілюстрована у прикладі.

Приклад. Використання масивів з даними форми в PHP

Код HTML :

```
     <SELECT NAME="myselect[] " MULTIPLE SIZE=3>
     <OPTION VALUE="valuel">Выберите мене!</OPTION>
     <OPTION VALUE="value2">НеТ, мене!</OPTION>
     <OPTION  VALUE="value3">Забудьте їх і виберіть
мене!</OPTION>
```

```
"DPTION VALUE="value4">Выберите мене, я ж кра-
щий!</OPTION>
</SELECT>
```

PHP -ход для доступу до вибраних елементів:

```
<?php
foreach($_GET['myselect'] as $val){ echo "Вибране:
$val<BR>";
}
echo "Вибране:  ".count($_GET['myselect']);
?>
```

Ця техніка не обмежується тільки елементом <SELECT> чи масивами, індексованими цілими числами. Якщо ви хочете використовувати строковий ключ для певного елемента форми, зазначте його (без лапок) усередині квадратних дужок:

<INPUT TYPE="text" NAME="data[email]" VALUE="joe .doe@joe.doe.com">

При відправленні форми попередні значення текстових полів будуть доступні за допомогою синтаксису $_GET [' data '] [' email '].

Управління завантаженням файлів

Щоб завантаження файлів працювало правильно у PHP, мають бути коректно встановлені безліч конфігураційних директив у файлі *php. ini*. Особливо це стосується директив *file_uploads, uploads_max_filesize, upload_tmp_dir* і *post_max_size*, які впливають на можливості PHP, пов'язані із завантаженням файлів. Детальніший опис цих директив міститься в керівництві по PHP.

Як згадувалося PHP дозволяє завантажувати файли з HTML-форм за допомогою елементів завантаження файлів. При організації завантаження файлів з HTML-форми слід пам'ятати про певні угоди, що належать до самого дескриптора <FORM>.

Щоб завантаження відбулося успішно, атрибут ENCTYPE дескриптора <FORM> має бути встановлений у MIME-значення multipart/form – data, а атрибут METHOD – в значення POST. Приклад HTML–форми, яка завантажує файл у сценарій upload.php, показаний на прикладі.

Приклад. HTML-код для завантаження файлів по протоколу HTTP

```
    <FORM    METHOD="POST"    ACTION="upload.php"
ENCTYPE="multipart/form - data"> <INPUT TYPE="file"
NAME="myfile">
    <BR>
      <INPUT    TYPE="submit"    VALUE="Завантажити
файл">
    </FORM>
```

Для вказівки максимального розміру завантажуваного файла може використовуватися спеціальний прихований елемент з ім'ям MAX_FILE_SIZE. Це обмеження розміру файла установлюється на клієнтській стороні й може не працювати на деяких клієнтах. У таких випадках потрібно буде реалізувати перевірку розміру за рахунок установки значення директиви *upload_max_filesize*.

При відправленні форми з прикладу файл буде завантажений на Web-сервер і збережений у тимчасовому каталозі, зазначеному директивою *upload_tmp_dir* у файлі **php.ini**. Потім PHP створює суперглобальну змінну **$_FILES i**, у цьому випадку, заносить у масив **$_FILES** ключ *myfile*. Значенням цього ключа буде інший масив, заповнений інформацією про завантажений файл. Зокрема, масив, уміщений у **$_FILES**['myfile'], має значення ключів, наведено в таблиці. 4.2.

Наступний масив ключів може містити або не містити значення, залежно від конкретних обставин завантаження файла. Наприклад, ключ *type* може бути порожнім, якщо браузер не надає MIME-інформації.

Таблиця 4.2 - Ключі, що створюються для файла при завантаженні

Ключ	Опис
name	Ім'я файлу на клієнтській машині.
type	MIME-тип файлу, якщо він відомий.
size	Розмір завантажуваного файлу в байтах.
tmp_name	Тимчасове ім'я, присвоєне файлу самим PHP при завантаженні на сервер.
error	Ціле значення, що представляє помилку, що сталася при завантаженні файлу.

Якщо у процесі завантаження файла виникла помилка, елементу **$_FILES**['myfile']['error'] буде присвоєне цілочисельне

значення, що представляє тип помилки й дорівнює одній з констант, перерахованих у таблиці. 4.3.

Таблиця 4.3 - Константи помилок завантаження файлів у php

Константа	Опис
UPLOAD_ERR_OK	Помилок немає.
UPLOAD_ERR_INT_SIZE	Розмір завантажуваного файлу перевищив максимальний, який заданий в php. ini.
UPLOAD_ERR_FORM_SIZE	Розмір завантажуваного файлу перевищив максимальний, вказаний в прихованому елементі MAX_FILE_SIZE.
UPLOAD_ERR_PARTIAL	Завантаження було перерване, файл завантажений частково.
UPLOAD_ERR_NOFILE	Файл не був завантажений.

Якщо файл не був переміщений, він буде видалений після закінчення виконання PHP-сценарію.

З міркувань безпеки, перш ніж переміщувати файл з тимчасового каталогу в новий, має бути викликана функція *is_uploaded_f ile ()* для підтвердження того, що файл дійсно був завантажений за допомогою PHP. Після того, як завантаження підтверджене, за допомогою функції *move_uploaded_file()* можна перемістити завантажений файл з його поточного розташування в нове. Щоб перемістити файл у каталог призначення, PHP повинен мати права запису в цей каталог. У главі 21 можна знайти детальну інформацію про застосування функцій, що мають відношення до завантаження файлів і роботи з правами доступу.

Функція move_uploaded_file() припускає, що файл знаходиться в каталозі, вказаному в конфігураційній директиві *upload_tmp_dir*.

Код у лістингу 4.15 обробляє файл, завантажений кодом з лістингу 4.14.

Приклад. Виконання завантаження файла у PHP

```
<?php
if(is_uploaded_file($_FILES['myfile1]['tmp_nam
e']))
{  move_uploaded_file($_FILES['myfile'  ]  ['
tmp_name' ], "/path/to/dir/newname"); }
?>
```

Контрольні питання:

1.Назвати типи змінних у PHP. Особливості динамічних змінних та функцій. Які файли включаються у PHP–сценарії?
2.Назвіть дві основні відмінності між директивами include та require.
3.Якими операторами, у PHP–сценарії, обмежена здатність повертати значення із зовнішніх файлів?Як у PHP–сценарію створюються посилання?
4.Які посилання можуть використовуватися у поєднанні з функціями?
5.Наведіть приклади передачі параметра та повернення значення по посиланню у PHP.
6.Які повинні включити відповідні HTML-дескриптори елементів форми, щоб форма відповідала своєму призначенню і приймала введення користувача?
7.Що означає термін суперглобальний? Яка функція створює глобальні змінні для зберігання значень з суперглобальних масивів?
8.Де використовуються функції *import_request_variables()* у PHP?
9.Як організувати доступ до зовнішнього введення з PHP–сценарію?
10.Які існують варіанти передачі значення змінної ***a*** з форми в сценарій?
11.Навести приклад коректної перевірки при відправленні форми з використанням елементів зображень.
12.Як використовувати масиви з даними форми у PHP?
13.Назвати директиви, які впливають на можливості PHP, пов'язані із завантаженням файлів.
14.Запишіть HTML-код для завантаження файлів по протоколу HTTP.
15.Який спеціальний прихований елемент може використовуватися для вказівки максимального розміру завантажуваного файла?
16.Перерахуйте константи помилок завантаження файлів у PHP.

Лекція 5. Рядки в PHP

План:

1. Швидкість і ефективність строкових виразів.
2. Порівняння рядків.
3. Функції строкової обробки.
4. Функції виводу формату.
5. Функції перетворення кодування.
6. Функції роботи з URL.
7. Функції перетворення регістру.

Як і більшість інших мов, PHP визначає рядки як послідовності символів. Важливо розуміти, що поняття "символу" не обмежене тільки тими символами, якими люди користуються щодня, - буквами алфавіту, десятковими цифрами і знаками пунктуації. Сенс, що асоціюється з поняттям "символ", означає тільки один байт даних. Залежно від того, як цей байт використовується, він може бути буквою, точкою растрового зображення і навіть частиною звукової доріжки у форматі MP_3.

Оскільки символ представлений одним байтом даних, убудовані рядки PHP здатні зберігати тільки до 256 різних значень кожного символа. Окремі мови, наприклад, китайська та японська, мають більш ніж 256 символів, і, отже, не можуть бути представлені звичайними рядками у PHP. На щастя, PHP забезпечує набір багатобайтових строкових функцій (MBString), які можуть мати справу з такими мовами, що використовують спеціальні символи.

Швидкість і ефективність строкових виразів

Три строкові нотації, які були розглянуті досі, представляють різний рівень продуктивності.

Незважаючи на невисоку вірогідність того, що продуктивність вашого застосування знизиться через строкові вирази, ви можете зіткнутися з іншими, серйознішими проблемами, а тому треба знати, що найшвидший спосіб оголошення рядків – використання поодиноких лапок, оскільки в цьому випадку

інтерпретатору не потрібно буде сканувати рядок і виконувати необхідні підстановки (зокрема: V і \\).

Синтаксис з подвійними лапками повільніший, оскільки весь рядковий вираз, що вміщується в них, повинен піддаватися скануванню і підстановкам змінних усередині нього. І, нарешті, синтаксис "here.doc" є найбільш повільним, тому що окрім операцій сканування для пошуку підстановок і спеціальних символів інтерпретатор також повинен піклуватися про пошук вашого роздільника.

Порівняння рядків

Визначення стосунків між двома рядками не таке очевидне, як ті ж операції з числами. Головна проблема полягає в контексті. Якщо ви порівнюєте рядки у двійковій формі, то два слова "Macro" і "macro" будуть повністю різні, оскільки байтове значення символ "M", відрізняється від значення символу "m". Проте, залежна від ваших вимог, "Macro" і "macro" можуть бути еквівалентні й повинні трактуватися саме так.

Найбільш простий спосіб порівняння двох рядків передбачає використання вбудованих операцій порівняння PHP. Проте є кілька пасток, про які слід знати. Розглянемо, наприклад, такий вираз:

```
echo (0 == 'O');
```

Через те, що один з операндів є цілим числом, рядок '0' неявно також конвертується в ціле перед виконанням порівняння, призводячи до результату, що дорівнює 1. Поки, на перший погляд, це не здається проблемою, але дуже легко стає нею, коли зустрінеться щось подібне до:

```
(0 (0 ==  'Macro');
```

Оскільки рядок 'Macro' конвертується в ціле значення 0 при обчисленні вираження, результатом також буде true, що на виході дасть 1.

Ніколи не застосовуйте прості операції порівняння при роботі з рядками, якщо тільки точно не впевнені в тому, що робите. Замість цього необхідно користуватися операціями порівняння з контролем типів, які можуть перевірити, що два порі-

внювані операнди належать до одного типу даних, перш ніж насправді порівнювати їх значення.

Наприклад, вираження:

((Про = = = 'Macro');

поверне значення false. Те ж станеться з виразом:

((Про = = = 'O').

Найакуратніший спосіб порівняння двох рядків, забезпечує функція *Strcmp()* :

int strcmp($vall, $val2);

Результат, зворотний *strcmp()*, залежить від алфавітного близько двох рядків. Якщо $vall і $val2 ідентичні, *strcmp()* поверне 0. Слід пам'ятати, що функція *strcmp()* виконує порівняння, чутливе до регістру, тому, наприклад, "Macro" і "macro" не будуть ідентичні.

Якщо два значення не еквівалентні, порівняння виконується відповідно до поточних локальних установок клієнта – іншими словами, використання алфавітних правил сортування залежить від локального середовища, у якому виконується сценарій. Якщо $vall за абеткою передує $val2, результат буде негативним, інакше – позитивним.

Наприклад, при використанні локальних установок на комп'ютері автора (Canadian - English), виходять такі результати:

```
echo strcmp('Apple ', 'Banana');// повертає < 0
echo strcmp('apple ', 'Apple'); // повертає > 0
echo strcmp{'1 ', 'test');      // повертає < 0
```

Цифри мають менше контекстне значення, ніж букви, а заголовні букви – менше контекстне значення, ніж прописні. У канадському англійському регіональному стандарті це також торкається двійкових значень кожного символа, але це не завжди правильно, зокрема для тих мов, де комбінації символів розглядаються як один символ (наприклад, 'de' у німецькій, або 'cz' – у чеській).

Якщо вам треба реалізувати порівняння рядків, не чутливе до регістру, на цей випадок PHP пропонує функцію strcasecmp(), яка набуває тих же параметрів, що і *strcmp()*:

echo strcasecmp('Macro ',' macro'); // повертає Про

Строкових функцій у PHP багато, причому багато з них дублюють один одного. Проте, прийоми роботи з основними блоками функцій (функції пошуку й заміни в тексті, функції видалення пробільних символом (trim-функции), функції виводу формату) потрібно представляти дуже чітко.

Рядок – це послідовність символів на зразок *char*, яка закінчується нульовим символом.

Функція htmlspecialchars

Синтаксис:

string htmlspecialchars(string str [, int quote_style [, string charset]]);

Перший аргумент – рядок, у якому потрібно виконати перетворення. Як другий необов'язковий аргумент приймається одна з двох констант : ENT_QUOTES або ENT_QUOTES. Перша константа передається, якщо потрібно транслювати лапки, а друга, якщо цього робити не потрібно. Третій необов'язковий аргумент набуває рядок, що представляє набір символів, використовуваних у перетворенні (за умовчанням ISO - 8859-1).

htmlspecialchars() належить до функцій перетворення символів. Вона проводить перетворення спецсимволів у їх HTML еквіваленти й гарантує, що введений користувачем скрипт відобразиться, але не виконається.

Функції пошуку в тексті

substr(); strpos(); strrpos(); strstr(); strchr(); stristr(); strrchr(); substr_count(); strspn(); strcspn()

substr()

Синтаксис:

string substr(string string, int start[, int length])

Ця функція повертає частину рядка. Перший аргумент – початковий рядок; другий – положення в рядку, який потрібно повернути, першого символа (відлік починається з нуля); третій – довжина рядка в символах, який потрібно повернути. Якщо

третій аргумент не вказаний, то повертається уся частина рядка, що залишилася.

Приклад:

```
<?php
$string = substr("Hello, world"!, 7, 3);
echo ($string);
?>
```

Результат виконання – рядок "wor". Якщо третій параметр не вказувати, тобто написати $string = substr("Hello, world"!, 6); то повернеться підрядок "world"!.

strpos()

Синтаксис:

string strpos(string haystack, string needle[, int offset])

Ця функція забезпечує дію, зворотну функції *substr*. Тобто вона повертає позицію в рядку haystack, у якій знайдений переданий нею підрядок *needle*.

Тобто такий скрипт

```
<?php
$string = strpos("Hello, world"!, "world");
echo($string);
?>
```

поверне число 7.

Необов'язковий параметр offset дозволяє вказати в рядку позицію, з якою потрібно починати пошук:

strrpos()

Синтаксис:

string strrpos(string haystack, string needle)

Ця функція шукає в рядку haystack останню позицію, де зустрічається символ *needle*.

strstr()

Синтаксис:

string strstr(string haystack, string needle)

Функція strstr() повертає ділянку рядка, заданого в параметрі haystack, починаючи з першого фрагмента, вказаного в параметрі needle і до кінця рядка. У разі невдачі функція повертає *false*.

Приклад:

```
<?php
$url = "http ://www.softtime.ru";
$www = strstr($url","w);
echo ($www);
?>
```

Результат: www.softtime.ru

Ця функція чутлива до регістру. Якщо *needle* не є рядком, значення перетвориться в ціле і використовується як код шуканого символа.

strchr()

Синтаксис:

string strchr(string haystack, string needle)

Ця функція працює абсолютно ідентично до функції *strstr()*

stristr()

Синтаксис:

string stristr(string haystack, string needle)

Ця функція працює абсолютно аналогічно до функції *strstr()*, тільки є нечутливою до регістру.

strrchr()

Синтаксис:

string strrchr(string haystack, string needle)

Ця функція відрізняється від аналогічних нею тим, що здійснює пошук останнього входження підрядка. Тобто функція *strrchr()* повертає ділянку рядка, заданого в параметрі *haystack*, починаючи з останнього фрагмента, зазначеного в параметрі needle і до кінця рядка. У разі невдачі повертає *false*.

Чутлива до регістру. У разі, якщо *needle* не є рядком, те значення перетвориться в ціле і використовується як код шуканого символа.

substr_count()

Синтаксис:

int substr_count(string haystack, string needle)

Функція *substr_count()* знаходить кількість входжень фрагмента в рядок. Вона повертає число фрагментів *needle*, наявних в рядку *haystack*. Функція чутлива до регістру.

Приклад:

```
<?
$str = "робота з графічним редактором";
$substr_count = substr_count($str","ра);
echo ($substr_count);
?>
```

Результат: 2.

strspn()

Синтаксис:

int strspn(string str1, string str2)

Функція *strspn()* визначає наявність початкових символів у рядку. Вона повертає довжину початкового фрагмента рядка str1, що складається повністю з символів, які є в рядку str2.

Приклад:

```
<?
$str = "проста програма";
$substr_count = strspn($str","прогр);
echo ($substr_count);
?>
```

Результат:3.

strcspn()

Синтаксис:

int strcspn(string str1, string str2)

Зворотна функції *strspn()* функція *strcspn()* Визначає відсутність початкових символів у рядку.

Функція *strcspn()* повертає довжину початкового фрагмента рядка str1, що складається повністю не з символів, які є в рядку str2.

Функції strlen(), chr() і ord()

strlen() ; chr() ; ord()

strlen()

повертає довжину рядка, яку набуває як аргумент

chr()

Ця функція набуває як аргумент ASCII код символа і повертає відповідний цьому коду фактичний символ

```
<?php
 $str = chr(36);
 echo ($str); // повертає символ "$"
?>
```

ord()

Виконує дію, зворотну до функції *chr()*

Trim-функции (функції видалення пробільних символів)

trim(); ltrim(); rtrim(); chop()

Це дуже корисна група функцій, без яких складно обійтися при роботі з рядками. До пробільних символів належить символи "\n", "\r", "\t", "\v", "\0" і пропуск. Особливо часто ці функції використовуються при файловому введенні-виводі.

trim()

Ця функція набуває як аргумент рядок, і видаляє з неї пропуски ліворуч і праворуч.

Функція *ltrim()* видаляє з рядка початкові пробільні символи (тобто ті, які ліворуч); *rtrim()* – кінцеві пробільні символи. Функція *chop()* є синонімом до функції *rtrim()*.

Функції виводу формату. Специфікатори перетворення

printf( ); sprintf( ); sscanf( )

printf() і sprintf()

Синтаксис:

int printf(string format [, mixed args])

string sprintf(string format [, mixed args])

Ці дві функції призначено для виводу формату і відрізняються вони тим, що функція *printf()* здійснює форматування і виводить результати у вихідний потік (браузер або консоль), а *sprintf()* після здійснення необхідного форматування просто повертає рядок.

Цими функціями можна користуватися просто як функціями виводу :

```
<?
printf("Hello"!); // виводить  "Hello"!
sprintf("Hello"!); // сама по собі нічого не виводить,
$str = sprintf("Hello"!); //  повертає рядок
printf($str); // яку потім можна вивести у вихідний потік
?>
```

Аргумент format цих функцій є рядком, що містить спеціальні символи (специфікатори) перетворення. Символи, які залишаються незмінними при форматуванні рядка, називають **директивами.**

Специфікація визначається символом "%", за яким може йти до п’яти специфікаторів у такому порядку:

1. **Специфікатор заповнення**

Установлює символ, яким рядок заповнюється до заданого розміру. За умовчанням використовується пропуск. Специфікатор заповнення діє тільки за наявності специфікатора мінімальної ширини.

2. **Специфікатор вирівнювання**

За умовчанням доповнення рядка до мінімальної ширини здійснюється з лівого краю (тобто рядок вирівнюється по правому краю). Якщо додано символ дефіса, то рядок вирівнюється по лівому краю.

3. **Специфікатор мінімальної ширини**

Є ціле число, що задає мінімальний розмір форматованого рядка. Якщо переданий рядок менший, то вона доповнюється символами, вказаними в специфікаторі заповнення.

4. **Специфікатор точності**

Призначений для вказівки кількості десяткових знаків у представленні чисел з плаваючою точкою. При застосуванні

цього специфікатора для форматування рядків він визначає максимальну кількість символів, яку треба узяти з переданого рядка.

5. **Специфікатор типу**

Цей специфікатор призначений для вказівки типу даних, які передані як аргумент. Специфікатор може набувати одне з таких значень :

b – ціле число, що представляється у двійковому вигляді;

з – ціле число, що представляється у вигляді символа з тим же ASCII кодом;

d – ціле число, що представляється в десятковому вигляді;

f – число з плаваючою точкою, що представляється у вигляді десяткового дробу;

про – ціле число, що представляється у вісімковому вигляді;

s – рядок;

x – ціле число, що представляється в шістнадцятиричному вигляді в нижньому регістрі;

X – ціле число, що представляється в шістнадцятиричному вигляді у верхньому регістрі

Тепер виведемо за допомогою функції *printf()* рядок у форматі дати dd/mm/yyyy. Наступний код виводить у результаті рядок "02/03/2003":

```
<?
$day = 2;
$month = 3;
$year = 2003;
printf("%02d/%02d/%04d", $day, $month, $year);
?>
```

У цьому прикладі місяць і день ми форматуємо як двозначні числа, а рік – як чотиризначне. При цьому ми вказуємо, що цілі числа доповнюються до мінімальної довжини нулями ліворуч: %04d

Перший символ – специфікатор заповнення і він дорівнює нулю. Оскільки доповнення дописується на початок числа, специфікатор вирівнювання відсутній. Специфікатор мінімальної ширини дорівнює двом. Специфікатор точності також відсутній, оскільки ми форматуємо ціле число. Специфікатор типу пред-

ставлений символом d, оскільки ми форматуємо число як десяткове ціле.

Наведемо ще один приклад.

```
<?
$value = 19;
printf("%.3f", $value);
?>
```

Цей скрипт виводить число 19 у вигляді 19.000

функція sscanf()

Синтаксис:

mixed sscanf(string str, string format [, string var1 ..])

Ця функція є повною протилежністю до функції *printf()*. Вона інтерпретує рядок str згідно з форматом format, аналогічно специфікації *printf()*. При вказівці тільки двох аргументів набутих значень повертаються як масив.

Давайте розглянемо такий приклад. Нехай у нас є рядок, у якому знаходиться інформація про назву й серійний номер виробу у вигляді "maxtor/203-5505" і нам потрібно витягнути з неї серійний номер. Пишемо ось такий скрип:

```
<?
$product = "maxtor/203-5505";
$str = sscanf($product","maxtor/%3d-%4d);
echo ("$str[0]-$str[1]");
?>
```

Якщо не зазначають у функції необов'язкові аргументи, то на виході вийде масив, і тому при виводі ми працюємо зі значеннями масиву.

Важливе зауваження: при вказівці необов'язкових параметрів їх потрібно передавати по посиланню (посилання позначається вказівкою символ "&" перед змінною).

Приклад, у якому використовуються додаткові необов'язкові параметри, :

```
<?
$book = "1\tThinking in PHP";
$str = sscanf($book",%d\t%s %s %s", &$id,
&$first, &$second, &$last);
echo ("book number $id - $first $second $last");
?>
```

Функції перетворення кодування

convert_cyr_string(); bin2hex()

convert_cyr_string()

Синтаксис:

string convert_cyr_string(string str, string from, string to)

Функція перетворить рядок з одного кодування кирилиці в іншу. Вона переводить рядок str з кодування *from* у кодування *to*. Значення *from і to* – це поодинокі символи, що визначає кодування:

k - koi8 - r;
w - windows - 1251;
i - iso8859 - 5;
a - x - cp866;
d - x - cp866;
m - x - mac - cyrillic;

Приклад. Перекодувати слово "студент" з кодування windows – 1251 в koi8 – r і назад:

```
<?
$str1 = "студент";
$str2 = convert_cyr_string($str1","w","k);
echo ("result of translate '$str1' to koi8-r
is '$str2'");
echo ("<br>");
$str3 = convert_cyr_string($str2","k","w);
echo ("result of translate '$str2' to win is
'$str3'");
?>
```

Результат:

result of translate 'студент' to koi8 - r is 'УФХДЕОФ'result of translate 'УФХДЕОФ' to win is 'студент'

bin2hex()

Проводить побайтове перетворення символьних даних у шістнадцятиричний вигляд.

Функція *bin2hex()* приймає як єдиний параметр рядок і повертає строкове шістнадцятиричне представлення символів, що містяться в цьому рядку.

Функції роботи з бінарними даними

pack() ; unpack()

pack()

Синтаксис:

string pack(string format [, mixed $args, ..])

Функція *pack()* упаковує задані в її параметрі аргументи в бінарний рядок. Формат параметрів і їх кількість задається параметром $format, за допомогою тих же специфікаторів форматування, тільки без знака %. Після кожного специфікатора може стояти число, яке говорить про те, скільки інформації буде оброблено цим специфікатором. Для форматів a, A, h і H це число задає кількість символів, які будуть уміщені в бінарний рядок з тих, що знаходяться у параметрі-рядку при виклику функції (тобто фактично визначається розмір поля виведення рядка).

unpack()

Синтаксис:

array unpack(string $format, string $data)

Розпаковує дані з двійкового рядка згідно з форматом. Функція повертає масив, що містить розпаковані елементи.

Функції роботи з блоками тексту

wordwrap(); str_replace(); substr_replace(); strtr(); stripslashes(); stripcslashes(); addslashes(); addcslashes(); quotemeta(); strrev()

wordwrap()

Синтаксис:

string wordwrap(string str [, int width [, string break [, int cut]]])

Функція *wordwrap()* розбиває початковий текст на рядки з певними завершуючими символами. Згідно з синтаксисом, ця функція розбиває блок тексту *str* на кілька рядків, які завершаються символами *break*, так, щоб в одному рядку було не більш за *width* букви.

Оскільки розбиття відбувається по межах слів, текст залишається читаним.

Приклад:

```
<?php
$str = "Вставай, країна величезна";
$mod_str = wordwrap($str, 5",\n");
echo($mod_str);
?>
```

str_replace()

Синтаксис:

string str_replace(string from, string to, string str)

Функція *str_replace()* замінює в початковому рядку str одні підрядки на інші. Тобто функція замінює в рядку str усі входження підрядка from на to і повертає результат. Ця функція може працювати з двійковими рядками.

Функція корисна тим, що з її допомогою можна користуватися стандартними тегами HTML. Ви можете за допомогою цієї функції замінити символи, вибрані для форматування на стандартні теги HTML.

Приклад:

```
$txt = str_replace("[B]","<B>"txt);
```

Тобто якщо Ви використовуєте для відображення тексту напівжирним шрифтом символи "[B]"Ви повинні їх замінити на символ "<B>"використовувані у HTML.

substr_replace()

Синтаксис:

string substr_replace(string str, string replacement, int start [, int length])

Ця функція замінює в початковому рядку одні підрядки на інші. Вона повертає рядок *str*, в якій частина від символу з позицією start і завдовжки *length* замінюється рядком replacement. Якщо аргумент довжини *length* не зазначений, заміна проводиться до кінця.

Якщо значення аргументу *start* позитивне, то відлік здійснюється від початку рядка str, інакше – від кінця. У разі ненегативного значення *length*, воно вказує довжину замінюваного

фрагмента. Якщо ж воно негативне, то це – число символів від кінця рядка str до останнього символа замінюваного фрагмента.

strtr()

Синтаксис:

string strtr(string str, string from, string to)

string strtr(string str, array from)

Ця функція призначена для комплексної заміни в рядку і має два вигляду синтаксису. У першому випадку функція *strtr()* повертає рядок str, у якій кожен символ, наявний у рядку from, замінюється на символ з рядка to. Якщо рядки from і to різної довжини, то зайві кінцеві символи в тому рядку, який довший, ігноруються. У другому випадку функція *strtr()* повертає рядок, у якому фрагменти рядка *str* замінюються на фрагменти, що відповідають індексам значень елементів масиву from.

Спочатку функція намагається замінити найбільші фрагменти початкового рядка, при цьому не виконуючи заміну в уже модифікованих частинах рядка. Таким чином можна виконати кілька замін відразу:

```
<?php
$str = array("" => "student"" "<name2>" => "Informatika");
$str_out = "вивчення програмування";
echo strtr($str_out,$str);
?>
```

А ось як можна за допомогою цієї функції відмінити дію функції htmlspecialchars() :

```
<?
$var = array_flip(get_html_translation_table());
$str = strtr($str, $var);
?>
```

Тобто, з рядка, у якому всі спецсимволи замінені на їх HTML-еквіваленти, ми отримуємо початковий рядок.

stripslashes()

Синтаксис:

string stripslashes(string str)

Функція видалення зворотних слешів. Тобто проводиться заміна в рядку *str* передуючих слешем символів на їх кодові еквіваленти. Функція працює з символами: \.

А також перетворить спецсимволи в їх двійкове представлення. Вона повертає рядок, у якому закоментовані зворотним слешем спецсимволи (з метою візуального відображення), перетворяться в їх двійкове представлення. Функція розпізнає С - подібні запису (вісімкові й шістнадцятиричні послідовності \n, \r і т.ін.).

addslashes()

Синтаксис:

string addcslashes(string str, string charlist)

Функція додавання слешів перед символами рядка "," і \. Цю функцію зручно використовувати при виклику функції *eval().*

Функція повертає рядок *str*, у яку вставлені символи зворотного слеша перед перерахованими в списку *charlist* символами. Це дозволяє перетворювати символи, які не друкуються в їх візуальне С-представленя.

quotemeta()

Синтаксис:

string quotemeta(string str)

Функція цитування метасимволів. Повертає рядок, у який додано зворотні слеші перед кожним з символів, : . \\ + * ? [^] ($). Функцію можна використовувати для підготовки шаблонів у регулярних виразах.

strrev()

Синтаксис:

string strrev(string str)

Функція здійснює реверс рядка.

Функції об’єднання/розподілу рядків

str_repeat(); str_pad(); chunk_split(); strtok(); explode(); implode(); join()

str_repeat()

Синтаксис:

string str_repeat(string str, int number)

Функція повторення рядка. Повторює рядок *str* у кількість разів, яка вказана в параметрі *number*.

Приклад:

```
<?
echo str_repeat("Hello"!,3);
// виводить Hello! Hello! Hello!
?>
```

str_pad()

Синтаксис:

string str_pad(string strinput, int pad_length [, string pad_string [, int pad_type]])

Ця функція доповнює рядок іншим рядком до певної довжини. Аргумент *strinput* задає початковий рядок. Аргумент *pad_length* задає довжину повертаного рядка. Якщо він має менше значення, ніж початковий рядок, то додавання не проводиться. Необов’язковий аргумент *pad_string* вказує на те, який рядок використовувати як доповнення. За умовчанням використовуються пропуски. Необов’язковий аргумент *pad_type* вказує, з якого боку слід доповнювати рядок: справа, зліва або з обох боків. Аргумент *pad_type* може набувати таких значень:

STR_PAD_RIGHT (за умовчанням)
STR_PAD_LEFT
STR_PAD_BOTH

chunk_split()

Синтаксис:

string chunk_split(string str [, int chunklen [, string end]])

Ця функція повертає фрагмент рядка. Функція *chunk_split()* повертає рядок, у якому між кожним блоком рядка

str довжиною *chunklen* (за умовчанням chunklen = 76) вставляється послідовність з роздільників *end* (за умовчанням: *end* = *"\r\n"*).

strtok()

Синтаксис:

string strtok(string arg1, string arg2)

Функція повертає рядок частинами. Вона повертає частину рядка *arg1* до роздільника *arg2*. При наступних викликах функції повертається наступна частина до наступного роздільника, і так до кінця рядка. При першому виклику функція набуває два аргументи: початковий рядок *arg1* і роздільник *arg2*. Зверніть увагу, що при кожному наступному виклику *arg1* вказувати не потрібно, інакше повертатиметься перша частина рядка.

Приклад:

```
<?
$str ="I am very glad to see%you% adhahjasdad";$tok = strtok($str, " ");
while($tok)
{ echo ($tok);
echo (" ");
$tok = strtok(" %");};
// виведе: "I" "am" "very" "glad" "to" "see" "you":
?>
```

Те що, ми написали в кінці рядка, не виводиться. Це результат того, що коли в рядку послідовно зустрічаються два або більше за роздільники, функція повертає порожній рядок, що, може припинити цикл обробки, як у цьому прикладі.

explode()

Синтаксис:

string explode(string arg, string str [, int maxlimit])

Функція *explode()* здійснює розподіл рядка в масив. Вона повертає масив рядків, кожна з яких відповідає фрагменту початкового рядка str, що знаходиться між роздільниками, зазначеним аргументом arg. Необов'язковий параметр maxlimit вказує максимальну кількість елементів у масиві. Неподілена частина, що залишилася, міститиметься в останньому елементі.

Приклад:

```
<?
$str = "one two three for five";
$str_exp = explode(" "str);
/* теперь $str_exp = array([0] => one, [1] =>
two, [2] => three, [3] => for, [4] => five) */
?>
```

implode()

Синтаксис:

string implode(string var, array param)

Функція *implode()* є зворотною до функції *explode()* і проводить об'єднання масиву в рядок. Функція повертає рядок, який послідовно містить всі елементи масиву, заданого в параметрі param, між якими вставляється значення, зазначене в параметрі var. Для прикладу виведемо все те, що ми тільки що "умасивили" функцією *explode()*, використовуючи пропуск як роздільник:

Приклад:

```
<?php
$str = "one two three four five";
$str_exp = explode(" "str);
/* $str_exp = array([0] => one, [1] => two,[2]
=> three, [3] => four, [4] => five) */
$str_imp = implode(" "str_exp);
echo($str_imp);
?>
```

join()

Синтаксис:

string join(string var, array param)

Аналог функції *implode()* – здійснює об'єднання масиву в рядок.

Останні три функції мають пряме відношення як до масивів, так і рядків.

Функції порівняння рядків

strcmp(); strncmp() ; strcasecmp(); strncasecmp(); strnatcmp(); strnatcasecmp(); similar_text(); levenshtein().

strcmp()

Синтаксис:

int strcmp(string str1, string str2)

Ця функція порівняння рядків. Вона порівнює два рядки й повертає:

0 - 0 – якщо рядки повністю збігаються;

1 – якщо, рядок str1 лексикографічно більший *str2*;

1 – якщо, навпаки, рядок str1 лексикографічно менший *str2*.

Функція є чутливою до регістру, тобто регістр символів впливає на результати порівнянь (оскільки порівняння відбувається побайтово).

***Приклад*:**

```
<?
$str1 = "ttt";
$str2 = "ttttttttttt";
echo("Result of strcmp ($str1, $str2) is ");
echo(strcmp (str1, str2)); echo("<br>");
echo("Result of strcmp ($str2, $str1)> is ");
echo(strcmp (str2, str1)); echo("<br>");
echo("Result of strcmp ($str1, $str1) is ");
echo(strcmp (str1, str1));
?>
```

strncmp()

Синтаксис:

int strncmp(string str1, string str2, int len)

Ця функція відрізняється від *strcmp()* тим, що порівнює початки рядків, а точніше перші len байтів. Якщо *len* менше довжини найменшої з рядків, то рядки порівнюються цілком.

В іншому функція поводиться аналогічно *strcmp()*, тобто повертає:

0 - 0 – якщо рядки повністю збігаються;

1 – якщо, рядок str1 лексикографічно більше *str2*;

1 – якщо, навпаки, рядок str1 лексикографічно менше *str2*.

Порівняння також проводиться побайтовий, тому функція чутлива до регістра.

strcasecmp()

Синтаксис:

int strcasecmp(string str1, string str2)

Функція працює аналогічно до *strcmp()*, тільки при роботі не враховується регістр букв.

strncasecmp()

Синтаксис:

int strncasecmp(string str1, string str2, int len)

Функція *strncasecmp()* порівнює початки рядків без урахування регістру.

strnatcmp()

Синтаксис:

int strnatcmp(string str1, string str2)

Здійснює так зване „природне" порівняння рядків.

Про цю функцію поговоримо детальніше. Ця функція є імітатором порівняння рядків людиною, тобто вона порівнює рядки так, як їх порівнювала б людина. Тобто, якщо, наприклад, ми порівнюватимемо файли з назвами *pict1.gif, pict20.gif, pict2.gif, pict10.gif*, то звичайне порівняння приведе до такого їх розташування: *pict1.gif, pict10.gif, pict2.gif, pict20.gif*. Природне ж сортування дасть результат, який нам звичніший: pict1.*gif, pict2.gif, pict10.gif, pict20.gif*.

Приклад

```
<?
$array1 = $array2 = array("pict10.gif",
"pict2.gif", "pict20.gif", "pict1.gif");
echo("звичайне сортування :"); echo ("<br>");
usort ($array1, strcmp);
print_r ($array1);
echo ("<br>");
echo("природне сортування :");
 echo("<br>");
usort ($array2, strnatcmp);
print_r ($array2);
?>
```

strnatcasecmp()

Синтаксис:

int strnatcasecmp(string str1, string str2)

Здійснює „природне” порівняння рядків без урахування регістру. Функція виконує те ж саме, що і *strnatcmp()*, тільки без урахування регістру.

similar_text()

Синтаксис:

int similar_text(string str_first, string str_second [, double percent])

Ця функція здійснює визначення схожості двох рядків.

Функція *similar_text()* визначає схожість двох рядків за алгоритмом Олівера. Функція повертає число символів, що збіглися в рядках *str_first і str_second*. Третій необов'язковий параметр передається за посиланням і в ньому зберігається відсоток збігу рядків. Функція повільна.

***Приклад**:*

```
<?php
$str1 = "Привіт, студент";
$str2 = "студенти";
$var = similar_text($str1, $str2);
$var1 = similar_text($str1, $str2, &$tmp);
// параметр $tmp передаємо по посиланню
echo("Результат виконання функції similar_text()для рядків $str1 і $str2 у кількості символів :");
echo("<br>"); echo("$var"); echo("<br>");
echo("і у відсотках :"); echo("<br>");
echo($tmp); // для виведення інформації у відсотках звертаємося до $tmp
?>
```

Результат:

Результат виконання функції *similar_text()* для рядків Привіт, студент і студенти у кількості символів: 7 і в процентах:60.8695652174

levenshtein()

Функція виконує визначення „відмінності Левенштейна” двох рядків.

Синтаксис:

int levenshtein(string str1, string str2)

int levenshtein(string str1, string str2, int cost_ins, int cost_rep, int cost_del)

int levenshtein(string str1, string str2, function cost)

Під поняттям „Відмінність Левенштейна” розуміється мінімальне число символів, яке вимагалося б замінити, вставити або видалити для того, щоб перетворити рядок *str1* на *str2*. Швидша, ніж попередня.

У функції три види синтаксису. У першому випадку функція повертає число необхідних операцій над символами рядків для перетворення str1 в str2:

```
<?
$str1 = "Hello, world"!;
$str2 = "Hello"!;
$var = levenshtein($str1,$str2);
echo($var); // поверне 7
?>
```

У другому випадку додаються три додаткові параметри: вартість операції вставки cost_ins, заміни cost_rep і видалення cost_del. Природно, функція в цьому випадку стає менш швидкодіючою.

```
<?
$str1 = "Hello, world"!;
$str2 = "Hello"!;
$var = levenshtein($str1,$str2, 3,3,3);
echo($var); // поверне 21
?>
```

Третій варіант дозволяє зазначити функцію, використовувану для розрахунку складності трансформації.

Функції роботи з URL

parse_url(); parse_str(); rawurlencode(); rawurldecode(); base64_encode(); base64_decode()

parse_url()

Функція обробляє URL і повертає його компоненти.

Синтаксис:

array parse_url(string url)

Ця функція повертає асоціативний масив, що містить безліч різних наявних компонентів URL : "scheme", "host", "port", "user", "pass", "path", "query" і "fragment".

Приклад

```
<?
$url = "http ://www.google.com.ru/search?hl = ru&ie
= UTF-8 & oe=UTF - 8&q=softtime&lr=";
$arr = parse_url($url);
print_r($arr);
?>
```

Результат:

```
Array ( [scheme] => http [host] =>
www.google.com.ru [path] => /search [query] =>
hl=ru&ie=UTF - 8&oe=UTF - 8&q=softtime&lr= )
```

parse_str()

Заносить компоненти URL у змінні.

Синтаксис:

void parse_str(string str [, array arr])

Функція *parse_str()* інтерпретує рядок str так, ніби цей рядок містив у собі змінні та їх значення і передавався б у URL. Тобто функція встановлює для цих змінних значення. Якщо заданий другий необов'язковий параметр arr, то значення, знайдені за допомогою функції *parse_str()*, зберігатися не в глобальних змінних, а в елементах зазначеного масиву.

rawurlencode()

Функція кодування URL.

Синтаксис:

string rawurlencode(string str)

Функція *rawurlencode* повертає рядок, у якому всі не алфавітно-цифрові символи, за винятком дефіса „-", знака підкреслення „_" і точки „.", замінені послідовностями, : знака відсотка (%), за яким ідуть дві шістнадцятиричні цифри, що означають код символа. Кодування потрібне для того, щоб буквені символи не оброблялися як роздільники URL-рядки й не спотворювалися при передачі в мережах.

rawurldecode()

Здійснює декодування URL.

Синтаксис:

string rawurldecode(string str)

Думаємо, що з цією функцією все зрозуміло, додамо лише, що вона аналогічна до функції *urldecode(),* але не сприймає "+" як пропуск.

base64_encode()

Функція кодує дані в кодуванні MIME base64.

Синтаксис:

string base64_encode(string data)

Функція base64_encode() повертає рядок, переданий у параметрі data в кодуванні MIME base64.

Кодування MIME base64 розроблено для передачі двійкових даних через транспортні шари, які не містять восьмий біт, наприклад, як поштові тіла. Зауважемо, що дані в кодуванні Base64 займають приблизно на 30 % більше місця, ніж оригінал.

base64_decode()

Декодує дані, закодовані в кодуванні MIME base64.

Синтаксис:

string base64_decode(string encoded_data)

Функція base64_decode() декодує encoded_data і повертає оригінал даних.

Функції перетворення регістру

strtolower(); strtoupper(); ucfirst(); ucwords()

strtolower()

Функція здійснює перетворення символів рядка в нижній регістр.

Синтаксис:

string strtolower(string str)

Приклад

<?

```
$str = "HELLO WORLD";
$str = strtolower($str);
echo $str; // виведе "hello world"
?>
```

strtoupper()

Здійснює перетворення рядка у верхній регістр.

Синтаксис:

string strtoupper(string str)

Функція не дуже добре працює з кирилицею.

ucfirst()

Здійснює перетворення першого символу рядка у верхній регістр.

Синтаксис:

string ucfirst(string str)

Функція повертає рядок із заголовним першим символом. При конвертації символів кирилиці можуть бути непорозуміння.

Приклад

```
<?
$str = "hello world";
$str = ucfirst($str);
echo $str; // виведе Hello world
?>
```

ucwords()

Здійснює перетворення першого символа кожного слова рядка у верхній регістр.

Синтаксис:

string ucwords(string str)

Повертає рядок, у якого перший символ кожного слова в рядку заголовний.

Під словом розуміється ділянка рядка, якій передує пробільний символ: пропуск, перехід на новий рядок, прогін сторінки, повернення каретки, горизонтальна і вертикальна табуляція.

Приклад

```
<?
$str = "hello world";
$str = ucwords($str);
echo $str;
```

```
// виведе Hello World
?>
```

Контрольні питання:

1. Скільки різних значень кожного символа здатні зберігати вбудовані рядки?
2. Який існує найшвидший спосіб оголошення рядків у PHP?
3. Назвіть найбільш простий спосіб порівняння двох рядків, який передбачає використання вбудованих операцій порівняння PHP.
4. Яка функція здійснює перетворення спецсимволів у їх HTML еквіваленти й гарантує, що введений користувачем скрипт відобразиться, але не виконається?
5. Які існують функції пошуку в тексті у PHP?
6. Які функції використовуються при видаленні пробільних символів?
7. Назвіть функції призначені для виводу формату, для форматування і виводу результати у вихідний потік (браузер або консоль)?
8. Як діють специфікатори: заповнення, вирівнювання, мінімальної ширини, точності, типу?
9. Наведіть функції роботи з бінарними даними.
10. Назвіть функції роботи з блоками тексту.
11. Особливості синтаксису функцій об’єднання/розподілу рядків.
12. Наведіть приклади використання функцій роботи з URL та перетворення регістру.

Модуль 2

Лекція 6. Масиви

План:

1. Методи ініціалізації масивів.
2. Цикл foreach для обходу масивів.
3. Багатовимірні масиви.
4. Функції кількості, пошуку й установки в початок масиву.
5. Робота з курсором (покажчик) масиву.
6. Функції сортування масивів.

Методи ініціалізації масивів

У PHP існують 2 методи ініціалізації масивів.

Присвоєння значень елементів масиву:

```
<?php
$car[] = "passenger car";
$car[] = "land-rover"; echo($car[1]); // виводить "land-rover"
?>
```

Індекс масиву можна вказати явно:

```
<?
$car[0] = "passenger car";
$car[1] = "land-rover"; echo($car[1]); // виводить "land-rover"
?>
```

Помітимо, що якщо при оголошенні елементів масиву змішуються змінні з явною індексацією, і без індексації, то тому елементу, індекс якого не заданий, PHP присвоїть перший доступний індекс, після найбільшого використаного досі індексу. Наприклад, якщо ми створимо масив з елементами, індекси яких будуть рівні 10, 20 і 30, а потім створимо елемент, індекс якого явно не вкажемо, то йому автоматично надасться індекс 31:

```
<?
$car[10] = "passenger car";
$car[20] = "land-rover";
$car[30] = "station-wagon";
$car[] = "victoria";echo($car[31]); ?>
```

Альтернативний спосіб визначення масивів полягає у використанні конструкції **array**() :

```
<?
$car = array("passenger car"," land - rover");
echo($car[1]); // виводить "land-rover"
?>
```

Для явної вказівки індексів у цьому випадку застосовується оператор **=>**:

```
<?php
$car = array("passenger car", 5 => "land-rover", "station-wagon", "victoria");
echo($car[0]); echo("<br>"); // виводить "passenger car"
echo($car[5]); echo("<br>"); // виводить "land-rover"
echo($car[6]); echo("<br>"); // виводить "station-wagon"
echo($car[7]); // виводить "victoria"
?>
```

Індексами масиву можуть бути і рядки:

```
<?
$car = array("pc" => "passenger car", "lr" => "land-rover");
echo($car["lr"]); echo("<br>"); // виводить "land-rover"
echo($car["pc"]);  // виводить "passenger car"
?>
```

Цикл foreach для обходу масивів

Обхід масиву в циклі у PHP4 можна організувати за допомогою циклу *foreach*, який має такий синтаксис:

foreach (**array as [$key =>] $value**)

{ **statements**; }

Сенс цього циклу – при проході кожного елемента масиву в змінну **$key** вміщується індекс цього елемента, а в змінну **$value** – його значення. Імена цих двох змінних довільні.

Приклад:

```
<?php
$car = array("passenger car", "land-rover", "station-wagon", "victoria");
foreach($car as $index => $val)
{echo("$index => $val <br>");}
?>
```

Змінна *$key* необов'язкова й може бути опущена.

Багатовимірні масиви

count() ; in_array() ; reset()

Для ініціалізації багатовимірних масивів використовуються вкладені конструкції *array()*. Обхід багатовимірних масивів досягається за допомогою вкладених циклів.

Приклад створення й обходу багатовимірного масиву.

```
<?php
$Areol = array(
"Луганська область" => array("Ровеньки","Стаханов","Краснодон",
"Сватово")
"Донецька область" => array("Енакиево", "Дебальцево")
"Львівська область" => array("Яремча","Ивано-Франковск") );
foreach($Areol as $key => $type)
{ echo("<h2>$key</h2>\n"".<ul>\n");
        foreach($type as $Areol)
          {echo("\t<li>$Areol</li>\n"); }
}
echo("</ul>\n");
?>
```

Функції кількості, пошуку й установки в початок масиву

Функція count()

Синтаксис:

int **count(**mixed **var)**

Ця функція набуває як аргумент масиву і повертає кількість елементів у ньому.

Функція in_array()

Синтаксис:

boolean **in_array(**mixed **needle**, array **haystack** [, bool **strict**]**)**

Ця функція шукає в масиві **haystack** значення **needle** і повертає **true,** якщо воно знайдене і **false** інакше.

Функція reset()

*Синтаксис:*mixed **reset(array array)**

Функція **reset()** установлює покажчик масиву на перший елемент і повертає значення першого елемента масиву.

У кожного масиві у PHP є внутрішній покажчик на поточний елемент масиву. При роботі з такими конструкціями, як **foreach,** не потрібно думати про покажчик, оскільки **foreach** установлює його на початок масиву. Проте, багато інших функцій масивів, такі як **prev(), next()**, займаються переміщенням покажчика масиву, що може мати значення при роботі з такими функціями, як **array_walk()**, які починають обробку масиву з того місця, де знаходиться покажчик.

Робота з курсором (покажчик) масиву

end(); next() ; prev(); current() ; key(); each()

end()

*Синтаксис:*mixed **end(array array arr)**

Функція *end()* виконує дію, зворотну до функції *reset()* – переносить курсор в кінець масиву. Синтаксис функції аналогічний до синтаксису функції *reset()*.

next()

Функція *next()* здійснює перенесення курсора масиву вперед на одну позицію.

*Синтаксис:*mixed **next(array array arr)**

Функція переміщує курсор масиву на подальший елемент, повертаючи значення елемента, на якому знаходився курсор до переміщення. Якщо елементів у масиві більше не залишилося, функція повертає **false**. При роботі з функцією потрібно не забувати: **false** також повертається, якщо курсор зустрінеться елемент з порожнім значенням. Для праці з масивами, що містять порожні елементи, краще використовувать функцію **each**().

prev()

Функція **prev**() проводить перенесення курсора назад на одну позицію. Синтаксис і робота функції повністю аналогічні до функції **nex**t().

*Синтаксис:*mixed **prev(array array arr)**

current()

Для визначення поточного елемента масиву без зміни положення курсора використовується функція **current**().

*Синтаксис:*mixed **current(array array arr)**

Функція **current**() повертає значення елемента, на якому в цей момент знаходиться курсор масиву, при цьому не зрушуючи курсор. У тому випадку, якщо курсор виявився за межами масиву, або масив складається з порожніх елементів, функція поверне **false**.

Повним синонімом функції **current**() є функція **pos**().

key()

Функція **ke**y() повертає індекс поточного елемента масиву.

*Синтаксис:*mixed **key(array array arr)**

each()

Синтаксис: array **each(array array arr)**

Функція **each()** повертає пару „індекс–значення” поточного елемента масиву і зрушує курсор масиву на подальший елемент. При цьому, як видно, функція повертає масив, причому він має чотири елементи:

1. [1] => „значення”
2. [value] => „значення”
3. [0] => індекс
4. [key] => індекс

Якщо курсор досяг кінця масиву, функція повертає **false**.

Приклад:

```
<?php
$name = array("maks", "igor", "sergey");
$each_name = each($name);
print_r($each_name); echo("<br>");
```

```
$each_name = each($name);
print_r($each_name); echo("<br>");
$each_name = each($name);
print_r($each_name);
?>
```

Функцію **each**() можна використовувати в парі з функцією **list**() для перебору елементів масиву.

Приклад:

```
<?php
$name = array("maks", "igor", "sergey");
reset($name);
while(list($key, $val) = each($name))
{echo ("$key = $val<br>");}
?>
```

Функція array_walk()

array_walk()

Досить важлива функція, що дозволяє застосовувати призначену для користувача функцію до кожного елементу масиву.

Синтаксис:

bool **array_walk(**array **arr**, callback **func** [, mixed **userdata**]**)**

Функція застосовує призначену для користувача функцію **func** до кожного елемента масиву **arr**. У призначену для користувача функцію передаються два або три аргументи: значення поточного елемента, його індекс і аргумент **userdata**. Останній аргумент є необов'язковим.

Зауважемо, що в разі, якщо **func** вимагає більше трьох аргументів, при кожному її виклику видаватиметься попередження, і, щоб вони не видавалися, треба поставити знак „**@"** перед функцією **array_walk()**. Функція **func** набуває значень і індекси масиву **arr** таким чином, що не може їх змінювати. Якщо виникає така необхідність, треба передавати аргумент **arr** по посиланню. У цьому випадку усі зміни відіб'ються в масиві.

Приклад: вивести всі елементи масиву. Для цього ми повинні спочатку написати функцію, яка їх виводитиме, а потім викличемо її за допомогою функції **array_walk()** :

```
<?php
$name = array ("m"=>"maks", "i"=>"igor", "s"=>"sergey");
function print_array ($item, $key)
{echo "$key=>$item<br>\n";}
array_walk ($name, 'print_array');
?>
```

Важливе зауваження. Ми не встановили курсор масиву на початок масиву, і перед тим, як викликати функцію **array_walk(),** потрібно викликати функцію **reset()** для цієї мети, тому що **array_walk()** починає працювати з того елемента, на якому знаходиться курсор масиву.

Приклад: збільшимо значення кожного його елемента на одиницю.

```
<? $number = array ("1"=>"15", "2"=>"20", "3"=>"25");
function printarray ($item, $key)
{echo "$key=>$item<br>\n";}
function add_array (&$item, $key)
// параметр $item передаємо по посиланню, оскільки його нам потрібно змінювати
{$item = $item + 1;}
echo("Before:<br>");
array_walk ($number, 'printarray');
echo("After:<br>");
array_walk ($number, 'add_array');
array_walk ($number, 'printarray');
?>
```

Функції сортування масивів

sort(); rsort(); asort(); arsort(); ksort(); krsort(); array_reverse(); shuffle(); natsort()

sort()

Функція сортування масиву за збільшенням.

Синтаксис:

void **sort(**array **array** [, int **sort_flags**]**)**

Функція сортує масив array за збільшенням. Необов'язковий аргумент **sort_flags** вказує як саме повинні сортува-

тися елементи (задає прапори сортування). Допустимими значеннями цього аргументу є такі:

–SORT_REGULAR – задає нормальне порівняння елементів (порівнює елементи "як є");

–SORT_NUMERIC – порівнює елементи як числа;

–SORT_STRING – порівнює елементи як рядки.

Взагалі кажучи, ця функція призначена для сортування списків. Під списком розуміється масив, ключі якого починаються з нуля і не мають пропусків. Функція **sort()** сприймає будь-який масив як список.

Приклад:

```
<?php
 $arr = array("Андрій", "Сергій", "Ольга", "Алла", "Микола", "Борис");
sort($arr);
for($i=0; $i < count($arr); $i++)
{ echo "$i:$arr[$i] <br>"; }
?>
```

Ця функція сортує рядки в так званому альфа-бета порядку, тобто за старшинством першої букви в алфавіті, що не завжди правильно.

rsort()

Сортування масиву за зменшенням.

Синтаксис:

void **rsort(**array **arr** [, int **sort_flags**]**)**

Аналогічна до функції **sort()**, тільки сортує за зменшенням.

asort()

Сортування асоціативного масиву за збільшенням.

Синтаксис:

void **asort(**array **arr** [, int **sort_flags**]**)**

Функція **asort()** сортує масив arr так, щоб його значення йшли в алфавітному (якщо це рядки) або зростаючому (для чисел) порядку. Важлива відмінність цієї функції від функції **sort()** полягає в тому, що при застосуванні функції **asort()** зберігають-

ся зв'язки між ключами і відповідними ним значеннями, чого немає у функції **sort()** (там ці зв'язки розриваються).

Приклад:

```
<?php
$arr = array("a" =>"one","b" => "two","c" => "three","d" => "four");
asort($arr);
foreach($arr as $key => $val)
{echo (" $key => $val ");}
?>
```

За умовчанням функція **asort()** сортує масив в алфавітному порядку. Значення прапорів сортування **sort_flags** наведено в описі функції **sort()**.

arsort()

Сортування асоціативного масиву за зменшенням.

Синтаксис:

void **arsort(**array **arr** [, int **sort_flags**]**)**

Ця функція аналогічна до функції **asort()**, тільки вона впорядковує масив не за збільшенням, а за зменшенням.

ksort()

Сортування масиву за збільшенням ключів.

Синтаксис:

int **ksort(**array **arr** [, int **sort_flags**]**)**

У цій функції сортування здійснюється не за значеннми, а ключами в порядку їх зростання.

Приклад:

```
<?php
$arr = array("a" =>"one","d" => "two","c" => "three","ab" => "four");
ksort($arr);
foreach($arr as $key => $val)
{echo (" $key => $val <br>");}
?>
```

krsort()

Сортування масиву за зменшенням індексів.

Синтаксис:

int **krsort(**array **arr** [, int **sort_flags**]**)**

Те ж саме, що і функція **ksort()**, тільки упорядковує масив по ключах в зворотному порядку (за зменшенням).

array_reverse()

Розставляння елементів масиву у зворотному порядку.

Синтаксис:

array **array_reverse(**array **arr** [, bool **preserve_keys**]**)**

Функція **array_reverse()** повертає масив, елементи якого слідують у зворотному порядку відносно масиву **arr**, переданого в параметрі. При цьому зв'язки між ключами і значеннями зберігаються. Якщо необов'язковий параметр **preserve_keys** зробити **true**, тоді у зворотному порядку переставляться ще і ключі.

Приклад:

```
<?php
$arr = array ("php", 4.0, array ("green", "red"));
$result = array_reverse ($arr, true);
echo "Масив: <br>";
foreach($arr as $key => $val)
{echo ("$key => $val <br>");}
echo("<br>");
foreach($result as $key => $val)
{echo ("$key => $val <br>");}
echo("<br>");
echo "Сортований масив: <br>";
$result_keyed = array_reverse ($arr, false);
foreach($result_keyed as $key => $val)
{echo ("$key => $val<br> ");}
?>
```

shuffle()

Перемішування елементів масиву випадковим чином.

Синтаксис:

void **shuffle(**array **arr)**

Функція **shuffle**() перемішує елементи масиву **arr** випадковим чином.

natsort()

Виконує „природне” сортування масиву.

Синтаксис:

void **natsort(**array **arr)**

З таким сортуванням ми вже зустрічалися, коли працювали з рядками. Попрацюємо з нею ще разок. Як пам’ятаєте, під природним сортуванням розуміється сортування таким чином, коли елементи того, що сортується, розташовуються у „зрозумілому” для людини порядку.

Приклад:

```
<?php
$array1 = $array2 = array("pict 10.gif", "pict 2.gif", "pict 20.gif", "pict 1.gif");
echo ("звичайне сортування :"); echo ("<br>");
sort($array1);
print_r($array1);
echo "<br>"; echo ("природне сортування :");
echo "<br>";
natsort($array2);
print_r($array2);
?>
```

Контрольні питання:

1. Які існують методи ініціалізації масивів?
2. У чому полягає альтернативний спосіб визначення масивів ?
3. Для чого використовують цикл *foreach* у PHP?
4. За допомогою яких циклів здійснюється обхід багатовимірних масивів? Навести приклад.
5. При роботі яких функцій здійснюється обробка масиву?
6. Яка функція сприймає будь-який масив як список?
7. Назвіть функції сортування асоціативного масиву за збільшенням, зменшенням, у зворотньому порядку?
8. Сформувати поняття „природне” сортування масиву.

Лекція 7. Файловий ввід/вивід і файлова система

План:

1. Перевірка існування і розміру файла.
2. Запис у файл. Читання файла в масив.
3. Перенаправлення файли у стандартний вихідний потік.
4. Запуск зовнішніх програм. Зворотні апострофи.
5. Робота з файловою системою.
6. Відкриття з'єднання через Socket.
7. Відображення і зміна характеристик файлів (додатково).

Вхідні і вихідні потоки даних інтенсивно використовуються при розробці web–додатків. Не обмежуючись простим читанням/записом файлів, PHP надає в розпорядження програміста засобу перегляду і модифікації серверної інформації, а також запуску зовнішніх програм.

Перевірка існування і розміру файла

Перш ніж намагатися працювати з файлом, бажано переконатися в тому, що він існує. Для вирішення цього завдання зазвичай використовуються дві функції:
file_exists( ) і is_file( ).

file_exists( )

Функція *file_exists ( )* перевіряє, чи існує заданий файл. Якщо файл існує, функція повертає TRUE, інакше повертається FALSE.

Синтаксис функції *file_exists( ):*

bool file_exists(string файл)

Приклад перевірки існування файлу*:*

```
if (! file_exists ($filename)) :
print "File $filename does not exist"!;
endif:
```

is_file( )

Функція **is_file( )** перевіряє існування заданого файла й можливість виконання з ним операцій читання/запису. По суті, *is_file()* є надійнішою версією *file_exists( )*, яка перевіряє не лише факт існування файла, але й те, чи підтримує він читання і запис даних :

bool is_file(string файл)

Наступний приклад показує, як переконатися в існуванні файла й можливості виконання операцій з ним:

```
<?php
$file = "somefile.txt";
if (is_file($file)) :
print "The file $file is valid and exists"!;
else:
print "The file $file does not exist or it is not a valid file"!;
endif;
?>
<br>
<?php
$$file="primer.txt";
if (is_file($file)){echo("ok");} else{("No");};
?>
```

Переконавшись у тому, що потрібний файл існує і з ним можна виконувати різні операції читання/запису, можна переходити до наступного кроку–відкриття файла.

filesize( )

Функція *filesize( )* повертає розмір (у байтах) файла із заданим ім'ям або FALSE у разі помилки.

Синтаксис функції *filesize( ):*

int filesize(string ім'я_файлу)

Припустимо, ви хочете визначити розмір файла *pastry.txt*. Для отримання потрібної інформації можна скористатися функцією filesize( ) :

```
$fs = filesize("pastry.txt"); print "Pastry.txt is $fs bytes".;
```

Виводиться наступний результат:

```
Pastry.txt is 179 bytes.
```

Перш ніж виконувати операції з файлом, необхідно відкрити його і зв'язати з файловим маніпулятором, а після завершення роботи з файлом його слід закрити.

Відкриття і закриття файлів

Перш ніж виконувати операції введення/виводу з файлом, необхідно відкрити його функцією *fopen( )*.

fopen( )

Функція *fopen( )* відкриває файл (якщо він існує) і повертає ціле число – так званий файловий маніпулятор *(file handle*).

Синтаксис функції *fopen( ):*

int fopen (string файл, string режим [, int включення_шляху]).

Файл, що відкривається, може знаходитися в локальній файловій системі, існувати у вигляді стандартного потоку введення/виводу або представляти файл у видаленій системі, що приймається засобами HTTP або FTP.

Параметр файл може задаватися в кількох формах, перерахованих нижче:

Якщо параметр містить ім'я локального файлу, функція **fopen( )** відкриває цей файл і повертає маніпулятор.

Якщо параметр заданий у вигляді **php://stdin, php://stdout** або **php ://stderr**, відкривається відповідний стандартний потік **введення/виводу**.

Якщо параметр розпочинається з префікса **http://**, функція відкриває підключення HTTP до сервера і повертає маніпулятор для вказаного файла.

Якщо параметр розпочинається з префікса **ftp ://**, функція відкриває підключення FTP до серверу і повертає маніпулятор для вказаного файла. В цьому випадку слід звернути особливу увагу на дві обставини: якщо сервер не підтримує пасивний режим FTP, виклик **fopen( )** завершується невдачею. Більше того, FTP–файли відкриваються або для читання, або для запису.

При роботі в пасивному режимі сервер чекає підключення з боку клієнтів. При роботі в активному режимі сервер сам встановлює з'єднання з клієнтом. За умовчанням зазвичай використовується активний режим.

Параметр–режим визначає можливість виконання читання і запису у файл. У таблиці. 7.1 перераховані окремі значення, що визначають режим відкриття файла.

Таблиця 7.1.– **Режими відкриття файлу**

Режим	Опис
r	Тільки читання. Покажчик поточної позиції встановлюється на початок файла
r+	Читання і запис. Покажчик поточної позиції встановлюється на початок файла
w	Тільки запис. Покажчик поточної позиції встановлюється на початок файла, а увесь вміст файла знищується. Якщо файл не існує, функція намагається створити його
w+	Читання і запис. Покажчик поточної позиції встановлюється на початок файла, а увесь вміст файлу знищується. Якщо файл не існує, функція намагається створити його
a	Тільки запис. Покажчик поточної позиції встановлюється у кінці файла. Якщо файл не існує, функція намагається створити його
a+	Читання і запис. Покажчик поточної позиції встановлюється у кінці файла. Якщо файл не існує, функція намагається створити його

Якщо необов'язковий третій параметр включення_шляху дорівнюэ 1, то шлях до файла визначається по відношенню до каталогу файлів, що включаються, вказаного у файлі **php.ini**.

Приклад відкриття файла *функцією fopen(*). Виклик *die( ),* використовуваний у поєднанні з *fopen(*), забезпечує висновок повідомлення про помилку в тому випадку, якщо відкрити файл не вдасться:

```
<?php
$file = "userdata.txt"; // Деякий файл
$fh = fopen($file, "a+") or die("File ($file) does not exist"!);
?>
```

Аналогічний приклад:

```
<?php
$file="primer 1.txt";
if (is_file($file)){echo("ok"); $fh = fopen($file, "a+");} else{("No");die("File ($file) does not exist"!);};
 ?>
```

Наступний фрагмент відкриває підключення до сайту PHP (http://www.php.net) :

```
<?php
$site = "http ://www.php.net ": // Сервер, доступний через HTTP
```

```
$sh = fopen($site, "r"); //Зв'язати маніпулятор з індек-
сною сторінкою php.net
?>
```

Після завершення роботи файл завжди слід закривати функцією fclose( ).

fclose ( )

Функція *fclose( )* закриває файл із заданим маніпулятором. При успішному закритті повертається TRUE, при невдачі - FALSE.

Синтаксис функції *fclose( ):*

int fclose(int маніпулятор)

Функція **fclose( )** успішно закриває тільки ті файли, які були раніше відкриті функціями **fopen( )** або **fsockopen( ).**

Приклад закриття файла:

```
<?php
$file = "userdata.txt";
if (file_exists($file)):
$fh = fopen($file, "r");
// Виконати операції з файлом
fclose($fh);
else: print "File Sfile does not exist"!;
endif;
 ?>
```

Запис у файл

З відкритими файлами виконуються дві основні операції – читання і запис.

is_writeable( )

Функція *is_writeable( )* дозволяє переконатися в тому, що файл існує і для нього дозволена операція запису. Можливість запису перевіряється як для файла, так і для каталогу.

Синтаксис функції *is_writeable( ):*

bool is_writeable (string файл)

Одна важлива обставина: швидше за все, PHP працюватиме під ідентифікатором користувача, використовуваним web–сервером (як правило, „nobody”).

Приклад використання *is_writeable( )* наведений в описі функції *fwrite( ).*

fwrite ( )

Функція *fwrite( )* записує вміст строкової змінної у файл, заданий файловим маніпулятором.

Синтаксис функції *fwrite( ):*

int fwrite(int маніпулятор, string змінна [, int довжина])

Якщо при виклику функції передається необов'язковий параметр – довжина, запис зупиняється або після запису вказаної кількості символів, або досягнувши кінця рядка.

Приклад: Перевірка можливості запису у файл

```
<?php
   // Інформація про трафік на призначеному для користувача сайті
$data = "08 :13:00|12:37:12|208.247.106.187|Win98";
$filename = "somefile.txt";
   // Якщо файл існує і в нього можливий запис
if ( is_writeable($filename)) :
   // Відкрити файл і встановити покажчик поточної позиції в кінець файлу
$fh = fopen($filename, "a+");
   // Записати вміст $data у файл
$_success = fwrite($fh, $data);
   // Закрити файл
fclose($fh);
else: print "Could not open Sfilename for writing";
endif;
   ?>
```

Функція **fputs( )** є псевдонімом **fwrite( )** і може використовуватися всюди, де використовується **fwrite( ).**

fputs( )

Функція **fputs( )** є псевдонімом **fwrite( )** і має такий самий синтаксис.

Синтаксис функції *fputs( ):*

int fputs(int маніпулятор, string змінна [, int довжина])

Читання з файла

Читання є найголовнішою операцією, що виконується з файлами. Нижче описані окремі функції, що підвищують ефектив-

ність читання з файла. Синтаксис цих функцій практично точно копіює синтаксис аналогічних функцій запису.

is_readable( )

Функція **is_readable( )** дозволяє переконатися в тому, що файл існує і для нього дозволена операція читання. Можливість читання перевіряється як для файла, так і для каталогу.

Синтаксис функції *is_readable( ):*

bool is_readable (string файл]

Швидше за все, PHP працюватиме під ідентифікатором користувача, використовуваним web–сервером (як правило, „nobody"), тому для того, щоб функція *is_readable( )* повертала TRUE, читання з файла має бути дозволене всім охочим.

Приклад перевірки, що файл існує і доступний для читання:

```
<?php
$filename='primer.txt';
if ( is_readable($filename)) :
// Відкрити файл і встановити покажчик поточної позиції в кінець файлу
$fh = fopen($filename, "r");
else:
print "$filename is not readable"!;
endif;
fclose($fh);
?>
```

fread( )

Функція **fread( )** читає з файла, заданого файловим маніпулятором, задана кількість байт.

Синтаксис функції *fwrite( ):*

int fread(int маніпулятор, int довжина).

Маніпулятор повинен посилатися на відкритий файл, доступний для читання (див. опис функції *is_readable( )*). Читання припиняється після прочитання заданої кількості байт або досягнувши кінця файла.

Розглянемо текстовий файл *pastry.txt*, приведений нижче. Читання і виведення цього файла в браузері здійснюється таким фрагментом:

Приклад. Текстовий файл *primer.txt*

```
Recipe: Pastry Dough
```

```
1 1/4 cups all - purpose flour
3/4 stick (6 tablespoons) unsalted butter, chopped
2 tablespoons vegetable shortening 1/4 teaspoon salt
3 tablespoons water
```

Програма для читання

```
<?php
$filename='primer.txt';
$fh = fopen($filename, "r") or die("Can' t open file"!);
$file = fread($fh, filesize($fh));
print $file;
fclose($fh);
?>
```

Використовуючи функцію *fllesize( )* для визначення розміру *pastry.txt* у байтах, ви гарантуєте, що функція *fread( )* прочитає увесь вміст файла.

fgetc( )

Функція *fgetc( )* повертає рядок, що містить один символ з файла в поточній позиції покажчика, або FALSE досягнувши кінця файла. *Синтаксис* функції *fgetc( )*:

string fgetc (int маніпулятор).

Маніпулятор повинен посилатися на відкритий файл, доступний для читання (див. опис функції **is_readable( )** раніше в цій главі).

У наступному прикладі продемонстровано посимвольне читання і виведення файла з використанням функції *fgetc( )*:

```
<?php
$fh = fopen("primer.txt", "r");
while (! feof($fh)) :
$char = fgetc($fh);
print $char;
endwhile;
fclose($fh);
?>
```

fgets( )

Функція **fgets( )** повертає рядок, прочитаний від поточної позиції покажчика у файлі, визначуваному файловим маніпулятором. Файловий покажчик повинен посилатися на відкритий файл, доступний для читання (див. опис функції *is_readable( )*).

Синтаксис функції *fgets( ):*

string fgets (int маніпулятор, int довжина)

Читання припиняється при виконанні однієї з таких умов :

з файла прочитана довжина – 1 байт;

з файла прочитаний символ нового рядка (включається в повертаний рядок);

з файла прочитана ознака кінця файла (EOF).

Якщо ви хочете організувати відрядкове читання файла, передайте в другому параметрі значення, що свідомо перевищує кількість байт у рядку.

Приклад відрядкового читання і виведення файла:

```
$fh = fopen("pastry.txt", "r");
while (! feof($fh));
$line = fgets($fh, 4096);
print $line. "<br>";
endwhile;
fclose($fh) :
```

fgetss( )

Функція *fgetss( )* повністю аналогічна до *fgets( )* за одним винятком – вона намагається видаляти з прочитаного тексту всі теги HTML і PHP:

string fgetss (Int маніпулятор, int довжина [, string дозволені_теги])

Створимо файл *science.html*

```
<html>
<head>
<title>Breaking News - Science</title>
<body>
<h1>Alien lifeform discovered</h1><br>
<b>August 20. 2000</b><br>
Early this morning, a strange new form of fungus was
found growing in the closet of W. J. Gilmore's old
apartment refrigerator. It is not known if powerful
radiation emanating from the tenant's computer monitor
aided in this evolution.
</body>
</html>
```

***Приклад*.** Видалення тегів з файлу HTML перед відображенням в браузері

```
<?php
$fh = fopen("science.html", "r");
while (! feof($fh)) :
print fgetss($fh, 2048);
endwhile;
```

```
fclose($fh);
?>
```

У деяких ситуаціях з файлу віддаляються усі теги, окрім деяких - наприклад, тегів розриву рядків
.

Приклад. *Вибіркове видалення тегів з файлу HTML*

```
<?
$fh = fopen('science.html", "r");
$allowable = "<br>";
while (!feof($fh)) :
print fgetss($fh, 2048, $allowable);
endwhile;
fclose($fh);
?>
```

Функція *fgetss( )* спрощує перетворення файлів, особливо за наявності великої кількості файлів HTML, що відформатували схожим чином.

Читання файла в масив

Функція **file( )** завантажує увесь вміст файла в індексований масив. Кожен елемент масиву відповідає одному рядку файла.

Синтаксис функції **file ( ):**

array file (string файл [, int включення_шляху]).

Якщо необов'язковий третій параметр **включення_шляху** рівний 1, то шлях до файла визначається по відношенню до каталогу файлів, що включаються, зазначеного у файлі *php.ini*.

У прикладі функція *file( )* використовується для завантаження файла *pastry.txt*.

***Приклад*.** Завантаження файла *pastry.txt* функцією *file( )*

```
<?
$file_array = file( "pastry.txt" );
while ( list( $line_num. $line ) = eacht($file_array
)) :
print "<b>Line $line_num:</b> "htmlspecialchars($line
), "<br>\n"
endwhile;
?>
```

Кожен рядок масиву виводиться разом з номером:

```
Line 0: Recipe : Pastry Dough
Line 1: 1 1/4 cups all - purpose flour
Line 2: 3/4 stick (6 tablespoons) unsalted butter,
chopped
```

```
Line 3: 2 tablespoons vegetable shortening
Line 4: 1/4 teaspoon salt
Line 5: 3 tablespoons water
```

Перенаправлення файла у стандартний вихідний потік

Функція **readfile( )** читає вміст файла й направляє його у стандартний вивід (у більшості випадків – у браузер).

Синтаксис функції *readfile( ):*

int readfile (string файл [, int включення_шляху]).

Функція повертає кількість прочитаних байтів. Файл може знаходитися в локальній файловій системі, існувати у вигляді стандартного потоку введення/виводу або представляти файл у видаленій системі, що приймається засобами HTTP або FTP. Параметр файл задається за тими ж правилами, що і у функції **fopen( ).**

Припустимо, у вас є файл *reclama.txt*, вміст якого ви хочете вивести у браузері:

Кафедра Информаційних технологій та систем Луганського національного університету імені Тараса Шевченка оголошує набір студентів за спеціальностями «Програмна інженерія», «Комп'ютерна інженерія»

При виконанні наступного фрагмента увесь вміст *reclama.txt* прямує у стандартний вихідний потік:

```
<?
$output_file = "reclama.txt";
//Направити увесь файл в стандартний вихідний потік
readfile($output_file);
?>
```

Відкриття файлового маніпулятора процесу

popen( )

Разом зі звичайними файлами можна відкривати файлові маніпулятори для взаємодії з процесами на сервері.

Завдання вирішується функцією *popen( )*, яка має такий *синтаксис:*

int popen (string команда, string режим).

Параметр команда визначає виконувану системну команду, а параметр режим описує режим доступу:

```
<?
// Відкрити файл "spices.txt" для запису
$fh = fopen("spices.txt","w");
// Додати декілька рядків тексту
fputs($fh, "Parsley, sage, rosemary\n");
fputs($fh, "Paprika, salt, pepper\n");
fputs($fh, "Basil, sage, ginger\n");
// Закрити маніпулятор
fclose($fh);
// Відкрити процес UNIX grep для пошуку слова Basil у файлі spices.txt
$fh = popen("grep Basil < spices.txt", "r");
// Вивести результат роботи grep
fpassthru($fh); ?>
```

Результат виглядає так:

Basil, sage, ginger

Функція **fpassthru( )** є аналогом до функції **passthru( ).**

pclose( )

Після виконання всіх операцій файл або процес необхідно закрити. Функція *pclose( )* закриває з'єднання з процесом, заданим маніпулятором, за аналогією з тим, як функція **fclose( )** закриває файл, відкритий функцією *fopen( ).*

Синтаксис функції *pclose( ).*

int pclose (int маніпулятор}.

У параметрі маніпулятор передається маніпулятор, отриманий раніше при успішному виклику *popen( ).*

Відкриття з'єднання через Socket

PHP не обмежується взаємодією з файлами і процесами – ви також можете встановлювати з'єднання через сокети. Сокет (*socket*) є програмною абстракцією, що дозволяє встановлювати зв'язок з різними службами іншого комп'ютера.

fsockopen( )

Функція **fsockopen( )** установлює сокетное з'єднання з сервером в Інтернеті через протокол TCP або UDP.

Синтаксис функції *fsockopen( ):*

int fsockopen (string вузол, int порт [, int код_помилки [, string текст_помилки [, int тайм-аут]]]).

Необов'язкові параметри **код_помилки** і **текст_помилки** містять інформацію, яка виводитиметься у разі невдачі при підключенні до серверу. Обидва параметри повинні передаватися по посиланню. Третій необов'язковий параметр, тайм-аут, задає тривалість очікування відповіді від серверу (у секундах).

У прикладі продемонстровано застосування функції *fsockopen( )* для отримання інформації про сервер. Проте перед його розглядом необхідно познайомитися ще з однією функцією – *socket_set_blocking( ).*

UDP (User Datagram Protocol) – комунікативний протокол, не орієнтований на з'єднання.

socket_set_blocking( )

Функція *socket_set_b1ocking( )* дозволяє встановити контроль над тайм-аутом для операцій з сервером:

socket_set_blocking (int маніпулятор, boolean режим)

Параметр маніпулятор задає відкритий раніше сокет, а параметр режим обирає режим, у який перемикається сокет (TRUE для блокуючого режиму, FALSE для неблокуючого режиму).

```
<?
function getthehost($host.$path){
// Відкрити підключення до вузла
$fp = fsockopen($host, 80, &$errno, &$errstr, 30);
// Перейти в блокуючий режим
socket_set_blocking($fp, 1),
// Відправити заголовки
fputs($fp, "GET $path HTTP/1.1\r\n");
fputs ($fp, "Host: $host\r\n\r\n"); $x = 1;
// Отримати заголовки
while($x < 10) :
$headers = fgets ($fp, 4096); print $headers;
$x++; endwhile;
// Закрити маніпулятор
fclose($fp); }
getthehost("www. apress.com", "/"); ?>
```

В результаті виконання ***приклад***у виводиться наступний результат:

HTTP/1.1 200 OK Server : Microsoft - IIS/4.0 Content - location:

http://www.apress.com/ 0efault.htm Date: Sat. 19 Aug 2000 23:03:25 GMT

Content - Type: text/html Accept - Ranges: bytes Last - Modified: Wed. 19 Jul

2000 20:25:06 GMT ETag : "f 0a61666dbff1bf1:34 a5" Content - Length: 1311

pfsockopen( )

Функція **pfsockopen( )** є стійкою *(persistent)* версією **fsockopen( ).** Це означає, що з'єднання не буде автоматично розірване після закінчення сценарію, у якому була викликана функція.

Синтаксис функції **pfsockopen( ):**

int pfsockopen (string вузол, int порт [, int код_помилки [, string текст _помилки [, int тайм-аут]]]).

Залежно від конкретних цілей вашого додатку може бути зручне використання *pfsockopen#00* замість *fsockopen#01*.

Запуск зовнішніх програм

Сценарії PHP також можуть виконувати програми, що знаходяться на сервері. Така можливість особливо часто використовується при адмініструванні системи через web – браузер, а також для зручнішого отримання зведеної інформації про систему.

exec( )

Функція *exec( )* запускає задану програму і повертає останній рядок її вихідних даних.

Синтаксис функції *exec( ):*

string exec (string команда [, string масив [, int повернення]]).

Зверніть увагу: функція *exec( )* тільки виконує команду, не виводячи результатів її роботи. Усі вихідні ці команди можна зберегти в необов'язковому параметрі масив. Крім того, якщо при заданому параметрі масив також задається змінна повернення, останньою привласнюється код повернення виконаної команди.

Приклад. Перевірка зв'язку з сервером із застосуванням функції *exec().*

```
<? php
exec("ping -c 5 www.php.net ", $ping);
```

```
// У Windows - exec("ping - n 5 www.php.net. $ping);
for ($i=0; $i< count($ping);$i++) :
print "<br>$ping[$i]"; endfor; ?>
```

Результат:

PING www.php.net (208.247.106.187) : 56 data bytes
64 bytes from 208.247.106.187: icmp_seq=0 ttl=243 time=66.602 ms
64 bytes from 208.247.106.187: icmp_seq=1 ttl=243 time=55.723 ms
64 bytes from 208.247.106.187: icmp_seq=2 ttl=243 time=70.779 ms
64 bytes from 208.247.106.187: icmp_seq=3 ttl=243 time=55.339 ms
64 bytes from 208.247.106.187: icmp_seq=4 ttl=243 time=69.865 ms
-- -- www.php.net ping statistics --
5 packets transmitted. 5 packets received. 0% packet loss
round - trip min/avg/max/stddev - 55.339/63.662/70.779/6.783 ms

Зворотні апострофи

Існує й інший спосіб виконання системних команд, що не вимагає виклику функцій, – виконувана команда полягає у зворотні апострофи **(` `)**, а результати її роботи відображуються в браузері.

Приклад:

```
$output = `ls`;
print "<pre>$output</pre>";
```

Цей фрагмент виводить у браузер вміст каталогу, у якому знаходиться сценарій.

Внутрішній параметр **ping -c 5** (**-n 5** в системі Windows) задає кількість опитувань серверу.

Якщо ви хочете просто повернути неформатовані результати виконання команди, скористайтеся функцією passthru( ).

passthru( )

Функція *passthru( )* працює майже так само, як *exec( )*, за одним винятком – вона автоматично виводить результати виконання команди.

Синтаксис функції *passthru( ):*

void passthru(string команда [, int повернення]).

Якщо при виклику **passthru( )** передається необов'язковий параметр повернення, цій змінній привласнюється код повернення виконаної команди.

escapeshellcmd( )

Функція *escapeshellcmd( )* екранує всі потенційно небезпечні символи, які можуть бути введені користувачем (наприклад, на формі HTML), для виконання команд **exec( ), passthru( ), system( )** *або* **popen( ).**

Синтаксис:

string escapeshellcmd (string команда).

До призначеного для користувача введення завжди слід ставитися з певною долею обережності, але навіть в цьому випадку користувачі можуть уводити команди, які виконуватимуться функціями запуску системних команд. Розглянемо такий фрагмент:

```
$user_input = 'rm - rf *'; // Видалити батьківський каталог і усі його підкаталоги
exec($user_input); // Виконати $user_input !!!
```

Якщо не здійснювати ніяких заходів обережності, така команда призведе до катастрофи. Щоб уникнути проблеми, можна скористатися функцією *escapeshellcmd( )* для екранування призначеного для користувача уведення :

```
$user_input = 'rm - rf *'; // Видалити батьківський каталог і усі його підкаталоги
exec( escapeshellcmd($user_input));// Екранувати небезпечні символи
```

Функція **escapeshellcmd( )** екранує символ *, запобігаючи катастрофічним наслідкам виконання команди.

Безпека є одним з найважливіших аспектів програмування в середовищі Web, тому присвячено цілу главу цій темі та її ставленню до програмування у PHP.

Робота з файловою системою

У PHP існують функції для перегляду і виконання різних операцій з файлами на сервері. Інформація про атрибути серверних файлів (місцезнаходження, власник і привілеї) часто буває корисною.

basename( )

Функція *basename( )* виділяє ім’я файла з переданого повного імені.

Синтаксис функції *basename( ):*

string basename(string повне_ім'я).

Виділення базового імені файла з повного імені відбувається таким чином:

```
$path = "/usr/local/phppower/htdocs/index.php";
$file = basename($path); // $file = "index.php"
```

Фактично ця функція видаляє з повного імені шлях і залишає тільки ім'я файлу.

getlastmod( )

Функція getlastmod( ) повертає дату і час останньої модифікації сторінки, з якої викликається функція.

Синтаксис функції *getlastmod( ):*

int getlastmod(void).

Повертане значення відповідає формату дати/часу UNIX, і для його форматування можна скористатися функцією date( ). Наступний фрагмент виводить дату останньої модифікації сторінки :

```
echo "Last modified: ".date( "H: i: s a". getlastmod(
));
```

stat( )

Функція *stat( )* повертає індексований масив з детальною інформацією про файл із заданим ім'ям:

array stat(string ім'я_файлу)

В елемент масиву повертається така інформація:

0 Пристрій
1 Індексний вузол (inode)
2 Режим захисту індексного вузла
3 Кількість посилань
4 Ідентифікатор користувача власника
5 Ідентифікатор групи власника
6 Тип пристрою індексного вузла
7 Розмір у байтах
8 Час останнього звернення
9 Час останньої модифікації
10 Час останньої зміни
11 Розмір блоку при введенні/виводі у файловій системі
12 Кількість виділених блоків

Таким чином, якщо ви хочете дізнатися час останнього звернення до файла, зверніться до елемента 8 повертаного масиву.

Приклад:

```
$file = "datafile.txt";
list($dev, $inode, $inodep, $nlink, $uid, $gid, $inodev, $size, $atime, $mtime, $ctime, $bsize) = stat($file);
print "$file is $size bytes. <br>";
print "Last access time: $atime <br>";
print "Last modification time: $mtime <br>";
```

Результат:

popen.php is 289 bytes.
Last access time: August 15 2000 12:00:00
Last modification time: August 15 2000 10:07:18

У цьому прикладі використана конструкція *list ()* для привласнення імен кожному повертаному значенню. Звичайно, з таким же успіхом можна повернути масив, у циклі перебрати елементи і вивести всю необхідну інформацію. Як бачите, функція *stat ( )* дозволяє отримати різні корисні відомості про файл.

Копіювання і перейменування файлів

До інших корисних системних функцій, які можуть виконуватися у сценаріях PHP, належить копіювання і перейменування файлів на сервері. Ці операції виконуються двома функціями: *сміттю( )* і *rename( )*.

copy( )

Скопіювати файл у сценарії PHP можна функцією PHP *copy( ).*

Синтаксис функції *copy( ):*

int **copy** (string джерело, string приймач)

Функція **copy**( ) намагається скопіювати файл джерело у файл–приймач; у разі успіху повертається TRUE, а при невдачі – FALSE. Якщо файл–приймач не існує, функція **copy( )** створює його.

Наступний приклад показує, як створити резервну копію файла за допомогою функції *copy( )*:

```
$data_file = "datal.txt";
copy($data_file,$data_file.'bak') or die("Could not copy $data_file");
```

rename ( )

Функція *rename( )* перейменовує файл. У разі успіху повертається TRUE, а при невдачі – FALSE.

Синтаксис функції *rename( ):*

bool rename (string старе_ім'я, string нове_ім'я).

Приклад перейменування файлу функцією *rename( ):*

```
$data_file = "datal.txt";
rename($data file, $datafile'.old') or die ("Could not rename $data file");
```

Видалення файлів

unlink( )

Функція *unlink( )* видаляє файл із заданим ім'ям.

Синтаксис:

int unlink (string файл).

Якщо ви працюєте з PHP у системі Windows, при використанні цієї функції іноді виникають проблеми. У цьому випадку можна скористатися описаною вище функцією *system( )* і видалити файл командою *DOS* **del**:

system ("del filename.txt");

Робота з каталогами

Функції PHP дозволяють переглядати вміст каталогів і переміщуватися по них. У прикладі зображена типова структура каталогів в системі UNIX.

Приклад. Типова структура каталогів

```
drwxr - xr - x 4 root wheel 512 Aug 13 13:51 book/
drwxr - xr - x 4 root wheel 512 Aug 13 13:51 code/
-- rw - r--r-- 1 root wheel 115 Aug 4 09:53 index.html
drwxr - xr - x 7 root wheel 1024 Jun 29 13:03 manual/
-- rw - r--r-- 1 root wheel 19 Aug 12 12:15 test.php
```

dirname( )

Функція *dirname( )* доповнює **basename( )** – вона витягає шлях з повного імені файла.

Синтаксис функції *dirname( ):* **string dirname (string шлях)**.

Приклад використання ***dirname( )*** для витягання шляху з повного імені:

```
$path = "/usr/locla/phppower/htdocs/index.php";
$file = dirname($path);
// $file = "usr/local/phppower/htdocs"
```

Функція **dirname( )** іноді використовується у поєднанні зі змінною **$SCRIPT_FILENAME** для отримання повного шляху до сценарію, з якого виконується команда:

$dir - dirname($SCRIPT_FILENAME);

is_dir( )

Функція is_dir( ) перевіряє, чи є файл із заданим ім'ям каталогом: **bool is_dir (string ім'я_файлу)**

У наступному прикладі використовується структура каталогів з попередніх прикладів:

```
$isdir = is_dir("index.html"); // Повертає FALSE
$isdir = is_dir("book");  // Повертає TRUE
```

mkdir()

Функція *mkdir( )* робить те ж, що й однойменна команда UNIX, – вона створює новий каталог.

Синтаксис функції *mkdir( ):*

int mkdir (string шлях, int режим).

Параметр *шлях* визначає доступ для створення нового каталогу. Не забудьте завершити параметр ім'ям нового каталогу! Параметр **режим** визначає дозволи, що призначаються створеному каталогу.

opendir( )

Подібно до того, як функція **fopen( )** відкриває маніпулятор для роботи із заданим файлом, функція *opendir( )* відкриває маніпулятор для роботи з каталогом.

Синтаксис функції *opendir( ):* **int opendir (string шлях)**.

closedir( )

Функція *closedir*( ) закриває маніпулятор каталогу, переданий як параметр.

Синтаксис функції *closedir( )*:

void closedir(int маніпулятор_каталогу).

readdir( )

Функція *readdir( )* повертає черговий елемент заданого каталогу.

Синтаксис: **string readdir(int маніпулятор_каталогу)**.

За допомогою цієї функції можна легко вивести список усіх файлів і підкаталогів, що знаходяться в поточному каталозі :

```
$dh = opendir(' . );
while ($file = readdir($dh)) :
print "$file <br>"; endwhile;
closedir($dh);
```

chdir( )

Функція *chdir( )* працює так само, як команда **UNIX cd**, – вона здійснює перехід у каталог, заданий параметром.

Синтаксис функції *chdir( )*: **int chdir (string каталог)**.

Приклад переходимо в підкаталог **book/** і виводимо його вміст:

```
$newdir = "book";
chdir($newdir) or die("Could not change to directory
($newdir)");
$dh = opendir(' . ');
print "Files:";
while ($file = readdir($dh));
print "$file <br>";
endwhile;
closedir($dh);
```

rewinddir( )

Функція *rewlnddir( )* переводить покажчик поточної позиції в початок каталогу, відкритого функцією *opendir( )*

Синтаксис функції *rewinddir( ):*

void rewinddir (int нанипулятор_каталогу).

Відображення і зміна характеристик файлів (додатково)

У кожного файлу в системах сімейства UNIX є три важливі характеристики:

– приналежність групі;

– власник;

– дозволи (permissions).

Усі ці характеристики можна змінити за допомогою відповідних функцій PHP. Функції, описані в цьому розділі, не працюють у системах сімейства Windows.

Якщо у вас немає досвіду роботи в операційних системах UNIX, інформацію про характеристики файлової системи UNIX можна отримати за адресою *http://sunsite.auc.dk/linux-newbie/FAQ 2.htm.* Теми належності групі, володіння і дозволів розглядаються в розділі 3.2.6.

chgrp( )

Функція *chgrp( )* намагається змінити групу, якій належить заданий файл.

Синтаксис функції *chgrp( )*:

int chgrp (string ім'я_файлу, mixed група).

filegroup( )

Функція *filegroup( )* повертає ідентифікатор групи власника файла із заданим ім'ям або FALSE у разі помилки.

Синтаксис функції *filegroup( )*:

int filegroup (string ім'я_файлу).

chmod( )

Функція *chmod( )* змінює дозволи файлу із заданим ім'ям.

Синтаксис функції *chmod( )*:

int chmod (string ім'я_файлу, int дозволи).

Дозволи задаються у вісімковій системі. Специфіка завдання параметра функції *chmod ( )* в такому прикладі:

```
chmod("data_file.txt", g+r); // Не працює
chmod("data_file.txt", 766); // Не працює
chmod("data_file.txt", 0766); // Працює
```

fileperms( )

Функція *fileperms( )* повертає дозволи файла із заданим ім'ям або FALSE у разі помилки.

Синтаксис функції *fileperms( )*:

int fileperms (string ім'я_файла).

chown( )

Функція *chown( )* намагається змінити власника файла. Право зміни власника файла надається тільки привілейованому користувачеві.

Синтаксис функції *chown( )*:

int chown (string ім'я_файлу, mixed користувач).

fileowner( )

Функція *fileowner( )* повертає ідентифікатор користувача для власника файла із заданим ім'ям.

Синтаксис функції *fileowner( )*:

int fileowner (string ім'я_файла)

Контрольні питання:

1. За допомогою яких функцій здійснюється перевірка існування файла та можливість виконання з ним операцій читання/запису?
2. У яких формах можуть задаватися параметри файла?
3. Перерахуйте значення, що визначають режим відкриття файла.
4. Навести приклади відкриття/закриття файла за допомогою функцій **fopen( ), fclose( )**.
5. За допомогою яких функцій проводиться запис/читання з файла?
6. Яка функція читає вміст файла і направляє його в стандартний вивід у PHP?
7. Яка функція вирішує завдання відкриття файлових маніпуляторів для взаємодії з процесами на сервері?
8. Сформувати поняття UDP. Навести приклад використання функції для отримання інформації про сервер.

Лекція 8. MySQL

План:

1. Використання функції роботи з файловою системою при складанні складних сценаріїв.
2. Реляційні бази даних.
3. Гібридні СУБД.
4. Індекси.
5. Способи завдання первинного ключа.
6. Робота з сервером MySQL.
7. Основні особливості роботи з MySQL. Мова SQL.

MySQL – це одна з найпопулярніших і найпоширеніших СУБД (система управління базами даних) в Інтернеті. Вона не призначена для роботи з великими обсягами інформації, але її застосування ідеальне для інтернет сайтів, як невеликих, так і досить великих.

MySQL відрізняться хорошою швидкістю роботи, надійністю, гнучкістю. Робота з нею, як правило, не викликає великих труднощів. Підтримка сервера MySQL автоматично включається в постачання PHP.

Важливим чинником є її безкоштовність. MySQL поширюється на умовах загальної ліцензії GNU (GPL, GNU Public License).

Раніше для довготривалого зберігання інформації ми працювали з файлами. Завдання тривалого зберігання інформації дуже часто зустрічається у програмуванні Web – додатків: підрахунок відвідувачів у лічильнику, зберігання повідомлень у форумі, видалене управління змістом інформації на сайті і т. ін.

Між тим, професійні прийоми роботи з файлами дуже трудомісткі: необхідно піклується про вміщення в них інформації, про її сортування, витягання, при цьому не треба забувати, що всі ці дії відбуватимуться на сервері хост-провайдера, де з дуже великою вірогідністю постає один з варіантів Unix – отже, треба так само піклуватися про права доступу до файлів і їх розміщення.

При цьому обсяг коду значно зростає, і зробити помилку в програмі дуже просто.

Усі ці проблеми вирішує використання бази даних. Бази даних самі піклуються про безпеку інформації і її сортування і дозволяють витягати і розміщувати інформацію за допомогою одного рядка. Код з використанням бази даних виходить компактнішим, і відлагоджувати його набагато легше. Крім того, не треба забувати і про швидкість – вибірка інформації з бази даних здійснюється значно швидше, ніж з файлів.

Застосування на PHP, що використовує для зберігання інформації базу даних (зокрема MySql), завжди працює швидше за застосування, побудоване на файлах. Річ у тому, що бази даних написані на мові C++, і написати на PHP програму, яка працювала б з жорстким диском ефективніше за базу даних, - завдання нерозв'язне.

Таким чином, основна гідність бази даних полягає в тому, що вона бере на себе всю роботу з жорстким диском і робить це дуже ефективно.

Реляційні бази даних

Завдання тривалого зберігання й обробки інформації з'явилося практично відразу з появою перших комп'ютерів. Для вирішення цього завдання у кінці 60-х років булj розробленj спеціалізовані програми, що дістали назву систем управління базами даних (СУБД).

Взаємодія з базою даних відбувається за допомогою Системи Управління Базою Даних (СУБД), яка розшифровує запити і здійснює операції з інформацією в базі даних. Тому правильніше було б говорити про запит до СУБД і про взаємодію з СУБД з Web – додатками. Але оскільки це дещо ускладнює сприйняття, далі скрізь ми говоритимемо ,,база даних", маючи на увазі при цьому СУБД.

Існують такі різновиди баз даних :
– ієрархічні;
– реляційні;
– об'єктно-орієнтовані;
– гібридні.

Ієрархічна база даних заснована на деревовидній структурі зберігання інформації. У цьому сенсі ієрархічні бази даних дуже нагадують файлову систему комп'ютера.

У реляційних базах даних дані зібрані в таблиці, які у свою чергу складаються зі стовпців і рядків, на перетині яких розташовані осередки. Запити до таких баз даних повертає таблицю, яка повторно може брати участь у наступному запиті. Дані в одних таблицях, як правило, пов'язані з даними інших таблиць, звідки і постала назва „реляційні".

В об'єктно-орієнтованих базах даних дані зберігаються у вигляді об'єктів. З об'єктно-орієнтованими базами даних зручно працювати, застосовуючи об'єктно-орієнтоване програмування. Проте, на сьогодні такі бази дан-ных ще не досягли популярності реляційних, оскільки доки значно поступаються їм в продуктивності.

Гібридні СУБД поєднують у собі можливості реляційних і об'єктно-орієнтованих баз даних.

У Web -приложениях, як правило, використовуються реляційні бази даних. Ми розглядатимемо приклад бази даних, на якій заснована більшість форумів. У цій базі зберігається інформація про авторів форуму (authors), про назви форумів (forums), про теми форуму (themes) і, власне, самі повідомлення (posts). Таким чином, наша база даних включатиме такі таблиці:

Таблиця 8.1 – Таблиці бази даних Forum

authors
forums
posts
themes

Модель реляційної бази даних представляє дані у вигляді таблиць, розбитих на рядки і стовпці, на перетині яких знаходяться дані. Приклад такої таблиці показаний в таблиці.8.2:

Таблиця 8.2 – Структура реляційної бази даних

	ряд		
рядок	id_forum	name	Description
	1	дизайн	Обговорюються питання дизайну

	2	MySQL	Обговорюються питання, пов'язані з MySQL
	3	PHP	Обговорюються питання, пов'язані з PHP
	4	різне	Інші питання

У таблиці. 8.2 наведено приклад таблиці *forums* бази даних великого форуму, у якому є кілька розділів, присвячених різним етапам побудови Web – додатків. Кожен рядок цієї таблиці є одним розділом форуму. Чотири рядки таблиці є усім форумом.

Кожен стовпець таблиці forums представляє один елемент даних для кожного з форумів. Стовпець id_forum містить унікальний ідентифікатор форуму, стовпець name містить назву форуму і стовпець description містить короткий опис проблеми, що обговорюється на форумі.

Коротко особливості реляційної бази даних можна описати таким чином:

- дані зберігаються в таблицях, що складаються із стовпців і рядків;
- на перетині кожного стовпця і рядка стоїть однакове значення;
- у кожного стовпця є своє ім’я, яке слугує його назвою, і всі значення в одному стовпці мають один тип. Наприклад, у стовпці **id_forum** усі значення мають цілочисельний тип, а в рядку **name** – текстовий;
- стовпці розташовуються в певному порядку, який визначається при створенні таблиці, на відміну від рядків, які розташовуються в довільному порядку. У таблиці може не бути не одного рядка, але обов’язково має бути хоч би один стовпець;
- запити до бази даних повертають результат у вигляді таблиць, які теж можуть поставати як об’єкт запитів.

Індекси

Індекс – це відсортований список значень полів, призначений для прискорення пошуку в базі даних.

Цікаві, як правило, не самі індекси, а унікальні індекси.

Унікальний індекс є списком значень, у якому кожне значення унікальне.

У базі даних Forum, треба зауважити, що дата додавання повідомлення у форум також не є унікальною, оскільки кілька учасників форуму можуть додати свої повідомлення одночасно.

Первинні ключі

Первинний ключ *(primary key)* є одним з прикладів унікальних індексів і застосовується для унікальної ідентифікації записів таблиці. Ніякі з двох записів таблиці не можуть мати однакових значень первинного ключа. Первинний ключ зазвичай скорочено означають як PK *(primary key)*.

Як ми вже говорили, у реляційних базах даних практично завжди різні таблиці логічно пов'язані один з одним. Первинні ключі якраз використовуються для однозначної організації такого зв'язку.

Наприклад, у базі даних **Forum** таблиці *themes i posts* пов'язані між собою таким чином (Рис 8.1)

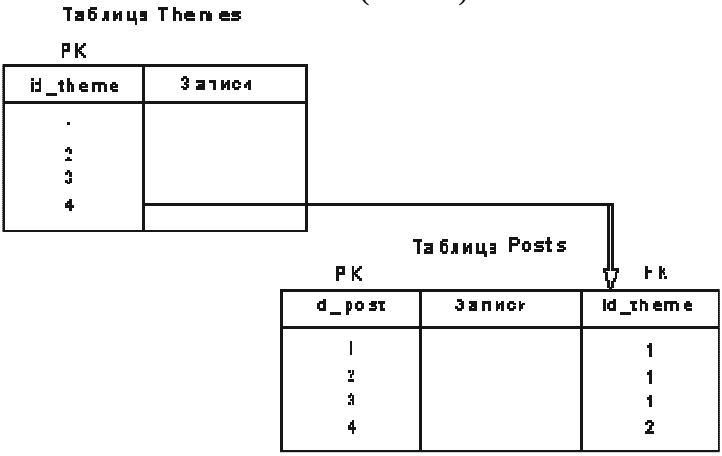


Рис. 8.1. Зв'язки у форумі

Первинним ключем таблиці **themes** є id_theme, а таблиці **posts** – **id_post**. Зверніть увагу, що поле **id_theme** наявне і в таблиці **posts**. Кожне значення цього поля в таблиці **posts** є зовнішнім ключем (у цьому випадку це зовнішній ключ для первинного ключа таблиці **themes)**.

Зовнішній ключ скорочено означають як FK *(foreign key).*Як видно з рис. 8.1, зовнішній ключ посилається на первинний ключ таблиці **themes,** встановлюючи однозначний логічний зв'язок між записами таблиць **themes** і **posts**. Інакше кажучи, якщо зовнішній ключ для запису (повідомлення) з PK=1 у таблиці **posts** має значення зовнішнього ключа, що дорівнює 1, то це означає, що це повідомлення належить до теми з PK=1 таблиці **themes.**

Способи завдання первинного ключа

За способом завдання первинних ключів розрізняють логічні (природні) ключі й сурогатні (штучні).

Для логічного завдання первинного ключа треба обрати в базі даних те, що природним чином визначає запис.

Якщо відповідних прикладів для природного завдання первинного ключа не знаходиться, користуються сурогатним ключем. Сурогатний ключ є додатковим полем у базі даних, призначене для забезпечення записів первинним ключем.

Навіть якщо в базі даних знаходиться природний первинний ключ, краще використовувати сурогатні ключі, оскільки їх застосування дозволяє абстрагувати первинний ключ від реальних даних. У цьому випадку полегшується робота з таблицями, оскільки сурогатні ключі не пов’язані ні з якими фактичними даними цієї таблиці.

Первинному ключу можна присвоїти атрибут auto_increment, що дозволяє автоматично генерувати унікальний ключ, якщо його тип є цілочисельним. При вставленні запису до бази даних значення ключа виставляється рівним нулю, MySQL автоматично обчислює максимальний номер первинного ключа, збільшує його на одиницю і привласнює це значення первинному ключу нового запису.

Нормалізація бази даних

Схемою бази даних називається структура зв’язків між полями й таблицями.

Нормалізацією схеми бази даних називається процедура, вироблювана над базою даних з метою видалення в ній надмірності.

Для того, щоб краще зрозуміти визначення нормалізації, розглянемо приклад. У таблиці зазначені прізвища співробітників і їх професії :

Таблиця 8.3. Приклад надмірності в таблицях бази даних

Професія	Співробітник
"Інженер"	Гусев И.К.
"Інженер"	Іванов П. В.
"Робітник"	Іванов К.Л.
"Робітник"	Дружків П. К.
"Робітник"	Фомичев В. М.

Для нормалізації необхідно розбити цю таблицю на дві – для професій (див. таблицю. 8.4) і для прізвищ співробітників (див. таблицю 8.5).

Таблиця 8.4. Таблиця професій

Професія	Первинний ключ
""Інженер"	1
""Робітник"	2

Таблиця 8.5. Таблиця співробітників

Професія (зовнішній ключ)	Співробітник
1	Гусев И.К.
1	Іванов П. В.
2	Іванов К.Л.
2	Дружків П. К.
2	Фомичев В. М.

Тепер кожен тип професії позначений унікальним числом, і рядок у базі даних наявний тільки в єдиному примірнику: в таблиці професій. Розмір поля „професія” зменшився, оскільки рядок займає більше пам’яті, ніж число.

У теорії баз даних говориться про те, що схема бази даних має бути повністю нормалізована. При роботі з повністю нормалізованими базами даних необхідно застосовувати дуже складні SQL – запити, що призводить до зворотного ефекту – уповільнення роботи бази даних. Тому іноді для спрощення запитів навіть вдаються до зворотної процедури – денормалізації.

MySQL

MySQL (http://www.mysql.com) – надійна СУБД на базі SQL, розроблена й супроводжувана фірмою Т. с.Х DataKonsultAB (Стокгольм, Швеція). Починаючи з 1995 року, MySQL стала однією з найпоширеніших СУБД у світі, що частково зумовлено її швидкістю, надійністю і гнучкою ліцензійною політикою (див. нижче).

Завдяки гарним характеристикам і великому набору стандартних інтерфейсних функцій, дуже простих у використанні, MySQL стала найпопулярнішим засобом для роботи з базами даних у PHP.

MySQL поширюється на умовах загальної ліцензії GNU (GPL, GNU Public License). Повний опис поточної ліцензійної політики MySQL наведено на сайті MySQL (http://www.mysql.com).

Робота з сервером MySQL

При підключенні до серверу MySQL за допомогою програми mysql треба ввести ім'я користувача, і, як правило, пароль. Якщо сервер і клієнт знаходяться на різних машинах, необхідно також вказати ім'я хоста, на якому запущений сервер MySQL:

```
shell> mysql - h host - u user - p
```

Після цього на екрані з'явиться запит **Enter password:,** і вам треба буде ввести свій пароль. Якщо з'єднання відбулося нормально, то на екрані з'являється така інформація і мітка командного рядка **mysql>:**

```
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 459 to server version :
Type 'help' for help.
mysql>
```

Поява мітки **mysql>** означає, що програма *mysql* готова до роботи.

Від'єднатися від серверу можна в будь-який момент, набравши команду QUIT :

```
mysql> QUIT
Bye
```

Як правило, на тільки що встановленому MySQL на локальній машині доступ здійснюється без уведення пароля і хоста, уведенням команди **mysql** у командному рядку.

Потім вводяться запити і команди.

Приклад команда, що запрошує у сервера інформацію про його версію і поточну дату *:*

```
mysql> SELECT VERSION(), CURRENT_DATE;
```

Відповіддю MySQL на цей запит буде наступна таблиця:

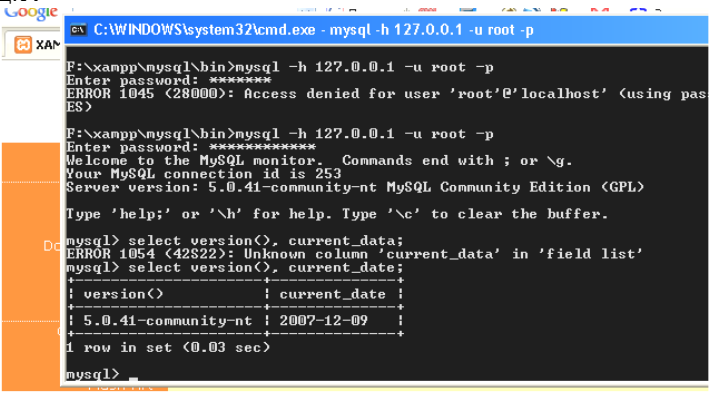


Рис 8.2. Відповідь бази даних

Основні особливості роботи з MySQL

Команда, що надсилається серверу, зазвичай складається зі SQL – виразу, за який йде *крапка з комою*. З правила є кілька винятків, які після команди *крапка з комою* не ставляться, наприклад, команда **QUIT**;

MySQL виводить результати запиту у вигляді таблиці;

Після виведення таблиці з результатами запиту, mysql повідомляє кількість повернутих рядків і час виконання запиту. Це зручно, оскільки дозволяє оцінити як продуктивність серверу, так і ефективність виконуваного запиту;

Після виведення результатів запиту й часу його виконання, *mysql* виводить новий рядок **mysql>,** що означає готовність до виконання нових команд.

Команди **MySQL** не чутливі до регістру.

MySQL дозволяє на одному рядку розмістити кілька команд, але кожна з них повинна закінчуватися крапкою з комою.

Приклад:

```
mysql> SELECT VERSION(); SELECT NOW();
```

На такий запит ми отримаємо такий результат:

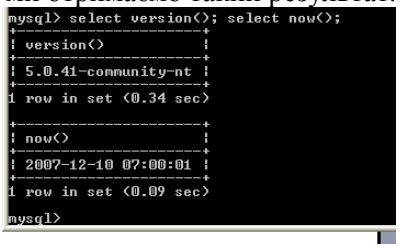


Рис 8.3. Відповідь на запит

Команди можна розміщувати в різних рядках

```
mysql> SELECT USER()
    -       -> CURRENT_DATE;
```

Результат:

```
user()                current_date
root@localhost        | 2007-12-10
  1 row in set (0.08 sec)
```

Після переходу на новий рядок мітка командного рядка змінилася з **mysql>** на **->**. Таким чином *mysql* показує, що завершеного запиту не отримано і вона чекає кінця запиту. Ця мітка дуже корисна, оскільки дозволяє уникнути окремих помилок. Наприклад, якщо не поставити крапку з комою у кінці запиту, mysql повідомить про це, вивівши мітку ->:

```
mysql> select user()
  ->
```

MySQL можна використовувати як калькулятор.

Приклад. *Уведіть запит:*

```
mysql> select cos(pi()/10), (2*5) - 5;
```

Мова SQL

Структурована мова запитів **SQL** дозволяє проводити різні операції з базами даних: створювати таблиці, уміщувати, онов-

лювати і видаляти з них дані, проводити запити з таблиць і т.ін. Далі ми послідовно розглянемо всі ці оператори.

Ми дотримуватимемося діалекту **SQL** характерного для СУБД MySQL тому не усі запити можуть виконуватися для інших баз даних.

Команди SQL

create database; use; create table; describe; alter table; drop table; drop database; insert into.values; delete; select; update; show

Команди SQL не чутливі до регістру, але традиційно вони набираються прописними буквами.

Типи полів бази даних

Список найбільших типів, що часто зустрічаються, наведено в таблицях 8.6. – 8.8. Для багатьох типів даних задається максимальна ширина відображення, що зазначається в дужках, яку ми далі позначатимемо символом *max*. Наприклад, запис INT(2) означає, що значення цього поля не може перевищувати 100.

До числових типів належать цілі числа й числа з плаваючою точкою. Для чисел з плаваючою точкою, окрім максимальної ширини відображення, можна також зазначити число значущих цифр після коми, що далі означає символом P.

Таблиця 8.6. Числові типи

Тип	Опис
TINYINT[(max)]	Дуже маленькі цілі числа діапазону – 127.128.
SMALLINT[(max)]	Маленькі цілі числа діапазону – 32768.32767.
MEDIUMINT[(max)]	Середні цілі числа.
INT[(max)]	Звичайні цілі числа.
FLOAT[(max, P)]	Числа з плаваючою точкою одинарної точності.
DOUBLE[(max, P)]	Числа з плаваючою точкою подвійної точності.
DECIMAL[(max, P)]	Числа з плаваючою точкою, приведені до типу char.

Таблиця 8.7. Типи дати і часу

Тип	Опис
DATE	Дата у форматі ГГГГ-ММ-ДД.
TIME	Час у форматі ЧЧ-ММ-СС.
DATETIME	Дата і час у форматі ГГГГ-ММ-ДД ЧЧ-ММ-СС.
YEAR	Рік у форматі ГГ або ГГГГ.
TIMESTAMP	Мітка часу для відліків за транзакціями у форматі ГГГГ-ММ-ДД ЧЧ-ММ-СС.

Таблиця 8.8. Строкові типи

Тип	Опис
CHAR (len)[BINARY]	Рядки з довжиною len, яке не перевищує 255 символів. Ключове слово BINARY вказує на те, що дані повинні оброблятися незалежно від регістру.
CHAR	Синонім CHAR(1).
VARCHAR (len)[BINARY]	Синонім CHAR(len) за винятком того, що рядки можуть бути довільної довжини.
TEXT	Рядки з максимальною довжиною символів рівної 65535. Дані цього типу чутливі до регістру.
BLOB	Двійкові рядки з максимальною довжиною символів рівної 65535. Тип BLOB (binary large object – великий двійковий об'єкт) призначений для зберігання двійкових даних, зокрема й зображень і звукових послідовностей.

Команди

CREATE DATABASE

Ця команда створює нову базу даних:

CREATE DATABASE db_name;

Тут *db_name* є ім'ям створюваної бази даних.

Приклад. Для того, щоб створити нову базу даних forum, наберіть у рядку-запрошенні клієнта MySQL *mysql*> цю команду і вкажіть назву бази даних:

```
mysql> CREATE DATABASE forum;
```

Кожен запит **MySQL** завершується крапкою з комою.

При успішному виконанні команди **MySQL** видасть рядок, у якому повідомляється, що цей запит виконаний успішно й показаний час, витрачений на виконання запиту.

Для того, щоб переконається, що база даних успішно створена, можна виконати команду SHOW DATABASES, яка покаже, які бази даних існують на вашому комп'ютері:

mysql> SHOW DATABASES;

Як бачимо, серед різних баз даних на комп'ютері автора наявна і тільки що створена база даних **forum** :

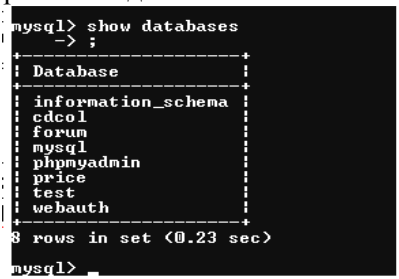


Рис 8.4. Відповідь бази даних forum

Команда **SHOW DATABASES** є внутрішньою командою **MySQL,** відсутньою у стандарті **SQL** і не підтримується іншими базами даних.

У **MySQL** при установці наявні дві бази даних: *test* і *mysql*. В останній зберігається системний каталог, що описує внутрішню структуру СУБД MySQL.

USE

Для того, щоб розпочати роботу з таблицями, необхідно повідомити **MySQL**, з якою базою даних ви маєте намір працювати. Це здійснюється за допомогою команди USE :

USE db_name;

Тут db_name - назва обраної бази даних.

Приклад. Оберемо створену базу *forum :*

```
mysql> USE forum;
Database changed;
```

CREATE TABLE

Команда CREATE TABLE створює нову таблицю в обраній базі даних і яка у простому випадку має такий *синтаксис*:

CREATE TABLE table_name [(create_definition, ..)]

Тут *table_name* – ім'я створюваної таблиці.

Приклад. Створимо першу таблицю бази даних forum, яка називається authors і містить різні дані про зареєстрованих відвідувачів форуму : нік (*name*), пароль *(passw*), e–mail (*email*), Web–адресу сайту відвідувача *(url)*, номер ICQ *(icq)*, відомості про відвідувача *(about)*, рядок що містить шлях до файла фотографії відвідувача *(photo)*, час додавання запиту *(time)*, останнім часом відвідування форуму *(last_time)*, статус відвідувача – чи є він модератором, адміністратором або звичайним відвідувачем *(statususer)*. Окрім перерахованих полів, у таблиці є поле **id_author**, що є первинним ключем таблиці. SQL – запит, що створює цю таблицю наведено в лістингу*:*

```
mysql> CREATE TABLE authors (id_author int(6) NOT NULL
auto_increment, name text, passw text, email text, url text,
icq text, about text, photo text, time datetime default NULL,
last_time datetime default NULL, themes int(10) default NULL,
statususer int(2) default NULL, PRIMARY KEY (id_author))
TYPE=MyISAM;
```

Виконавши SQL – команду SHOW TABLES, можна переконатися, що таблиця успішно створена

```
+-----------------+
| Tables_in_forum |
+-----------------+
| authors         |
+-----------------+
1 row in set (0.05 sec)
```

Рис 8.5. Таблиця в базі даних

Аналогічним чином створимо таблицю **forums**, у якій знаходяться дані про розділи форуму.

У таблиці **forums** наявні такі поля: первинний ключ *(id_forum)*, назва розділу *(name)*, правила форуму *(rule)*, короткий опис форуму *(logo)*, порядковий номер *(pos)*, прапор, що набуває значення 1, якщо форум прихований і 0, якщо загальнодоступний *(hide*).

Команда SHOW TABLES показує, що таблиці в базі даних успішно створені.

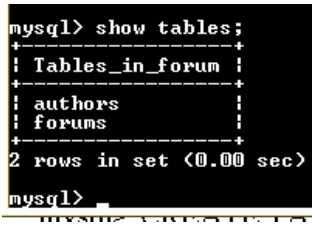


Рис 8.6. Таблиця в базі даних

DESCRIBE

Команда *DESCRIBE* показує структуру створених таблиць і має такий синтаксис:

DESCRIBE table_name

Тут table_name – ім'я таблиці структура якої запрошується.

Команда DESCRIBE не належить до стандарту SQL і є внутрішньою командою СУБД MySQL.

Приклад. Структура таблиці *authors.*

```
mysql> DESCRIBE authors;
```

Після виконання цієї команди інтерпретатор *mysql* виведе наступну таблицю

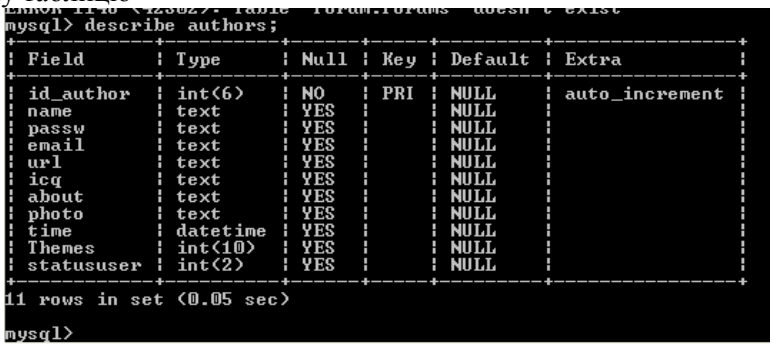


Рис 8.7. Таблиця команди authors

ALTER TABLE

Команда ALTER TABLE дозволяє змінити структуру таблиці. Ця команда дозволяє додавати і видаляти стовпці, створювати та знищувати індекси, перейменовувати стовпці і саму таблицю. Команда має такий *синтаксис*:

ALTER TABLE table_name alter_spec.

Параметр *alter_spec* має значення, представлені в таблиці 8.9.

Таблиця 8.9. Значення параметра alter_spec

Синтаксис	Опис команди
ADD create_definition [FIRST|AFTER column_name]	Додавання нового стовпця create_definition. create_definition є назвою нового стовпця і його типом. Конструкція FIRST додає новий стовпець перед стовпцем column_name. Конструкція AFTER додає новий стовпець після стовпця column_name. Якщо місце додавання не вказане, за умовчанням стовпець додається в кінець таблиці.
ADD INDEX [index_name] (index_col_name,..)	Додавання індексу index_name для стовпця index_col_name. Якщо ім'я індексу index_name не вказується, йому привласнюється ім'я співпадаюче з ім'ям стовпця index_col_name.
ADD PRIMARY KEY (index_col_name,..)	Робить стовпець index_col_name або групу стовпців первинним ключем таблиці.
CHANGE old_col_name new_col_name type	Зміна стовпця з ім'ям old_col_name на стовпець з ім'ям new_col_name і типом type.
DROP col_name	Видалення стовпця з ім'ям col_name.
DROP PRIMARY KEY	Видалення первинного ключа таблиці.
DROP INDEX index_name	Видалення індексу index_name.

Додамо в таблицю forums новий стовпець *test*, розмістивши його після стовпця *name*.

```
mysql> ALTER TABLE forums ADD test int(10) AFTER name;
```

Виконавши команду **DESCRIBE forums**, можна побачити, що стовпець *test* успішно доданий після стовпця *name*

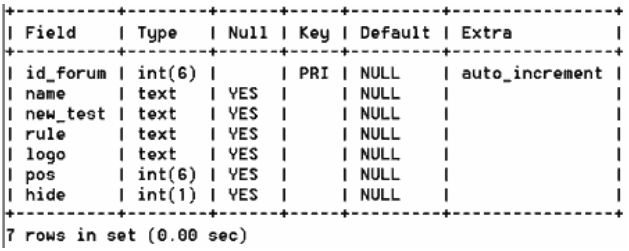
```
mysql> describe forums;
+----------+---------+------+-----+---------+----------------+
| Field    | Type    | Null | Key | Default | Extra          |
+----------+---------+------+-----+---------+----------------+
| id_forum | int(6)  | NO   | PRI | NULL    | auto_increment |
| name     | text    | YES  |     | NULL    |                |
| test     | int(10) | YES  |     | NULL    |                |
| rule     | text    | YES  |     | NULL    |                |
| logo     | text    | YES  |     | NULL    |                |
| pos      | int(6)  | YES  |     | NULL    |                |
| hide     | int(1)  | YES  |     | NULL    |                |
+----------+---------+------+-----+---------+----------------+
7 rows in set (0.00 sec)

mysql>
```

Рис 8.8. Таблиця команди DESCRIBE forums

Приклад. Перейменуємо створений стовпець *test* у текстовий стовпець *new_test*

```
mysql> ALTER TABLE forums CHANGE test new_test text;
```

Стовпець успішно перейменований:



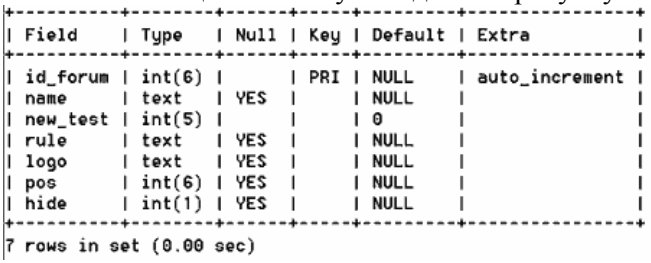
```
+----------+--------+------+-----+---------+----------------+
| Field    | Type   | Null | Key | Default | Extra          |
+----------+--------+------+-----+---------+----------------+
| id_forum | int(6) |      | PRI | NULL    | auto_increment |
| name     | text   | YES  |     | NULL    |                |
| new_test | text   | YES  |     | NULL    |                |
| rule     | text   | YES  |     | NULL    |                |
| logo     | text   | YES  |     | NULL    |                |
| pos      | int(6) | YES  |     | NULL    |                |
| hide     | int(1) | YES  |     | NULL    |                |
+----------+--------+------+-----+---------+----------------+
7 rows in set (0.00 sec)
```

Рис 8.9. Перейменований стовпець test

При зміні тільки типу стовпця, а не його імені, вказівка імені все-одно потрібна, хоча в цьому випадку воно фактично повторюватиметься.

```
mysql> ALTER TABLE forums CHANGE new_test new_test int(5)
not null;
```

Результат виконання цього запиту наведено на рисунку:



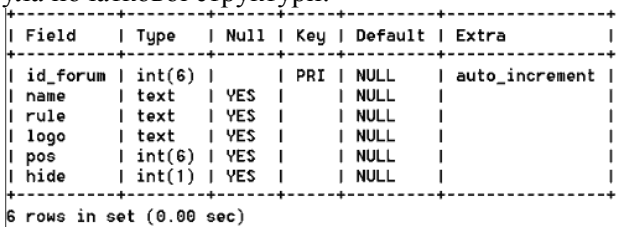
```
+----------+--------+------+-----+---------+----------------+
| Field    | Type   | Null | Key | Default | Extra          |
+----------+--------+------+-----+---------+----------------+
| id_forum | int(6) |      | PRI | NULL    | auto_increment |
| name     | text   | YES  |     | NULL    |                |
| new_test | int(5) |      |     | 0       |                |
| rule     | text   | YES  |     | NULL    |                |
| logo     | text   | YES  |     | NULL    |                |
| pos      | int(6) | YES  |     | NULL    |                |
| hide     | int(1) | YES  |     | NULL    |                |
+----------+--------+------+-----+---------+----------------+
7 rows in set (0.00 sec)
```

Рис 8.10. Результат зміни типу стовпця test

Тепер видалимо стовпець *new_test*:

```
mysql> ALTER TABLE forums DROP new_test;
```

Як видно з малюнка, після видалення цього стовпця таблиця *forums* набула початкової структури:



```
+----------+--------+------+-----+---------+----------------+
| Field    | Type   | Null | Key | Default | Extra          |
+----------+--------+------+-----+---------+----------------+
| id_forum | int(6) |      | PRI | NULL    | auto_increment |
| name     | text   | YES  |     | NULL    |                |
| rule     | text   | YES  |     | NULL    |                |
| logo     | text   | YES  |     | NULL    |                |
| pos      | int(6) | YES  |     | NULL    |                |
| hide     | int(1) | YES  |     | NULL    |                |
+----------+--------+------+-----+---------+----------------+
6 rows in set (0.00 sec)
```

Рис 8.11. Результат видалення стовпця

DROP TABLE

Команда *DROP TABLE* призначена для видалення однієї або кількох таблиць:

DROP TABLE table_name [, table_name,..].

Приклад. Для видалення таблиці forums треба виконати такий *SQL -запит:*

```
mysql> DROP TABLE forums;
```

DROP DATABASE

Команда *DROP DATABASE* видаляє базу даних з усіма таблицями такими, що належать до її складу:

DROP DATABASE database_name.

Приклад. Видалимо базу даних *forum :*

```
mysql> DROP DATABASE forum;
```

INSERT INTO.VALUES

Команда *INSERT.VALUES* вставляє нові записи в начвну таблицю. *Синтаксис* команди:

INSERT INTO table_name VALUES (values,.).

Після оператора VALUES у дужках через кому перераховуються значення відповідних полів таблиці відповідно до їх типів.

Приклад. Давайте вставимо в базу даних authors кілька записів, у яких розташована інформація про зареєстрованих відвідувачів форуму:

```
   mysql> INSERT INTO authors VALUES (1, 'Maks ', '123 ', '
maks@mail.ru
   ' ' www.softtime.ru ', '', 'програміст', '', '', '', 0, 0);
   mysql> INSERT INTO authors VALUES (2, 'Igor ', '123 ',
'igor@mail.ru
   ''http://www.softtime.ru ', '', 'Програміст', '', '', '',
407, 0);
   mysql> INSERT INTO authors VALUES (3, 'Sergey ', '212 ',
'sergey@mail.ru
   ''http://www.softtime.ru ', '', 'Дизайнер', '', '', '', 408,
0);
```

DELETE

DELETE FROM table_name [WHERE definition].

Команда DELETE видаляє з таблиці *table_name* записи, що задовольняють заданим у *definition* умовам, і повертає число видалених записів.

От як можна видалити всі записи з таблиці **authors** :

```
mysql> DELETE FROM authors;
```

Важливою частиною запитів DELETE, UPDATE і SELECT є оператор WHERE, який дозволяє задати умови для вибору записів, на які діятимуть ці команди.

Наступний запит видаляє з таблиці відвідувача, первинний ключ для якого дорівнює **1**:

```
mysql> DELETE FROM authors WHERE id_author = 1;
```

Умови відбору можуть бути значно складніші, так у прикладі віддаляються усі автори з паролем '123' і первинний ключ яких перевищує **10**:

```
mysql> DELETE FROM authors WHERE passw = '123' AND id_author
> 10;
```

Оператор AND являється логічним „и". У запитах можна так само застосовувати логічне або „або".

SELECT

Команда *SELECT* призначена для витягання рядків даних з однієї або кількох таблиць і має в загальному випадку наступний такий:

SELECT column,..
[FROM table WHERE definition]
[ORDER BY col_name [ASC | DESC], ..]
[LIMIT [offset], rows]

Тут *column* – ім'я обраного стовпця. Можна вказати кілька стовпців через кому. Якщо необхідно обрати всі стовпці можна просто вказати знак зірочки *****. Ключове слово FROM вказує таблицю *table*, з якої витягаються записи. Ключове слово WHERE визначає, так само як і в операторові DELETE, визначає умови відбору рядків. Ключове слово ORDER BY сортує рядки запитів за стовпцем *col_name* в прямому (ASC) або зворотному порядку (DESC). Ключове слово LIMIT повідомляє MySQL про вивід тільки rows запити починаючи з позиції *offset*.

Давайте вставимо в таблицю *forums* кілька записів, щоб потім на їх прикладі виконувати різні варіанти команди SELECT.

```
mysql> INSERT INTO forums VALUES (1, 'Форум 1 ', '', '', 1, 0);
mysql> INSERT INTO forums VALUES (2, 'Форум 2 ', '', '', 2, 0);
mysql> INSERT INTO forums VALUES (3, 'Форум 3 ', '', '', 3, 0);
mysql> INSERT INTO forums VALUES (4, 'Форум 4 ', '', '', 4, 0);
mysql> INSERT INTO forums VALUES (5, 'Форум 5 ', '', '', 5, 0);
```

Для того, щоб проглянути усю таблицю *forums*, виконується такий запит:

```
mysql> SELECT * FROM forums;
```

Обираємо всі стовпці з таблиці *forums* без обмежень. Результат показано на рис 8.11:

```
+----------+--------+------+------+------+------+
| id_forum | name   | rule | logo | pos  | hide |
+----------+--------+------+------+------+------+
|        1 | Форум1 |      |      |    1 |    0 |
|        2 | Форум2 |      |      |    2 |    0 |
|        3 | Форум3 |      |      |    5 |    0 |
|        4 | Форум4 |      |      |    3 |    0 |
|        5 | Форум5 |      |      |    4 |    0 |
+----------+--------+------+------+------+------+
```

Рис 8.12. Результат після вибірки без обмежень

Можна обрати не всі стовпці таблиці, а лише частину, для цього необхідно явно задати список обраних стовпців :

```
mysql> SELECT id_forum, name FROM forums;
```

у цьому випадку MySQL виведе лише два стовпці з первинним ключем *id_forum* і назвою форуму *name*

```
+----------+--------+
| id_forum | name   |
+----------+--------+
|        1 | Форум1 |
|        2 | Форум2 |
|        3 | Форум3 |
|        4 | Форум4 |
|        5 | Форум5 |
+----------+--------+
```

Рис 8.13. Результат після заданної вибірки

Оператор **LIMIT** використовується для обмеження кількості рядків, повернених командою SELECT.

Приклад:

```
mysql> SELECT * FROM forums LIMIT 3;
```

У результаті цього запиту буде виведено перші 3 записи з 5

```
+----------+--------+------+------+------+------+
| id_forum | name   | rule | logo | pos  | hide |
+----------+--------+------+------+------+------+
|        1 | Форум1 |      |      |    1 |    0 |
|        2 | Форум2 |      |      |    2 |    0 |
|        3 | Форум3 |      |      |    5 |    0 |
+----------+--------+------+------+------+------+
```

Рис 8.14. Результат запиту

Оператор **LIMIT** може також набувати двох числових аргументів, які мають бути цілими числами. У цьому випадку останній аргумент задає максимальну кількість повертаних рядків, а пе-

рший повідомляє MySQL починаючи з якої за рахунком рядка проводити відлік

```
mysql> SELECT * FROM forums LIMIT 1,3;
```

В цьому випадку будуть повернені рядки 2, 3 і 4

```
+----------+--------+------+------+------+------+
| id_forum | name   | rule | logo | pos  | hide |
+----------+--------+------+------+------+------+
|        2 | Форум2 |      |      |    2 |    0 |
|        3 | Форум3 |      |      |    5 |    0 |
|        4 | Форум4 |      |      |    3 |    0 |
+----------+--------+------+------+------+------+
```

Рис 8.15. Результат після перевірки рядків

Оператор **WHERE** застосовується в команді **SELECT** так само, як і в команді DELETE.

Оберемо з таблиці тільки ті записи, у яких значення *id_forum* більше 2:

```
mysql> SELECT * FROM forums WHERE id_forum > 2;
```

Результат показано на такому рисунку:

```
+----------+--------+------+------+------+------+
| id_forum | name   | rule | logo | pos  | hide |
+----------+--------+------+------+------+------+
|        3 | Форум3 |      |      |    5 |    0 |
|        4 | Форум4 |      |      |    3 |    0 |
|        5 | Форум5 |      |      |    4 |    0 |
+----------+--------+------+------+------+------+
```

Рис 8.16. Результат після заданної вибірки

Порядок сортування записів, що виводяться, можна задавати за допомогою оператора ORDER BY:

```
mysql> SELECT * FROM forums WHERE id_forum > 2 ORDER BY pos;
```

У цьому запиті виводяться всі записи зі значенням поля id_forum не менше двох, які при цьому сортуються за значенням поля pos. Результат такого запиту показано на рисунку:

```
+----------+--------+------+------+------+------+
| id_forum | name   | rule | logo | pos  | hide |
+----------+--------+------+------+------+------+
|        4 | Форум4 |      |      |    3 |    0 |
|        5 | Форум5 |      |      |    4 |    0 |
|        3 | Форум3 |      |      |    5 |    0 |
+----------+--------+------+------+------+------+
```

Рис 8.17. Результат запиту

UPDATE

UPDATE table

SET col_name1=expr1 [, col_name2=expr2 ..]

[WHERE definition]
[LIMIT rows]

Команда UPDATE оновлює стовпці таблиці *table* відповідно до їх нових значень у рядках наявної таблиці. У вираженні SET зазначається, які саме стовпці слід модифікувати і які величини мають бути в них установлені. У виражезі WHERE, якщо він наявний, задається, які рядки повинні оновлюватися. В інших випадках оновлюються всі рядки. Ключове слово LIMIT дозволяє обмежити число оновлюваних рядків.

У прикладі розділу форуму з первинним ключем 2 встановлюється нова назва (PHP) і встановлюється атрибут *hide*, що дорівнює 1, роблячи форум невидимим.

UPDATE forums SET name='PHP ', hide=1 WHERE id_forum=2;

SHOW

З цією командою ми вже зустрічалися раніше, коли виконували запити виду *show databases* і *show tables* для отримання списку баз даних і таблиць в обраній базі даних.

Розглянемо ще кілька варіантів використання цієї команди.

Вивести список усіх стовпців обраної таблиці можна за допомогою такого запиту:

mysql> SHOW FIELDS FROM authors;

Результат наведено на рисунку:

Field	Type	Null	Key	Default	Extra
id_author	int(6)		PRI	NULL	auto_increment
name	text	YES		NULL	
passw	text	YES		NULL	
email	text	YES		NULL	
url	text	YES		NULL	
icq	text	YES		NULL	
about	text	YES		NULL	
photo	text	YES		NULL	
time	datetime	YES		NULL	
last_time	datetime	YES		NULL	
themes	int(10)	YES		NULL	
statususer	int(2)	YES		NULL	

Рис 8.18. Результат запиту show

Можна також відобразити інформацію про всі індекси конкретної таблиці:

mysql> SHOW INDEX FROM authors;

Виконавши команду SHOW PROCESSLIST можна побачити список усіх запитів, що виконуються в системі:

```
+----+------+----------------+-------+---------+------+-------+-----------------
-+
| Id | User | Host           | db    | Command | Time | State | Info
 |
+----+------+----------------+-------+---------+------+-------+-----------------
-+
|  1 | ODBC | 127.0.0.1:3008 | forum | Query   | 0    | NULL  | SHOW PROCESSLIST
 |
+----+------+----------------+-------+---------+------+-------+-----------------
-+
```

Контрольні питання:

1. Сформувати поняття MySQL, назвати особливості застосування.
2. Назвати різновиди баз даних.
3. Описати модель реляційної бази даних у Web–додатках.
4. Навести приклад таблиці forums бази даних великого форуму.
5. Описати особливості реляційної бази.
6. Сформувати поняття індексу, унікального індексу, первинного та зовнішнього ключа.
7. Перелічити засоби завдання первинного ключа.
8. Навести приклади нормалізації баз даних. Для чого існує денормолізація баз даних?
9. Назвати особливості роботи з MySQL.
10. Перерахувати команди запитів SQL.
11. За допомогою якого запиту можна вивести список усіх стовпців вибраної таблиці?

Лекція 9. Взаємодія MySQL с PHP

План:

1. Підтримка баз даних у PHP.
2. Стандартні функції PHP для роботи з MySQL.
3. Підключення до серверу MySQL.
4. Інструментарій phpMyAdmin.
5. Зміна табличних даних за допомогою phpMyAdmin.

Підтримка баз даних у PHP

Один з найважливіших аспектів PHP – підтримку баз даних. У PHP реалізована велика підтримка практично всіх наявних серверів баз даних, зокрема:

Adabas D	Informix	PostgreSQL
Dbase	Ingres	Solid
Direct MS - SQL	InterBase	Sybase
Empress	mSQL	UNIX dbm
File - Pro (read - only)	MySQL	Velods
FrontBase	ODBC	
IBM DB2	Oracle (OCI7 і OC18)	

Підтримка бази даних у PHP представлена набором стандартних функцій для з'єднання з базою, обробки запитів і розриву зв'язку.

Стандартні функції PHP для роботи з MySQL

Загальна послідовність дій при взаємодії з сервером MySQL виглядає так:

– установити з'єднання з сервером MySQL. Якщо спроба завершується невдачею, вивести відповідне повідомлення і завершити процес.

– обрати базу даних сервера MySQL. Якщо спроба вибору завершується невдачею, вивести відповідне повідомлення і завер-

шити процес. Допускається одночасне відкриття кількох баз даних для обробки запитів.

Обробити запити до обраної бази (чи базам).

Після завершення обробки запитів закрити з'єднання з сервером баз даних.

У прикладах цієї лекції використовуються база даних і її таблиці з попередньої лекції.

Підключення до сервера у MySQL

mysql_connect()

Функція *mysql_connect( )* встановлює зв'язок з сервером MySQL Після успішного підключення до MySQL можна переходити до вибору баз даних, що обслуговуються цим сервером.

Синтаксис функції *mysql_connect( )*:

int mysql_connect ([string хост [:порт] [:/шлях//до/сокета] [, string ім'я користувача] [, string пароль]).

У параметрі хост передається ім'я хостового компютера, вказане в таблицях привілеїв серверу MySQL. Звичайно, воно ж використовується для перенаправлення запитів на web – сервер, на якому працює MySQL, оскільки до сервера MySQL можна підключатися у видаленому режимі. Разом з ім'ям хоста можуть вказуватися необов'язкові параметри – номер порту, а також шлях до сокета (для локального хоста). Параметри *ім'я_користувача* й пароль повинні відповідати імені користувача і паролю, заданим у таблицях привілеїв MySQL. Зверніть увагу: усі параметри є необов'язковими, оскільки таблиці привілеїв можна настроїти так, щоб вони допускали з'єднання без перевірки. Якщо параметр хост не заданий, *mysql_connect( )* намагається встановити зв'язок з локальним хостом.

Приклад відкриття з'єднання з MySQL*:*

```
<?php
@mysql_connect("localhost", "root", "12345") or
die("Could not connect to MySQL server"!);
?>
```

У цьому прикладі *localhost* – ім'я комп'ютера, *root* – ім.'я користувача, а 12345 - пароль доступу до бази даних MySQL.

Знак @ перед викликом функції *mysql_connect( )* пригнічує усі повідомлення про помилки, видавані при невдалій спробі підключення вони замінюються повідомленням, вказаним при виклику *die( )*. Зверніть увагу: значення, повертань при виклику mysql_connect( ), в цьому прикладі не використовується.

Якщо в програмі використовується всього одне з'єднання з сервером MySQL, це цілком нормально. Але якщо програма встановлює з'єднання з кількома серверами MySQL на різних хостах, слід зберегти ідентифікатор з'єднання, повертаний при виклику *mysql_connect( )*, щоб адресувати такі команди потрібному серверу MySQL.

Приклад:

```
<?php
$link1 = @mysql_connect("www.somehost.com", "web",
"abcde") or die("Could not connect to MySQL server"!);
$link2 = @mysql_connect("www.someotherhost.com",
"usr", "secret") or die("Could not connect to MySQL
server"!);
?>
```

Ідентифікатори $link1 і $link2 передаються при подальших зверненнях до баз даних із запитами.

Функція *mysql_pconnect( )* забезпечує підтримку відновлюваних *(persistent)* з'єднань. У розрахованих на багатьох користувачів середовищах рекомендується використовувати *mysql_pconnect( )* замість *mysql_connect( )* для економії системних ресурсів. За типами параметрів і повертаного значення функція *mysql_pconnect( )* збігається з *mysql_connect( )*.

mysql_select_db( )

Після успішного з'єднання з MySQL необхідно обрати базу даних, що знаходиться на сервері. Для цього використовується функція *mysql_select_db*(). *Синтаксис* функції *mysql_select_db( )*:

int mysql_select_db (string ім'я_бази_даних [, int ідентифікатор_з'єднання]).

Параметр **ім.'я_бази_даних** визначає обрану базу даних, ідентифікатор якої повертається функцією *mysql_select_db( )*. Зверніть увагу: параметр **ідентифікатор_з'єднання** необов'язковий лише при одному відкритому з'єднанні з сервером MySQL. За

наявності кількох відкритих з'єднань цей параметр повинен зазначатися.

Приклад вибору бази даних функцією *mysql_select_db( ) :*

```
<?php
@mysql_connect("localhost", "root", "12345") or
die("Could not connect to MySQL server"!);
@mysql_select_db("forum") or die("Could not select
forum database"!);
?>
```

Якщо в програмі обирається тільки одна база даних, зберігати її ідентифікатор не обов'язково. Проте при виборі кількох баз даних повертані ідентифікатори зберігаються, щоб ви могли послатися на потрібну базу при обробці запиту. Якщо ідентифікатор не вказаний, використовується остання обрана база даних.

mysql_close( )

Після завершення роботи з сервером MySQL з'єднання необхідно закрити. Функція *mysql_close( )* закриває з'єднання, визначуване необов'язковим параметром. Якщо параметр не заданий, функція *mysql_close( )* закриває останнє відкрите з'єднання.

Синтаксис функції *mysql_close( )*:

int mysql_close ([int ідентифікатор_з'єднання]).

Приклад.

```
<?php
@mysql_connect("localhost", "root", "12345") or
die("Could not connect to MySQL server"!);
@mysql_select_db("forum") or die ("Could not select
company database"!);
print "You' re connected to a MySQL database"!;
mysql_close();
?>
```

В цьому прикладі зазначити ідентифікатор з'єднання не треба, оскільки на момент виклику *mysql_close( )* існує лише одне відкрите з'єднання з сервером.

З'єднання, відкриті функцією *mysql_pconnect( )*, закривати не обов'язково.

mysql_query( )

Функція *mysql_query( )* забезпечує інтерфейс для поводження із запитами до баз даних.

Синтаксис функції *mysql_query( )*:

int mysql_query (string запит [, int ідентифікатор_з'єднання]).

Параметр запит утримує текст запиту на мові SQL. Запит передається або з'єднанню, визначуваному необов'язковим параметром **ідентифікатор_з'єднання,** або, за відсутності параметра, останньому відкритому з'єднанню.

Часто помилково думають, що функція **mysql_query( )** повертає результати обробки запиту. Це не так – залежно від типу запиту виклик *mysql_query( )* може приводити до різних наслідків. При успішному виконанні команди SQL SELECT повертається ідентифікатор результату, який згодом передається функції *mysql_result( )* для наступного форматування і відображення результатів запиту. Якщо обробка запиту завершилася невдачею, функція повертає FALSE.

Якщо при обробці запитів витрачається надто багато пам'яті, використовується стандартна функція PHP **mysql_free_result**. При виклику їй передається ідентифікатор результату, повертаний **mysql_query( ).** Функція *mysql_free_result( )* звільняє усю пам'ять, пов'язану з цим запитом.

mysql_affected_rows ( )

У багатьох ситуаціях вимагається дізнатися кількість записів, що беруть участь в запиті SQL з командами INSERT, UPDATE, REPLACE або DELETE. Завдання вирішується функцією *mysql_affected_rows( ).*

Синтаксис функції:

int mysql_affected_rows ([int ідентифікатор_з'єднання])

Параметр **ідентифікатор_з'єднання** не є обов'язковим. Якщо він не вказується, *mysql_affected_rqws( )* намагається використовувати останнє відкрите з'єднання.

Приклад:

```
<?php
```

```
// Підключитися до сервера і вибрати базу даних
@mysql_connect("localhost", "root", "12345") or
die("Could not connect to MySQL server"!);
@mysql_select_db("forum") or die("Could not select
forum database"!);
// Створити запит
$query = "UPDATE authors SET name = \"игорь\" WHERE
prod_id = \"3\"";
// Виконати запит
$result = mysql_query($query);
// Визначити кількість оновлених записів;
print "Total row updated". mysql_affected_rows();
mysql_close();
?>
```

При виконанні цього фрагмента буде виведений наступний результат:

```
Total row updated: 1
```

При виконанні цього фрагмента буде виведено такий результат:

```
Total row updated: 1
```

Функція **mysql_affected_rows( )** не працює із запитами, заснованими на команді SELECT. Для визначення кількості записів, повернених при виклику SELECT, використовується функція *mysql_num_rows( )*, описана нижче.

В одній специфічній ситуації функція *mysql_affected_rows( )* працює з помилкою. При виконанні команди DELETE без секції WHERE *mysql_affected_rows( )* завжди повертає 0.

mysql_num_rows( )

Функція *mysql_num_rows( )* визначає кількість записів, повертаних командою SELECT.

Синтаксис функції *mysql_num_rows( )*:

int mysql_num_rows(int результат).

Приклад використання *mysql_num_rows( ) :*

```
<?php
// Підключитися до сервера і вибрати базу даних
@mysql_connect("localhost", "root", "12345") or
die("Could not connect to MySQL server"!);
@mysql_select_db("forum") or die("Could not select
forum database"!);
```

```
    // Вибрати усі прізвища, які починаються з 'm'
    $query = "SELECT name FROM authors WHERE name LIKE
\"m*\"";
    // Виконати запит
    $result = mysql_query($query);
    print       "Total       rows       selected:       "
.mysql_num_rows($result);
    mysql_close();
      ?>
```

Оскільки таблиця містить лише одну людину, ім.'я якої починається з букви m (max), повертається тільки один запис. Результат:

```
Total rows selected: 1
```

mysql_result( )

Функція *mysql_result()* використовується у поєднанні з *mysql_query( )*

((при виконанні запиту з командою SELECT) для отримання набору даних.

Синтаксис функції *mysql_result()*:

int mysql_result (int ідентифікатор_результату, int запис [. mixed поле"]')

У параметрі **ідентифікатор_результату** передається значення, повернене функцією *mysql_query( )*. Параметр запис посилається на певний запис набору даних, визначуваного параметром **ідентифікатор_результату**. Нарешті, в необов'язковому параметрі поле можуть передаватися:

– зміщення поля в таблиці;

– ім'я поля;

– ім'я поля у форматі *ім'я_поля*_ім'я_таблиці.

Приклад: Вибірка і форматування даних у базі даних MySQL

```
    <?php
    // Підключитися до сервера і вибрати базу даних
    @mysql_connect("localhost",   "root",   "12345")   or
die("Could not connect to MySQL server"!);
    @mysql_select_db("forum") or die("Could not select
forum database"!);
    // Вибрати усі записи з таблиці authors
    $query = "SELECT * FROM authors";
    $result = mysql_query($query);
    $x = 0;
```

```
    print "<table>\n";
    print
"<tr>\n<th>id_author</th><th>Name</th><th>Password</th>\
n</tr>\n";
    while ($x < mysql_numrows($result)) :
    $id = mysql_result($result, $x, 'id_author');
    $name = mysql_result($result, $x, 'name');
    $price = mysql_result($result, $x, 'passw');
    print "<tr>\n";
    print
"<td>$id</td>\n<td>$name</td>\n<td>$price</td>\n";
    print "</tr>\n";
    $x++;
    endwhile;
    print "</table>";
    mysql_close();
    ?>
```

Функція *mysql_result( )* зручна для роботи з порівняно невеликими наборами даних, проте існують і інші функції, що працюють набагато ефективніше, – а саме, функції *mysql_fetch_row( )* і *mysql_fetch_array( ).*

mysql_fetch_row()

Зазвичай набагато зручніше відразу присвоїти значення всіх полів запису елементам індексованого масиву (починаючи з індексу 0), ніж багаторазово викликати *mysql_result( )* для отримання окремих полів. Завдання вирішується функцією *mysql_fetch_row( )*, що має такий *синтаксис*:

array mysql_fetch_row (int результат).

Використання функції *list( )* у поєднанні з *mysql_fetch_row( )* дозволяє заощадити кілька команд, необхідних при використанні *mysql_result( ).*

Приклад. Вибірка даних функцією *mysql_fetch_row( ).*

```
    <?php
    // Підключитися до сервера і вибрати базу даних
    @mysql_connect("localhost",   "root",   "12345")   or
die("Could not connect to MySQL server"!);
    @mysql_select_db("forum")  or  die("Could  not  select
forum database"!);
    $query = "SELECT * FROM authors";
    $result = mysql_query($query);
    print "<table>\n";
```

```
    print
"<tr>\n<th>id_author</th><th>Name</th><th>Password</th>\
n</tr>\n";
    while ($row = mysql_fetch_array($result)) :
    print "<tr>\n";
    print
"<td>".$row["id_author"].</td>\n<td>".$row["name"]".
    </td>\n<td>" .$row["passw"]. "</td>\n";
    print "</tr>\n";
    endwhile;
    print "</table>";
    mysql_close();
      ?>
```

mysql_fetch_array()

Функція *mysql_fetch_array( )* аналогічна до *mysql_fetch_row( ),* проте за умовчанням значення полів запису зберігаються в асоціативному масиві. Ви можете обрати тип індексації (асоціативний, числовий або комбінований).

Синтаксис функції *mysql_fetch_array( )*:

array mysql_fetch_array (int ідентифікатор результату [, тип_індексації]).

У параметрі ідінтифікатор_результату передається значення, повернуте функцією *mysql_query#00.* Необов'язковий параметр **тип_індексації** набуває одне з таких значень:

MYSQL_ASSOC – функція ***mysql_fetch_array( )*** повертає асоціативний масив. Якщо параметр не зазначений, це значення використовується за умовчанням;

MYSQL_NUM – функція ***mysql_fetch_array( )*** повертає масив з числовою індексацією;

MYSQL_BOTH – до полів повертаного запису можна звертатися як по числових, так і за асоціативними індексами.

Приклад. Вибірка даних функцією *mysql_fetch_array( ).*

```
    <?php
    @mysql_connect("localhost",   "root",   "12345")   or
die("Could not connect to MySQL server"!);
    @mysql_select_db("forum")  or  die("Could  not  select
forum database"!);
    $query = "SELECT * FROM authors";
    $result = mysql_query($query);
    print"<table>\n";
    print     "<tr>\n<th>id_author</th>     <th>Name</th>
<th>Password</th>\n </tr>\n";
```

```
    while ($row = mysql_fetch_array($result)) :
    print "<tr>\n";
    print  "<td>".  $row["id_author"].  "</td>\n  <td>"  .
$row["name"]   .   "</td>\n   <td>"   .   $row["passw"]   .
"</td>\n" ;
    print "</tr>\n";
    endwhile;
    print "</table>";
    mysql_close();
    ?>
```

Приклад уведення і виведення інформації з таблиці.

```
   // Підключитися до сервера і вибрати базу даних
   @mysql_connect("localhost",   "root",   "12345")   or
die("Could not connect to MySQL server"!);
   @mysql_select_db("student") or die("Could not select
forum database"!);
   $query="INSERT INTO stud VALUES (1, 'Іванов', '21')";
   $result = mysql_query($query);
   $query="INSERT INTO stud VALUES (2, 'Петров', '19')";
   $result = mysql_query($query);
   $query="INSERT INTO stud VALUES (3, 'Сидоров', '20')";
   $result = mysql_query($query);
   $query="INSERT   INTO   stud   VALUES   (4,   'Миколаїв',
'23')";
   $result = mysql_query($query);
   // Вибрати усі записи з таблиці authors
   $query = "SELECT * FROM stud ORDER BY surname";
   $result = mysql_query($query);
   $x = 0;
   print "<table>\n";
   print
"<tr>\n<th>id</th><th>surname</th><th>age</th>\n</tr>\n"
;
   while ($x < mysql_numrows($result)) :
   $id = mysql_result($result, $x, 'id');
   $name = mysql_result($result, $x, 'surname');
   $price = mysql_result($result, $x, 'age');
   print "<tr>\n";
   print<td>$id</td>\n<td>$name</td>\n<td>$price</td>\n";
   print "</tr>\n";
   $x++;
   endwhile;
   print "</table>";
   mysql_close();
    ?>
```

Інструментарій phpMyAdmin

Для виконання вправ, наведених нижче, буде потрібно доступ до MySQL–серверу. Як інтерфейс для MySQL використовується phpMyAdmin–PHP застосування, запущене на Web–сервері. Для повного ознайомлення з можливостями phpMyAdmin рекомендується прочитати книгу "Mastering phpMyAdmin for effective MySQL Management".

Створення таблиць у phpMyAdmin

Як приклад, ми використовуватимемо географічну інформаційну систему. Допустимо, ми вирішили, що нам потрібна інформація про міста та країни – таким чином, нам знадобляться дві таблиці, які будуть частиною бази даних 'geodb'. Для створення таблиць можна використовувати вкладку "Structure" на сторінці перегляду бази даних, або використовувати блок SQL – запитів для введення відповідного виразу :

Щоб створити таблицю, скористаємося виразом CREATE TABLE, в якому ми задамо ім’я нашої нової таблиці. Вирз починається з CREATE TABLE, після якої йде ім’я таблиці. Потім у дужках зазначається список стовпців та інформація про ключі. Кожному стовпцю дається ім'я, вказується тип даних, вказується атрибут NULL або NOT NULL (тут, NOT NULL означає, що колонка не може мати значення NULL), і значення за умовчанням, якщо воно доречне.

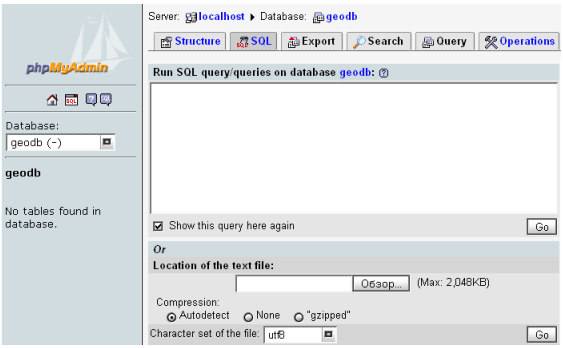


Рис 9.1. Інтерфейс phpMyAdmin

```
CREATE TABLE cities ( id int(11) NOT NULL auto_increment, city_name varchar(50) NOT NULL default '', latitude varchar(15) NOT NULL default '', longitude varchar(15) NOT NULL default '', population int(11) NOT NULL default '0 ', country_code char(2) NOT NULL default '', PRIMARY KEY (id)) TYPE=MyISAM AUTO_INCREMENT=1
```

Стовпець id – це первинний ключ *(primary key)*, колонка, яка унікально ідентифікує кожне місто. Тип даних цього стовпця – INT (ціле число нормального розміру), MySQL призначає унікальні значення для цього стовпця, завдяки атрибуту auto_increment. Зверніть увагу на те, що ми не можемо використовувати назву міст як первинний ключ, оскільки деякі імена міст не унікальні у світі. Для відображення чисельності населення ми також використовуємо ціле число.

Інші стовпці використовують як типи даних рядка фіксованої довжини (CHAR) або рядка змінної довжини (VARCHAR). Коли ми точно знаємо довжину рядка, краще використовувати CHAR, задаючи довжину стовпця як CHAR(2). Інакше, ми використовуємо як тип даних рядка змінної довжини, зазначаючи тільки максимальну довжину рядка, наприклад: VARCHAR(15).

Після списку стовпців ми можемо вказати додаткові параметри таблиці, наприклад, її тип, перше значення для стовпця автоінкремента. SQL – вираз закінчується крапкою з комою.

Створивши таблицю для міст, ми робимо ту ж операцію, але цього разу для таблиці країн.

```
CREATE TABLE countries ( country_code char(2) NOT NULL default '', country_name varchar(100) NOT NULL default '' ) TYPE=MyISAM;
```

Зауважемо, що стовпець *'country_code'* наявний в обох таблицях. Це відбиває принцип зв'язку : *country_code в 'cities'* пов'язаний з однойменним стовпцем у таблиці *'countries'*. Таким чином, ми економимо на місці, вказуючи назву країни в базі даних тільки одного дня.

В іншому розділі (Migrating to InnoDB) техніка зв'язування розглядається детальніше. Після того, як таблиці створено, слід увести в них які-небудь дані.

Зміна табличних даних за допомогою phpMyAdmin

ивчимо базовий синтаксис виразів INSERT, UPDATE, DELETE, і SELECT.

Додавання даних за допомогою INSERT

Спершу вивчимо вираз INSERT на прикладі коду, який генерує phpMyAdmin при виконанні операції INSERT. Для цього відкриваємо вкладку Insert на сторінці перегляду таблиці *'countries '*, і вводимо дані про країну:

Після того, як ми клацаємо на Go, дані записуються в таблицю і phpMyAdmin показує нам використаний вираз INSERT:

INSERT INTO 'countries' ('country_code', 'country_name') VALUES ('ca ', 'Canada');

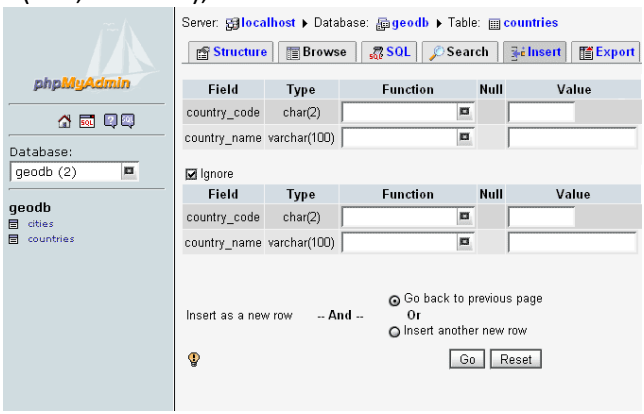


Рис 9.2. Діалогове вікно INSERT у phpMyAdmin

Після частини INSERT INTO, йде ім'я таблиці. У MySQL, ми можемо уміщувати іимена таблиць та імена столбців у зворотні галочки """, якщу в іменах використовуються спецсимволи, зарезервирвовані слова. Потім ми відкриваємо першу дужку, перераховуємо стовпці до яких буде здійснене вставка, розділяючи їх один від одного комами. Після переліку списку назв стовпців дужка закривається і зазначається зарезервоване слово VALUES, після якого в дужках перераховуються значення, які необхідно вставляти в таблицю, причому перераховуються у

тому ж порядку назви стовпців. Якщо значення мають символьний тип даних, необхідно брати їх у дужки.

Давайте занесемо до таблиці 'cities' ці міста:

```
INSERT INTO 'cities' ( 'id', 'city_name', 'latitude', 'longitude', 'population ', 'country_code' ) VALUES (''Sherbrooke ', '45 23 59.00 ', '- 71 46 11.00 ', 125000, 'ca');
```

Тут задаємо порожнє значення для id, тому що атрибут автоінкремента цього стовпця забезпечує автоматичне виставлення унікального значення. Також слід звернути увагу, що значення 'population' – числове, тому не в лапках.

Розглянемо окремі дані для кількох інших країн і міст, які знадобляться нам пізніше.

```
INSERT INTO 'countries' ( 'country_code', 'country_name' ) VALUES ('zh ', 'China'); INSERT INTO 'cities' ( 'id', 'city_name', 'latitude', 'longitude', 'population', 'country_code' ) VALUES (''Shanghai ', '31 13 58.00 ', '121 26 59.99 ', 11000000, 'zh');
```

Оновлення даних за допомогою UPDATE

Спочатку клацніть на 'Browse' для таблиці 'cities', у результаті буде виведено поки єдиний запис

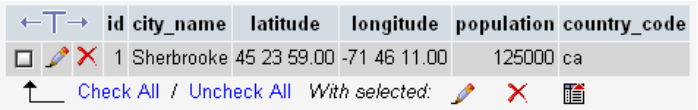


Рис 9.3. Результат запису 'Browse' у phpMyAdmin

Натискуючи на іконці у вигляді олівця на папері (чи посиланню Edit), ми переходитимемо на панель редагування цього рядка. Змінимо значення стовпця 'population' на 130000. Після клацання на 'Save ', phpMyAdmin відображує такий вираз:

```
UPDATE 'cities' SET 'population' = '130000' WHERE 'id' = '1' LIMIT 1 ;
```

Ключове слово в цьому виражезі – 'UPDATE ', за яким йде назва таблиці. Слово 'SET' передує списку модифікацій (у нашому випадку – тільки для стовпця 'population') який записується у форматі "стовпець = нове значення".

Ми бачимо, що у виразі присутня умова: WHERE 'id' = '1 ', у якому використовується первинний ключ, щоб обме-

жити зміну значення стовпця 'population' тільки цим рядком, тобто тільки для цього міста.

Частина limit 1 додається phpMyAdmin і є гарантією, що якщо первинний ключ не заданий, зміни не будуть застосовані більш ніж до одного запису.

За один запит 'UPDATE' можуть бути змінені значення відразу кількох стовпців :

```
UPDATE 'cities' SET 'city_name' = 'Sherbrooke, Quebec ', 'population' = '130001' WHERE 'id' = '1' LIMIT 1 ;.
```

Видалення даних за допомогою DELETE

У режимі 'Browse' (перегляд) таблиці 'cities ', натисніть на червоній іконці кошики (чи посиланню Delete) - буде згенерований запит, що запрошує підтвердження виконання такого виразу:

```
DELETE FROM 'cities' WHERE 'id' = '1' LIMIT 1 ;
```

Синтаксис тут дуже простий, і містить тільки назву таблиці, і умова при якому буде виконана операція видалення.

Виключення умови WHERE із запитів UPDATE або DELETE цілком допустиме в SQL, але в такому разі дія вираження буде застосована до кожного запису таблиці!

Вибірка даних за допомогою SELECT

Витягання інформації з таблиць – імовірно найбільш часто використовуваний вид запиту. Наприклад, запити SELECT дозволяють отримати відповіді на подібні питання: "які міста мають чисельність населення більшу, ніж це число"?.

Фактично, ми вже заздалегідь використовували SELECT, коли клікали на посилання Browse для таблиці 'cities'. Це згенерувало просту форму вираження запиту SELECT :

```
SELECT * FROM 'cities' LIMIT 0,30;
```

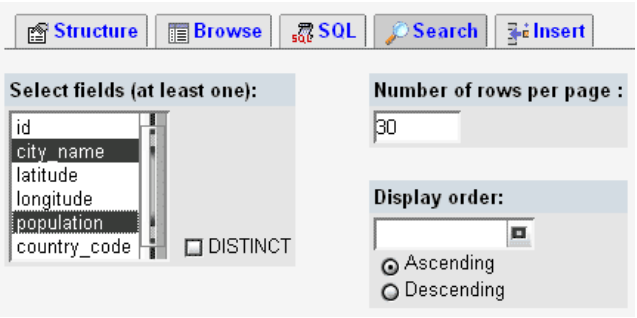


Рис 9.4. Результат запиту SELECT

Зірочка тут означає "усі стовпці". Ми додали FROM і ім'я таблиці, в якій буде виконаний запит на вибірку. LIMIT 0,30 означає що вибірка починається за записом номеру 0 (найпершою), і містить максимум 30 записів.

Вкладка Search дозволяє побачити більшу кількість опцій для запиту SELECT. Виберемо вкладку Search для таблиці cities, і виберемо тільки ті стовпці, які нам потрібні, :

Потім праворуч від списку стовпців ми виберемо порядок сортування отриманої вибірки по стовпцю 'population' по убуванню:

city_name	population	country_name
Shanghai	11000000	China
Sherbrooke, Quebec	130001	Canada

Рис 9.5. Результат сортування вибірки по стовпцю 'population'

В результаті phpMyAdmin згенерує наступний запит:

```
SELECT 'city_name', 'population' FROM 'cities' WHERE 1ORDER BY 'population' DESC LIMIT 0,30;
```

Ми бачимо, що зірочка була замінена списком стовпців, розділених комами. Умова WHERE 1, додане phpMyAdmin -му, завжди істинно і вибирає усі записи. Трохи пізніше ми побачимо, що можна замінити його іншою умовою. Крім того, з'являється умова ORDER BY, після якої слідує назва стовпця по якому ми хочемо сортувати результат вибірки, і ключове слово DESC для сортування по убуванню (ми могли також використовувати ASC для сортування за збільшенням).

Умови в SQL - запитах

Найпростішим способом додати умову – клік по SQL - query: Edit, на сторінці результатів, в результаті якого буде відкрито спливаюче вікно "Query". Додамо умову для стовпця 'country ':

```
SELECT 'city_name', 'population' FROM 'cities' WHERE country_code = 'zh' ORDER BY 'population' DESC;
```

Ця умова вибере усі міста, що знаходяться в Китаї. При позначенні умов може бути використана багата безліч операторів і функцій. Ось - два ***приклади***:

Знайти канадські міста з чисельністю населення більш 100000:

```
WHERE population > 100000 AND country_code = 'ca';
```

Знайти міста, чиї назви які починаються з символу "A":

```
WHERE city_name like 'A%'.
```

Функції угрупування

Підсумкова інформація може бути згенерована в результаті угрупування по певному стовпцю. Давайте упізнаємо середню чисельність міського населення в країні:

```
SELECT country_code, AVG(population) FROM cities GROUP BY country_code
```

Інші можливі функції угрупування - MIN(), MAX(), SUM() і COUNT(), які обчислюють відповідно мінімальне значення, максимальне значення, суму значень, і число записів. Наприклад, за допомогою такого запиту ми можемо отримати число міст у країні:

```
SELECT country_code, count(city_name) FROM citiesGROUP BY country_code;
```

Об’єднання

Зазвичай, реляційна база даних містити безліч таблиць, пов’язаних загальними ключами. Часто виникає необхідність у запитах відразу для кількох таблиць. Зв’язати, або об’єднати

таблиці можна за допомогою різних методів: ми зосередимося на найпростішому методі, що полягає в порівнянні ключів.

У нижче розглянутому запиті умова FROM містить список назв таблиць, розділених комами. У списку стовпців ми використовуємо назви таблиць і точку як префікс перед кожною назвою стовпця (у цьому немає необхідності в разі, якщо всі стовпці з однієї таблиці).

```
SELECT cities.city_name, cities.population,
countries.country_name FROM cities, countries WHERE
cities.country_code = countries.country_code LIMIT
0,30.
```

Контрольні питання:

1. Як у PHP представлена підтримка бази даних для її з'єднання, обробки запитів і розриву зв'язку?
2. Як виглядає загальна послідовність дій при взаємодії з сервером MySQL?
3. Як підключити до серверу MySQL?
4. Що трапляється при виконанні команди DELETE без секції WHERE *mysql_affected_rows( )*?
5. Навести приклад введення і виведення інформації з таблиці.
6. Назвати основні можливості phpMyAdmin.
7. Перелічити особливості створення таблиць у phpMyAdmin.
8. За допомогою яких виразів phpMyAdmin здійснюэться зміна табличних даних?
9. Як задати умови у SQL–запитах?
10. Назвати особливості функцій угрупування та об'єднання.

Питання та завдання до модульних робіт

Модульна робота 1.

1.Обчислити: $y=\begin{cases}\sin x^{2}, & при \quad x=5;\\ 18x, & при \quad x>5,\end{cases}$

2.Обчислити: $y=\begin{cases}e^{-x}, & при\, x<7;\\ tgx, & при\, x=7;\\ \sin x, & при\, x>7.\end{cases}$

3.Дано дійсне число x. Обчислити n доданків.

$$y=x-\frac{x^3}{3}+\frac{x^5}{5}-\frac{x^7}{7}+...$$

4.Обчислити з точністю до 0.001. $\frac{1}{1!}+\frac{2}{2!}+\frac{3}{3!}+\frac{4}{4!}+\frac{5}{5!}+...$

5.Дано натуральне число n(n< 9999). Переставити першу та останню цифри числа n.

6.Дано натуральне число n ($n\le 9999$). Знайти суму цифр числа n.

7. Упорядкувати список з 10 студентів.

8. Визначення наявності студента у списку з 8 студентів.

9. Передача значення змінної через форму.

10. Типи змінних (за видимістю). Зміна видимості змінних (приклади).

11. Оператори і операція розгалуження.

12.Дано натуральне число n. Обчислити:

$$y=\begin{cases}\sum_{i=1}^{n} i^{2}, & при \quad x\ge 1;\\ \emptyset, & при \quad x<1.\end{cases}$$

13. Динамічні змінні.

14. Змінні (за значенням). Перетворення типів.

15. Оператори повторення (цикли).

16. Обчислити: $y=\frac{\sqrt{a^2-4bc}}{2c}$; при відомих a, b, c.

17. Дано натуральне число n. Обчислити:

$$y = \begin{cases} \prod_{i=1}^{n}(i-2) & \text{, при } x > 0; \\ 1 & \text{, при } x \le 0. \end{cases}$$

Модульна робота 2.

1. Вважати рядок з текстового файла 1.txt, попередньо перевірити на існування цього файла, замінити поєднання 'ма' на 'ам', вивести на екран початковий рядок, записати результат у текстовий файл 2.txt і вивести результат з файла на екран.

2. У текстовому файлі stud.txt знаходяться дані у форматі № Прізвище Ім'я Курс (попередньо записані 7 студентів групи). Зчитати дані з файла в масив, попередньо перевірити на наявність файла. Упорядкувати масив з курсів у порядку спадання і вивести на екран. Створити окремі файли для студентів кожного курсу.

3. Створити форму авторизації у файлі form.php, яка містить поля Нік, Пароль, Електронна Пошта і кнопку передачі даних у файл proverka.php. Дані, уведені у форму, зберігаються в базі даних, причому в разі збігу ніка або адреси електронної пошти видається повідомлення й потрібне повторне введення унікальних даних. Вивести дані про користувачів на екран у файлі users.php.

4. Створити текстовий файл a.txt, у який записати фразу з текстового поля форми файла wwod.php. Вважати рядок з файла й замінити прогалини між словами на подвійні пробіли. Дописати новий рядок у той же файл. Зчитати обидва рядки на екран.

5. У текстовому файлі student.txt знаходяться дані у вигляді № Прізвище Ім'я Рік народження (усього 7 осіб). Вважати дані з файла в масив. Упорядкувати масив за прізвищами й записати впорядковані дані в новий файл spisok.txt у форматі № Рік народження Прізвище. Вивести обидва списки на екран.

6. Створити форму у файлі shop.php, яка містить два поля списків різних товарів (не менше 5 товарів у кожному списку), поле авторизації за ніком, поле area, у якому відображаються обрані елементи.Реалізувати можливість формування корзини покупця в текстовому полі з можливістю скасування обраного товару та записом номера замовлення, суми замовлення і ніка замовника в базі даних. Вивести дані про замовлення на екран у файлі zakaz.php.

Питання до самостійної роботи

До занять 1 - 3

1.Яку роль відіграють протоколи в мережі Інтернет?

2.Яким чином організована передача даних в Інтернет?

3.За рахунок чого в мережі Інтернет досягається надійність передачі даних?

4.Назвіть основні організації, що курирують розвиток архітектури і протоколів Інтернет.

5.Де реєструються доменні імена в Російському сегменті Інтернет?

6.Хто керує Інтернет і в якому сенсі?

7.Хто реєструє доменні імена в Інтернет?

8.Що таке RFC?

9.Які завдання вирішує консорціум W3C?

10.Що таке IP-адреса?

11.Що таке доменне ім’я?

12.Опишіть структуру доменних імен.

13.Що таке DNS? Опишіть роботу DNS -сервера.

14.Що таке проксі-сервер?

15.Які завдання вирішує проксі-сервер?

16.Назвіть основні протоколи Інтернет і їх призначення.

17.Яку принципову проблему розв'язало створення FTP?

18.Яку принципову проблему розв'язало створення Telnet?

19.Для чого використовується програма Telnet?

20.Які ви знаєте поштові протоколи в Інтернет?

21.Що таке HTTP-протокол?

22.Яка модель взаємодії закладена в основу протоколу HTTP?

23.Що таке товстий клієнт і тонкий клієнт?

24.Яку структуру має запит клієнта?

25.У якій частині запиту клієнта знаходиться URL запрошуваного ресурсу?

26.Яку роль виконують поля заголовка запиту клієнта?

27.У якій частині запиту клієнта знаходиться MIME інформація?

28.Яку структуру має відповідь сервера?

29.Яку роль виконують поля заголовка відповіді сервера?

30.Яка інформація міститься в рядку стану відповіді сервера?

31.У якій частині відповіді сервера знаходиться MIME інформація?

32.У якій частині відповіді сервера знаходиться запрошуваний клієнтом ресурс?

33.Що таке MIME?

34.Який формат має поле Content-Type? Наведіть приклади.

35.Що таке URL, URN, URI?

36.Чим https відрізняється від http?

37.Які типи аутентифікації підтримуються у Веб?

38.Що таке SSL?

39.Що таке Cookie і для чого вони призначені?

40.Чи існують які-небудь обмеження доступу до записів Cookie?

41.Де і в якому форматі передаються дані Cookie?

42.Хто є ініціатором запису Cookie?

До занять 4-6.

1. Яким чином виконуються веб-застосування на клієнтському комп'ютері?

2. Яким чином виконуються веб-застосування на веб-сервері?

3. Які обмеження і чому накладаються на клієнтські застосування?

4. Що таке насичене Інтернет-застосування?

5. Які інструменти використовуються для розробки насичених Інтернет-застосувань?

6. Що таке EcmaScript?

7. У чому відмінність між JavaScript і JScript?

8. До якого класу мов належить мова JScript?

9. Яку структуру має програма мовою JScript?

10. Які типи даних підтримуються у JScript?

11. Які оператори використовуються у JScript?

12. У чому полягає особливість роботи з об'єктами у JScript?

13. Що таке VBScript?

14. Що таке Java-аплети?

15.Які переваги мають Java-аплети? Які недоліки у Java-аплетів?

16.Що таке ActionScript?

17.Що таке SilverLight і XAML?

18.Що таке DOM?

19.Що таке DHTML? Чим DHTML відрізняється від динамічно створюваних сторінок?

20.Що таке регулярні вирази?

21.Для чого використовуються регулярні вирази?

До занять 7-9

1.Яким чином можна розширити можливості клієнт-серверної взаємодії у рамках протоколу HTTP?

2.Що таке плагін?

3.Що таке сценарій?

4.Що таке CGI?

5.Які завдання вирішує стандарт CGI?

6.Які етапи містить виконання CGI-сценарію?

7.Яким чином веб-сервер визначає, що вимагається саме виконати програму, а не повернути клієнтові файл з кодом?

8.Яке поле повинно обов'язково розміщуватися в заголовку відповіді сервера?

9.Чим відрізняється обробка запиту клієнта сценарієм залежно від методи запиту?

10.Що таке змінні оточення?

11.Які змінні оточення використовуються сценаріями для отримання даних від клієнта?

12.Перерахуєте мови розробки веб-сценарії і коротко схарактеризуйте їх.

13.Що таке ISAPI?

14.Які ви знаєте типи ISAPI?

15.Назвіть переваги й недоліки ISAPI.

16.До якого класу мов належить JavaScript?

17.Які ви знає сильні сторони мови JavaScript?

18.Які типи масивів використовуються у PHP?

19.Чим відрізняються скалярні масиви від асоціативних?

20.Які зумовлені змінні використовуються у PHP?

21.Що таке дескриптор потоку?

22.Які оператори пошуку й заміни використовуються у Perl?

23.До якого класу мов належить PHP?

24.Які ви знає сильні сторони PHP?

25.Яким чином відбувається упровадження PHP-коду в HTML документ?

26.Які типи даних використовуються у PHP?

27.Які об'єктно-орієнтовані можливості реалізовані у PHP?

Питання для самостійного опрацювання

1.Що означає відкритість середовища розробки Microsoft Visual Studio .NET?

2.Які можливості надає платформа Framework.Net?

3.Що таке CLR?

4.Що таке FCL?

5.Що таке простір імен у .NET?

6. Що таке збірка в .NET? Що таке рішення в .NET?

7. Що нового внесено мовою C#?

8. Які типи даних підтримуються в C#?

9. Які типи масивів використовуються в C#?

10. Які оператори використовуються в C#?

11. Що таке інтерфейси в C#? Які ви знаєте інтерфейси в C#?

12. Що таке серіалізація в C#?

13. Який зв'язок між ASP і ASP.NET?

14. Які можливості надає для розробника ASP.NET?

15. Які простори імен доступні для ASP.NET?

16. У чому сутність моделі розподілу коду представлення і коду реалізації? Як ця модель реалізується в ASP.NET?

17. Які об'єкти ASP.NET використовуються для збереження даних веб-сторінки в проміжках між зверненнями до неї?

18. Які процеси відбуваються при взаємодії користувача із застосуванням ASP.NET?

19. Які елементи WebForms на веб-сторінці реалізуються допомогою ASP.NET?

20. Які елементи WebForms використовуються в ASP.NET для роботи з джерелами даних?

21. Чим зумовлена необхідність появи стандартного інтерфейсу для отримання і відправлення даних джерелам різних типів?

22. Що таке ODBC?

23. Яким чином ураховується специфіка джерела даних при роботі з ODBC?

24. Як вигладає сценарій роботи веб-застосування з джерелом даних?

25. Що таке ADO?

26.Яка модель доступу до джерел даних є основною в Microsoft .NET?

27.Які частини містить ADO.NET ?

28.За допомогою якого класу реалізується можливість роботи з від'єднаними наборами даних в ADO.NET ?

29.Яку роль відіграє постачальник даних в ADO.NET ?

30.Що таке SGML?

31.Які недоліки мови HTML ви знаєте?

32.Що таке DTD?

33.Які переваги має мова XML порівняно з HTML?

34.Назвіть основні недоліки мови XML.

35.Що належить до сімейства технологій XML?

36.Яку роль відіграє XML як метамова?

37.Який зв'язок між XML, SGML і HTML?

38.Що означає правильно побудований XML документ?

39.Що означає дійсний XML документ?

40.Які синтаксичні правила повинні дотримуватися у XML документі?

41.Що таке схема XML? Для чого вона потрібна?

42.Які ви знаєте специфікації схем XML?

43.Яким чином реалізується відображення XML-документа?

44.Для чого потрібні XML-схеми?

45.Що таке DTD-схема?

46.Яким чином описується модель вмісту у DTD?

47.Елементи яких типів використовуються у DTD?

48.Як описуються атрибути у DTD?

49.Які ви знаєте індикатори входження у DTD?

50.Як реалізується включення DTD у XML файл?

51.Що таке XDR? Які саме елементи використовуються в XDR?

52.Які атрибути може мати ElementType в XDR?

53.Які дочірні елементи може мати ElementType?

54.Які індикатори входження і групи змісту використовуються в XDR?

55.Що таке валідатор?

56.Яке призначення має XML DOM?

57.Яким чином описується документ у XML DOM?

58.Чи існують окрім XML DOM альтернативні API для XML документів?

59.Яку роль відіграють події в SAX?

60.Які події можуть оброблятися за допомогою SAX?

61.Для чого потрібні CSS і XSL?

62.Що таке XSL?

63.Що таке XSLT?

64.Що таке XPath?

65.Який зв'язок між XSLT і XPath?

66.Що таке XQuery?

67.Які підходи до веб-інтеграції вам відомі? Для чого потрібна веб-інтеграція?

68.Яку роль відіграє XML у веб-інтеграції і чому?

69.Що таке врапер?

70.Які протоколи використовуються для веб-інтеграції?

71.Що таке веб-сервіс?

72.У чому полягає сутність сервіс-орієнтованої архітектури?

73.Які переваги мають веб-сервіси .NET?

74.Яким чином відбувається реалізація веб-сервісу .NET?

75.Який інструмент використовується для розробки веб-сервісів .NET?

76.На яких стандартах базується робота веб-сервісів?

77.Опишіть, яку роль виконують специфікації WSDL, SOAP, DISCO і UDDI.

78.Що таке CMS? Для чого призначена CMS?

79.Що таке WCMS? Для чого призначена WCMS?

80.Які типи WCMS ви знаєте?

81.Як називається реалізація системи WCMS, пропонована Майкрософтом?

82.Що таке CMF?Для чого використовується CMF?

83.Що таке веб-синдикація?

84.Що таке веб-потік?

85.Що таке агрегування веб-потоків?

86.Яким чином здійснюється взаємодії веб-потоку й агрегатора?

87.Які формати даних використовуються для опису веб-потоків?

88.Що таке RSS-канал?

89. Що таке портал? Які типи веб-порталів ви знаєте?

90. Що таке веб-портал?

91. Що таке портлет?

92. Для чого призначені копроративні портали? Яку структуру має типовий корпоративний портал?

93. Які ви знаєте засоби для створення порталів?

94. Що таке AJAX?

95.У чому полягає сутність позасмугового запиту до веб-серверу?

96.Яка компонента використовується для виконання позасмугових звернень до веб-серверу?

97.Які формати даних використовуються для передачі даних при позасмугових зверненнях?

98.Яким чином реалізується AJAX у рамках Microsoft .NET?

99.Який протокол використовується для досупу до ресурсів Інтернет з мобільних пристроїв?

100.Яка мова розмітки використовується для документів, що завантажуються з веб–мобільними пристроями?

101.Яку структуру має WML–сторінка?

102.Що таке .NET Mobile?

103.Які можливості надає .NET Mobile розробникові веб-додатків?

104.Яким чином здійснюється робота .NET Mobile?

105.Що означає термін Веб 2.0? Яку можливість надають користувачеві Веб 2.0 сайтів?

106.Які технології розглядаються як ключові для Веб 2.0? Назвіть відомі недоліки Веб 2.0?

107.Що таке мешап? Які компоненти містить мешап?

108.Чим мешап відрізняється від простого впровадження даних?

109.Які типи мешапів ви знаєте? Які інструменти пропонує Microsoft для конструювання мешапів?

110.Який сенс має термін „Соціальний Веб”?

111.Що таке соціальне ПЗ? Які групи інструментів можна виділити всередині соціального ПО?

112.Наведіть приклади програмних систем, що належать до соціального ПЗ?

113.Що таке фолксономія, які можливості вона надає?

114.Що таке семантична веб-мережа?

115.Що таке онтологія? Яка мова використовується для опису онтологій?

116.Що таке семантичні веб-сервісм і чим вони відрізняються від звичайних веб-сервісів?

Завдання для самостійного виконання

1.Розробити web-магазин комерційної фірми з урахуванням продажів товарів у базі даних.

2. Розробити web-сайт для поширення програмного забезпечення з фіксацією даних про користувачів і скачувані програми в базі даних.

3. Розробити гостьову книгу зі зберіганням обговорень і відомостей про користувачів у базі даних.

4. Створення web–сервисів.

КОРОТКИЙ ДОВІДНИК ОСНОВНИХ ФУНКЦІЙ

Математичні функції

Abs – повертає модуль числа. *Синтаксис:* `mixed abs(mixed $number);`

round – округлення дробового числа до цілого. *Синтаксис:* `double round(double $val, numder);`

ceil – доповнення дробового числа до наступного цілого. *Синтаксис:* `int ceil(float $number)`

floor – видалення дробової частини числа. *Синтаксис:* `int floor(float $number);`

base_convert – конвертація числа з однієї системи числення в іншу. *Синтаксис:* `string base_convert(string $number, int $frombase, int $tobase);`

decbin – здійснює конвертацію десяткового числа у двійкове. *Синтаксис:* `string decbin(int $number);`

dechex – здійснює конвертацію десяткового числа у шістнадцятиричне. *Синтаксис:* `string dechex(int number);`

decoct – здійснює конвертацію десяткового числа у вісімкове. *Синтаксис:* `string decoct(int number);`

hexdec – здійснює конвертацію шістнадцятиричного числа у десяткове. *Синтаксис:* `int hexdec(string hex_string);`

octdec – здійснює конвертацію вісімкового числа у десяткове. *Синтаксис:* `int octdec(string octal_string);`

deg2rad – здійснює конвертацію градусів у радіани. *Синтаксис:* `double deg2rad(double number);`

rad2deg – здійснює конвертацію радіанів у градуси. *Синтаксис:* `double rad2deg(double number);`

number_format – форматування числа. *Синтаксис:* `number_format($number, $decimals, $dec_point="", $thousands_sep=",");`

srand – здійснює ініціалізацію генератора випадкових чисел. *Синтаксис:* `void srand(int seed);`

getrandmax – здійснює максимально можливе випадкове число. *Синтаксис:* `int getrandmax();`

rand – здійснює генерацію випадкового числа. *Синтаксис:* `int rand([int max [, int min]]);`

mt_rand – функція повертає MT-випадкове число, досить рівномірно навіть для того, щоб використовувати його у криптографії. *Синтаксис*: `int mt_rand(int $min=0, int $max=RAND_MAX);`

mt_srand – настроює MT-генератор випадкових чисел на нову послідовність. *Синтаксис:* `void mt_srand(int seed);`

mt_getrandmax – повертає максимальне MT-випадкове число. *Синтаксис*: `int mt_getrandmax();`

lcg_value – функція генерує випадкове дробове число. *Синтаксис:* `double lcg_value();`

min – ця функція повертає найменше з чисел, заданих в її аргументах. *Синтаксис:* `mixed min(mixed $arg1 [int $arg2, .., int $argn]);`

max – отримання найбільшого аргументу. *Синтаксис:* `mixed max(mixed $arg1 [int $arg2, .., int $argn]);`

sqrt – повертає квадратний корінь з аргументу. *Синтаксис:* `float sqrt(float $arg);`

log – повертає натуральний логарифм аргументу. *Синтаксис:* `float log(float $arg);`

log10 – повертає десятковий логарифм аргументу. *Синтаксис:* `float log10(float $arg);`

exp – повертає e (2,718281828) в мірі $arg. *Синтаксис:* `float exp(float $arg);`

pow – піднесення до ступеня. *Синтаксис:* `float pow(float $base, float $exp);`

sin – повертає синус аргументу. *Синтаксис:* `float sin(float $arg);`

cos – повертає косинус аргументу. *Синтаксис:* `float cos(float $arg);`

tan – повертає тангенс аргументу, заданого в радіанах. *Синтаксис:* `float tan(float $arg);`

acos – повертає арккосинус аргументу. *Синтаксис:* `float acos(float $arg);`

asin – повертає арксинус. *Синтаксис:* `float asin(float $arg);`

atan – повертає арктангенс аргументу. *Синтаксис:* `float atan(float $arg);`

atan2 – отримання арктангенса двох чисел. *Синтаксис:* `float atan2(float $y, float $x);`

pi – повертає число пі - 3,14. *Синтаксис:* `double pi();`

bcadd – складання двох чисел довільної точності. *Синтаксис:* `string bcadd(string left_operand, string right_operand [, int scale]);`

bccomp – порівняння двох чисел довільної точності. *Синтаксис:* `int bccomp(string left_operand, string right_operand, [int scale]);`

bcdiv – операція ділення для двох чисел довільної точності. *Синтаксис:* `string bcdiv(string left_operand, string right_operand [, intscale]);`

bcmod – повертає залишок цілочисельного ділення. *Синтаксис:* `string bcmod(left_operand, string modulus);`

bcmul – операція множення для двох чисел довільної точності. *Синтаксис:* `string bcmul(string left_operand, string right_operand [, int scale]);`

bcpow – піднесення одного числа довільної точності до ступеня іншого. *Синтаксис:* `string bcpow(string x, string y, [int scale]);`

bcscale – установлює точність обчислень. *Синтаксис:* `string bcscale(int scale);`

bcsqrt – отримання квадратного кореня числа довільної точності. *Синтаксис:* `string bcsqrt(string operand [, int scale]);`

bcsub – віднімає одне число довільної точності з іншого. *Синтаксис:* `string bcsub(string left_operand, right_operand [, int scale]);`

Робота з масивами

array – створення й ініціалізація масиву. *Синтаксис:* `array array([mixed ..]);`

range – створює масив і заповнює його значеннями. *Синтаксис:* `array range(int low, int high[, int step]);`

array_reverse – розставляння елементів масиву у зворотному порядку. *Синтаксис:* `array array_reverse(array arr [, bool preserve_keys]);`

shuffle – перемішування елементів масиву. *Синтаксис:* `void shuffle(array arr);`

sort – сортування масиву за збільшенням. *Синтаксис:* `bool sort(array arr [, int sort_flags]);`

rsort – сортування масиву за убуванням. *Синтаксис:* `bool rsort ( array array [, int sort_flags]);`

asort – сортування асоціативного масиву за збільшенням. *Синтаксис:* `bool asort(array arr [, int sort_flags]);`

arsort – сортування асоціативного масиву за убуванням. *Синтаксис:* `bool arsort(array arr [, int sort_flags]);`

ksort – сортування масиву за збільшенням ключів. *Синтаксис:* `bool ksort(array arr [, int sort_flags]);`

krsort – сортування масиву за убуванням індексів. *Синтаксис:* `bool krsort(array arr [, int sort_flags]);`

natsort – виконує „природне" сортування масиву. *Синтаксис:* `void natsort(array arr);`

natcasesort – виконує „природне" сортування масиву без урахування регістру символів. *Синтаксис:* `void natcasesort(array arr);`

uasort – призначене для користувача сортування асоціативного масиву зі збереженням індексних асоціацій. *Синтаксис:* `bool uasort(array arr, function cmp_function);`

uksort – призначене для користувача сортування масиву за ключами. *Синтаксис:* `ool uksort(array arr, function cmp_function);`

usort – колристувацьке сортування значень масиву. *Синтаксис:* `bool usort(array arr, callback cmp_function);`

array_multisort – сортування декількох масивів або багатовимірного масиву. *Синтаксис*: `bool array_multisort(array ar1, [, mixed arg [, mixed .. [, array ..]]]) ;`

reset – здійснює скидання курсора масиву. *Синтаксис:* `mixed reset(array arr);`

end – здійснює перенесення курсора в кінець масиву. *Синтаксис:* `mixed end(array arr);`

next – здійснює перенесення курсора вперед. *Синтаксис:* `mixed next(array arr);`

prev – здійснює перенесення курсора назад і повертає попереднє значени. *Синтаксис*: `mixed prev(array arr);`

current – визначення поточного елемента масиву. *Синтаксис:* `mixed current(array arr);`

pos – визначення поточного елемента масиву. *Синтаксис:* `mixed pos(array arr);`

key – функція повертає індекс поточного елемента масиву. *Синтаксис:* `mixed key(array arr);`

each – отримання поточного елемента масиву. *Синтаксис:* `array each(array arr);`

array_walk – застосування призначеної для користувача функції до елементів масиву. *Синтаксис:* `bool array_walk(array arr, callback function [, mixed userdata]);`

array_flip – змінює місцями індекси та значення масиву. *Синтаксис:* `array array_flip(array arr);`

array_keys – повертає список з ключів масиву. *Синтаксис:* `array array_keys(array arr [, mixed search_value]);`

array_values – видалення асоціативних індексів масиву. *Синтаксис:* `array array_values(array arr);`

in_array – здійснює перевірку масиву на наявність значення. *Синтаксис:* `bool in_array ( mixed needle, array haystack [, bool strict]);`

array_count_values – повертає кількість значень масиву. *Синтаксис:* `array array_count_values(array arr);`

sizeof – повертає число елементів масиву. *Синтаксис:* `int sizeof(array arr);`

count – повертає число елементів у масиві або об'єкті. *Синтаксис:* `int count(mixed var [, int mode]);`

array_sum – повертає суму всіх елементів масиву. *Синтаксис:* `mixed array_sum(array arr);`

array_rand – робить випадкову вибірку індексів масиву. *Синтаксис:* `mixed array_rand(array arr [, int num_req]);`

array_change_key_case – функція встановлює значення ключів масиву у верхній або нижній регістр. *Синтаксис:* `array array_change_key_case(array arr[, int registr]);`

array_combine – функція об'єднує два масиви, причому значення першого становиться ключами, а значення другого–значеннями. *Синтаксис:* `array array_combine(array keys, array values);`

array_key_exists – перевірка існування заданого ключа в масиві. *Синтаксис***:** `bool array_key_exists(mixed key, array search);`

array_diff – визначення виняткового перетину масивів. *Синтаксис:* `array array_diff(array arr1, array arr2 [, array ..]);`

array_diff_assoc – визначення виняткового перетину масивів з урахуванням індексів масивів. *Синтаксис:* `array array_diff_assoc(array arr1, array arr2 [, array ..]);`

array_diff_uassoc – визначення відмінності між масивами за допомогою призначеної для користувача функції з додатковою перевіркою ключів масиву. **Синтаксис:** `array array_diff_uassoc(array arr1, array arr2 [, array .., callback key_compare_func]);`

array_udiff – порівнює масиви, використовуючи призначену для користувача функцію. *Синтаксис:* `array array_udiff( array array1, array array2 [, array .., callback data_compare_func]);`

array_udiff_assoc – порівнює масиви, використовуючи призначену для користувача функцію. *Синтаксис:* `array array_udiff_assoc( array array1, array array2 [, array .., callback data_compare_func]);`

array_udiff_uassoc – порівнює масиви, використовуючи призначену для користувача функцію. *Синтаксис:* `array array_udiff_uassoc ( array array1, array array2 [, array .., callback data_compare_func, callback key_compare_func]);`

array_intersect – визначення включного перетину масивів. *Синтаксис:* `array array_intersect(array arr1, array arr2 [, array ..]);`

array_intersect_assoc – функція повертає перетини значень масивів із збереженням індексів. *Синтаксис:* `array array_intersect_assoc(array array1, array array2 [, array ..]);`

array_merge – злиття масивів. *Синтаксис:* `array array_merge(array arr1, array arr2 [, array ..]);`

array_merge_recursive – рекурсивне злиття складних масивів. *Синтаксис:* `array array_merge_recursive(array arr1, array arr2 [, array ..]);`

array_slice – отримання частини масиву. *Синтаксис:* `array array_slice(array arr, int offset [, int lenght]);`

array_splice – видаляє частину масиву або замінює її частиною іншого масиву. *Синтаксис*: `array array_splice(array arr, int offset [, int lenght [, int replacement]]);`

array_pad – додає до масиву кілька елементів. *Синтаксис:* `array array_pad(array input, int pad_size, mixed pad_value);`

array_pop – витягає і видаляє останні елементи масиву. *Синтаксис:* `mixed array_pop(array arr);`

array_push – додає один або кілька елементів у кінець масиву. *Синтаксис*: `int array_push(array arr, mixed var1 [, mixed var2, .]);`

array_shift – витягає і видаляє перший елемент масиву. *Синтаксис:* `mixed array_shift(array arr);`

array_unshift – додає одне або кілька значень на початок масиву. *Синтаксис:* `int array_unshift(list arr, mixed var1 [, mixed var2, ..]);`

array_unique – видаляє дублюючі значення в масиві. *Синтаксис:* `array array_unique(array arr);`

array_chunk – функція розбиває масив на частини. *Синтаксис:* `array array_chunk(array arr, int size [, bool preserve_keys]);`

array_fill – функція заповнює масив певними значеннями. *Синтаксис:* `array array_fill(int start_index, int num, mixed value);`

array_filter – функція застосовує фільтр до масиву, використовуючи призначену для користувача функцію. *Синтаксис:* `array array_filter(array input [, callback callback]);`

array_map – застосування призначеної для користувача функції до всіх елементів вказаних масивів. *Синтаксис:* `array array_map(mixed callback, array arr1 [, array ..]);`

list – заносить елементи масиву в змінні. *Синтаксис:* `void list ( mixed ..);`

compact – упаковує в масив змінні з поточного контексту. *Синтаксис:* `array compact(mixed varname1 [, mixed $varname2, ..]);`

extract – імпорт елементів масиву в змінні. *Синтаксис:* `int extract (array var_array [, int extract_type [, string prefix]]);`

Строкові функції

chr – повертає один символ з певним кодом. *Синтаксис:* `string chr(int ascii);`

ord – повертає ascii код символа. *Синтаксис:* `int ord(string str);`

trim – видаляє із заданого рядка початкові й кінцеві пробільні символи. *Синтаксис:* `string trim(string str);`

strchr – пошук першого входження символа в рядок. *Синтаксис:* `string strchr(string haystack, string needle);`

strstr – пошук першого входження підрядка в рядок. *Синтаксис:* `string strstr(string haystack, string needle);`

stristr – знаходження першого входження підрядка, не враховуючи регістр. *Синтаксис:* `string stristr(string haystack, string needle);`

strrchr – пошук останнього входження підрядка. *Синтаксис:* `string strrchr(string haystack, string needle);`

strpos – знаходить позицію першого входження підрядка в заданому рядку. *Синтаксис:* `int strpos(string where, string what [, int fromwhere]);`

stripos – знаходить позицію першого входження підрядка в заданому рядку без урахування регістру. *Синтаксис:* `int stripos(string where, string what [, int fromwhere]);`

strrpos – знаходить у заданому рядку останню позицію, у якій знаходиться заданий фрагмент. *Синтаксис:* `int strrpos(string where, string what);`

substr_count – знаходить кількість входжень фрагмента в рядок. *Синтаксис*: `int substr_count(string where, string what);`

strspn – визначає наявність початкових символів у рядку. *Синтаксис:* `int strspn(string str1, string str2);`

strcspn – визначає відсутність початкових символів у рядку. *Синтаксис:* `int strcspn(string str1, string str2);`

strcmp – порівнює рядки. *Синтаксис:* `int strcmp(string str1, string str2);`

strncmp – порівнює початки рядків. *Синтаксис:* `int strncmp(string str1, string str2, int len);`

strcasecmp – порівнює рядки без урахування регістру. *Синтаксис:* `int strcasecmp(string str1, string str2);`

strncasecmp – порівнює початок рядків без урахування регістру. *Синтаксис:* `int strncasecmp(string str1, string str2, int len);`

strnatcmp – здійснює „природне” порівняння рядків. *Синтаксис:* `int strnatcmp(string str1, string str2);`

strnatcasecmp – здійснює „природне” порівняння рядків без урахування регістру. *Синтаксис:* `int strnatcasecmp(string str1, string str2);`

similar_text – здійснює визначення схожості двох рядків. *Синтаксис:* `int similar_text(string firsrt, string second [, double percent]);`

levenshtein – визначення відмінності Левенштейна двох рядків. *Синтаксис:* `int levenshtein(string str1, string str2);`

`int levenshtein(string str1, string str2, int cost_ins, int cost_rep, int cost_del);`

`int levenshtein(string str1, string str2, function cost);`

print – виводить рядок, значення змінної або виразу. *Синтаксис:* `print(string arg)`

echo – проводить виведення одного або декількох значень. *Синтаксис*: `echo(string arg1, string [argn]..);`

printf – виведення відформатованого рядка. *Синтаксис:* `int printf(string format [, mixed args, ..]);`

sprintf – здійснює форматування рядка з підстановкою змінних. *Синтаксис:* `sprintf($format [, args, ..]);`

sscanf – здійснює інтерпретацію рядка згідно з форматом і занесенням значень у змінні. *Синтаксис:* `mixed sscanf(string str, string format [, string var1 ..]);`

substr – повертає ділянку рядка з певною довжиною. *Синтаксис:* `string substr(string str, int start [, int length]);`

str_repeat – повторює рядок певну кількість разів. *Синтаксис:* `string str_repeat(string str, int number);`

str_pad – доповнює рядок іншим рядком до певної довжини. *Синтаксис:* `string str_pad(string input, int pad_length [, string pad_string [, int pad_type]]);`

chunk_split – повертає фрагмент рядка. *Синтаксис:* `string chunk_split(string str [, int chunklen [, string end]]);`

strtok – повертає рядок частинами. *Синтаксис:* `string strtok(string arg1, string arg2);`

explode – здійснює розподіл рядка в масив. *Синтаксис:* `array explode(string seperator, string str [, int limit]);`

implode – здійснює об'єднання масиву в рядок. *Синтаксис:* `string implode(string glue, array pieces);`

join – здійснює об'єднання масиву в рядок. *Синтаксис:* `string join(string glue, array pieces);`

str_replace – замінює в початковому рядку одні підрядки на інші. *Синтаксис:* `string str_replace(string from, string to, string str);`

substr_replace – замінює в початковому рядку одні підрядки на інші. *Синтаксис:* `string substr_replace(string str, string replacement, int start [, int lemgth])`

wordwrap – розбиває початковий текст на рядки з певними завершуючими символами. *Синтаксис:* `string wordwrap(string str [, int width [, string break [, int cut]]])`

strtr – комплексна заміна в рядку. *Синтаксис:* `string strtr(string str, string from, string to); string strtr(string str, array from)`

stripslashes – видалення зворотних слешів. *Синтаксис:* `string stripslashes(string str);`

stripcslashes – перетворення спеціальних символів у їх двійкове представлення. *Синтаксис:* `string stripcslashes(string str);`

addslashes – додавання слешів перед спеціальними символами рядка. *Синтаксис:* `string addslashes(string str);`

addcslashes – форматування рядка слешами у С-уявлення. *Синтаксис:* `string addcslashes(string str, string charlist);`

quotemeta – цитування метасимволів. *Синтаксис:* `string quotemeta(string str);`

strrev – здійснює реверс рядка. *Синтаксис:* string strrev(string str)

nl2br – замінює символи переведення рядка. *Синтаксис:* `string nl2br(string string)`

strip_tags – видаляє з рядка теги. *Синтаксис:* `string strip_tags(string str [, string allowable_tags])`

get_meta_tags – функція шукає і обробляє всі теги <META>. *Синтаксис:* `array get_meta_tags(string filename, int use_include_path);`

get_html_translation_table – функція повертає таблицю трансляції, яка використовується функціями htmlspecialchars() і htmlentities(). *Синтаксис:* `string get_html_translation_table(int table [, int quote_style]);`

htmlspecialchars – здійснює перетворення спецсимволів у HTML-уявлення. *Синтаксис:* `string HtmlSpecialChars(string str [, int quote_style]);`

htmlentities – здійснює конвертацію символів, HTML-уявлення. *Синтаксис:* `string htmlentities(string str [, int quote_style]);`

hebrev – конвертація логічного тексту Hebrew у той, що відображається. *Синтаксис:* `string hebrev(string hebrew_text [, int max_chars_per_line]);`

quoted_printable_decode – перетворення цитованого рядка в 8-бітову. *Синтаксис:* `string quoted_printable_decode (string str);`

strtolower – здійснює перетворення символів рядка в нижній регістр. *Синтаксис***:** `string strtolower(string str);`

strtoupper – здійснює перетворення заданого рядка у верхній регістр. *Синтаксис:*`string strtoupper(string str);`

ucfirst – здійснює перетворення першого символа рядка у верхній регістр. *Синтаксис:* `string ucfirst(string str);`

ucwords – здійснює перетворення першого символа кожного слова рядка у верхній регістр. *Синтаксис:* `string ucwords(string str);`

setlocale – установка регіональних налаштувань. *Синтаксис:* `string SetLocale(string category, string locale);`

convert_cyr_string – перетворює рядок з одного кодування кирилиці в іншу. *Синтаксис:* `string convert_cyr_string(string str, string from, string to);`

bin2hex – здійснює перетворення символьних даних у шістнадцятиричний вигляд. *Синтаксис:* `string bin2hex(string str);`

iconv – перетворить рядок з одного кодування в іншу. *Синтаксис:* `string iconv(string from, string to, string str);`

`mb_convert_encoding`

`mb_convert_encoding ( string $str, string $to_encoding [, mixed $from_encoding ] )`

parse_url – обробляє URL і повертає його компоненти. *Синтаксис:* `array parse_url(string url);`

parse_str – заносить рядки URL в змінні. *Синтаксис:* `void parse_str(string str [, array arr]);`

urlencode – кодування рядка в URL. *Синтаксис:* `string urlencode (string str);`

urldecode – декодування рядка з URL *Синтаксис:* `string urldecode (string str) ;`

rawurlencode – кодування URL. *Синтаксис:* `string RawUrlEncode(string str);`

rawurldecode – здійснює декодування URL. *Синтаксис:* `string rawurldecode(string str);`

base64_encode – кодує дані в кодуванні MIME base64. *Синтаксис:* `string base64_encode(string data);`

pack – пакетування даних у двійковий рядок. *Синтаксис:* `string pack(string format [, mixed $args, ..]);`

unpack – розпаковує дані з двійкового рядка. *Синтаксис:* `array unpack(string format, string data);`

strlen – повертає довжину рядка. *Синтаксис:* `int strlen(string str);`

count_chars – повертає інформацію про символи рядка. *Синтаксис:* `mixed count_chars(string str [, int mode])`

md5 – отримання рядка-хеша MD5. *Синтаксис:* `string md5(string str);`

crc32 – отримання поліміналу рядка crc32. *Синтаксис:* `int crc32(string str);`

crypt – здійснює симетричне шифрування. *Синтаксис:* `string crypt(string str [, string salt]);`

metaphone – здійснює обчислення метафон-хеша. *Синтаксис:* `string metaphone(string str);`

soundex – обчислення хеша схожості вимови. *Синтаксис:* `string soundex(string str).`

Функції дати та часу

checkdate – перевіряє правильність дати/часу. *Синтаксис:* `int checkdate(int month, int day, int year);`

date – формат локального часу/дати. *Синтаксис:* `string date(string format [, int timestamp]);`

localtime – отримує інформацію про дату/часу. *Синтаксис:* `array localtime([int timestamp [, bool is_associative]]);`

strftime – орматує час згідно з локальними установками. *Синтаксис:* `string strftime(string format [, int timestamp])`

getdate – отримує інформацію про дату/часу. *Синтаксис:* `array getdate(int timestamp);`

gmdate – отримання дати у форматованому рядку для часу GMT. *Синтаксис:* `string gmdate(string format, int timestamp);`

gmstrftime – форматування локальних часу/дати. *Синтаксис:* `string gmstrftime(string format, int timestamp);`

mktime – отримує тимчасову мітку UNIX для дати. *Синтаксис:* `int mktime([int hour] [,int minute] [,int second] [,int month] [,int day] [,int year] [, int is_dst]);`

gmmktime – аналог функції time() для часу GMT. *Синтаксис:* `int gmmktime(int hour, int minute, int second, int month, int day, int year [, int is_dst]);`

time – отримання часу в секунадх. *Синтаксис:* `int time();`

microtime – повертає поточну тимчасову мітку UNIX в мікросекундах. *Синтаксис:* `string microtime();`

strtotime – лексичне перетворення рядка часу в Unix timestamp. *Синтаксис:* `int strtotime(string time [, int now]).`

Робота з файлами

fopen – відкриває файл і „прив'язує" його до дескриптора. *Синтаксис:* `int fopen(string $filename, string $mode, bool $use_include_path=false);`

tmpfile – створює новий тимчасовий файл з унікальним ім'ям і відкриває його на читання і запис. *Синтаксис:* `int tmpfile();`

fclose – закриває файл, відкритий заздалегідь функцією fopen(). *Синтаксис:* `int fclose(int $fp);`

fread – читає з відкритого файла певну кількість символів. *Синтаксис:* `string fread(int $f, int $numbytes);`

fwrite – запис у файл. *Синтаксис:* `int fwrite(int $f, string $str);`

fgets – читає з файла один рядок, що закінчується символом нового рядка n. *Синтаксис***:** `string fgets(int $f, int $length);`

fputs – повний аналог fwrite(). *Синтаксис:* `int fputs(int $f, string $str);`

feof – покажчик кінця файла. *Синтаксис:* `int feof(int $f);`

fseek – установлює покажчик файла на певну позицію. *Синтаксис:* `int fseek(int $f, int $offset, int $whence=SEEK_SET);`

ftell – повертає положення покажчика файла. *Синтаксис:* `int ftell(int $f);`

file_exists – перевіряє існування файла, що викликається. *Синтаксис:* `bool file_exists(string filename);`

filetype – повертає тип файла. *Синтаксис:* `string filetype(string filename);`

is_file – перевірка існування звичайного файла. *Синтаксис:* `bool is_file(string filename);`

is_dir – перевірка існування каталогу. *Синтаксис:* `bool is_dir(string filename);`

is_link – перевірка існування символічного посилання на файл. *Синтаксис:* `bool is_link(string filename);`

is_readable – перевірка існування файла, доступного для читання. *Синтаксис:* `bool is_readable(string filename);`

is_writeable – перевірка існування файла, доступного для запису. *Синтаксис:* `bool is_writeable(string filename);`

is_executable – перевірка існування файла, що запускається. *Синтаксис:* `bool is_executable(string filename);`

is_uploaded_file – перевірка існування файла, завантаженого методом HTTP POST. *Синтаксис:* `bool is_uploaded_file(string filename);`

stat – функція збирає разом усю інформацію, що видається операційною системою для зазначеного файла, і повертає її у вигляді масиву. *Синтаксис:* `array stat(string $filename);`

fileatime – повертає час останнього доступу до файла. *Синтаксис:* `int fileatime(string filename);`

filemtime – повертає час останньої зміни файла або false у разі відсутності файла. *Синтаксис:* `int filemtime(string $filename);`

filectime – повертає час створення файла. *Синтаксис:* `int filectime(string $filename);`

filesize – повертає розмір файла в байтах або false, якщо файла не існує. *Синтаксис:* `int filesize(string $filename);`

touch – установлює час модифікації. *Синтаксис:* `int touch(string $filename [, int $timestamp]);`

basename – виділяє ім'я файла з шляху. *Синтаксис:* `string basename(string $path);`

dirname – виділяє ім'я каталогу. *Синтаксис:* `string dirname(string $path);`

tempnam – генерує унікальне ім'я файла в певному каталозі. *Синтаксис:* `string tempnam(string $dir, string $prefix);`

realpath – перетворить відносний шлях на абсолютний. *Синтаксис:* `string realpath(string $path);`

copy – копіює файл. *Синтаксис:* `bool copy(string $src, string $dst) ;`

unlink – видалення файла. *Синтаксис:* `bool unlink(string $filename);`

file – прочитує файл і розбиває його за рядками. *Синтаксис:* `list file(string $filename);`

ftruncate – усікає файл. *Синтаксис:* `bool ftruncate(int $f, int $newsize);`

fflush – негайний запис усіх змін у файлі. *Синтаксис:* `void fflush(int $f);`

set_file_buffer – установлює розмір буфера. *Синтаксис:* `int set_file_buffer(int $f, int $size);`

flock – блокування файла. *Синтаксис:* `bool flock(int $f, int $operation [, int $wouldblock]).`

Функції для роботи з каталогами

mkdir – створення каталогу. *Синтаксис:* `bool mkdir(string $name, int $perms);`

rmdir – видалення каталогу. *Синтаксис:* `bool rmdir(string $name);`

chdir – зміна поточного каталогу. *Синтаксис*: `int chdir(string $directory);`

getcwd – повний шлях. *Синтаксис:* `string getcwd();`

diskfreespace – визначає вільний простір у каталозі. *Синтаксис:* `float diskfreespace (string directory);`

Робота з базами даних

mysql_connect – установлює мережне з'єднання з базою даних MySQL. *Синтаксис:* `int mysql_connect([string $host-`

```
name[: port][:/path/to/socket][, [,string $username
[, string $password]]]);
```

mysql_pconnect – установлює стійке мережне з'єднання з базою даних MySQL. *Синтаксис:* `int mysql_pconnect([string $hostname[: port][:/path/to/socket][, [,string $username [, string $password]]]);`

mysql_close – закриває встановлене попереднє з'єднання з базою даних. *Синтаксис:* `int mysql_close ([int link_identifier];`

mysql_change_user – змінює параметри підключення. *Синтаксис:* `int mysql_change_user(string user, string password [, string database [, int link_identifier]]);`

mysql_list_dbs – повертає список бази даних на сервері. *Синтаксис:* `int mysql_list_dbs([int link_identifier]);`

mysql_db_name – повертає ім'я бази даних зі списку. *Синтаксис:* `int mysql_db_name(int result, int row [, mixed field]);`

mysql_select_db – вибір однієї бази даних MySQL. *Синтаксис:* `int mysql_select_db (string database_name [, int link_identifier]);`

mysql_create_db – створення бази даних MySQL. *Синтаксис:* `int mysql_create_db(string dbname [, int link_identifier]);`

mysql_drop_db – видалення бази даних MySQL. *Синтаксис:* `int mysql_drop_db(string database_name [, int link_identifier]);`

mysql_list_tables – повертає список таблиць у БД. *Синтаксис:* `int mysql_list_tables(string database [, int link_identifier]);`

mysql_tablename – повертає ім'я таблиці в БД. *Синтаксис:* `int mysql_tablename(int result, int i);`

mysql_query – посилає запит базі даних MySQL. *Синтаксис:* `int mysql_query(string query [, int link_identifier]);`

mysql_db_query – посилає запит до зазначеної бази даних MySQL. *Синтаксис:* `int mysql_db_query(string database, string query [, int link_identifier]);`

mysql_num_rows – повертає кількість рядків у результаті запиту. *Синтаксис:* `int mysql_num_rows(int result);`

mysql_insert_id – отримує вставлений ідентифікатор. *Синтаксис:* `int mysql_insert_id([int $link_identifier]);`

mysql_data_seek – установлює покажчик поточного рядка. *Синтаксис:* `int mysql_data_seek(int result, int row_number);`

mysql_free_result – знищує набір записів. *Синтаксис:* `int mysql_free_result(int result);`

mysql_result – отримання певного поля результату. *Синтаксис:* `int mysql_result(int result, int row [, mixed field]);`

mysql_fetch_array – витягає з результату черговий запис і вміщує його до асоціативного масиву. *Синтаксис:* `array mysql_fetch_array(int result [, int result_type]);`

mysql_fetch_row – записує запис у нумерований масив. *Синтаксис:* `array mysql_fetch_row(int result);`

mysql_fetch_object – отримання запису у властивостях об'єкта. *Синтаксис*: `object mysql_fetch_object(int result);`

mysql_fetch_lengths – повертає довжину елемента запису. *Синтаксис:* `array mysql_fetch_lengths(int result);`

mysql_fetch_field – повертає інформацію про властивості об'єкта і про поле запису. *Синтаксис:* `object mysql_fetch_field(int result [, int field_offset]);`

mysql_field_seek – здійснює переміщення курсора до вказаного поля. *Синтаксис:* `int mysql_field_seek(int result, int field_offset);`

mysql_field_name – повертає ім'я поля. *Синтаксис:* `string mysql_field_name(int result, int filed_index);`

mysql_field_table – повертає ім'я таблиці, з якої було витягнуто поле. *Синтаксис:* `string mysql_field_table(int result, int field_offset);`

mysql_field_len – повертає довжину поля. *Синтаксис:* `int mysql_field_len(int result, int filed_offset);`

mysql_field_type – повертає тип набору записів у результаті. *Синтаксис:* `string mysql_field_type(int result, int filed_offset);`

mysql_field_flags – ця функція повертає прапори, які були використані при створенні зазначеного поля в таблиці. *Синтаксис:* `string mysql_field_flags(int result, int field_offset);`

mysql_list_fields – повертає список полів таблиці. *Синтаксис:* `int mysql_list_fields(string dbname, string tblname [, int link_identifier]);`

mysql_num_fields – ця функція повертає число полів в одному рядку результату, тобто число колонок у результаті. *Синтаксис:* `int mysql_num_fields(int result);`

mysql_errno – повертає номер останньої помилки. *Синтаксис:* `int mysql_errno ([int link_identifier]);`

mysql_error – повертає повідомлення про помилку. *Синтаксис:* `string mysql_error ([int link_identifier ]).`

Рекомендована література

1.Айзекс С. Dynamic HTML / С. Айзекс. – СПб.: БХВ-Петербург, 2003. – 496 с.

2.Браун М. HTML 3.2 / М. Браун. — СПб: БХВ-Петербург, 2001. – 1040 с.

3.Гаевский А. Ю. Самоучитель по созданию Web-страниц: HTML, Javascript и Dynamic HTML / А. Ю. Гаевский, В. А. Романовский. - К. : АСК, 2002. – 472 с.

4.Дронов В. JavaScript в Web-дизайне // В. Дронов / Издательство: БХВ-Петербург, 2001. – 880 с.

5.Дунаев С. Сам себе Web-мастер / С.Дунаев. – СПб. : БХВ-Петербург, 2002. – 512 с.

6.Дунаев С. Технологии Интернет программирования/ С.Дунаев. – СПб. : БХВ-Петербург, 2003. – 480 с.

7.Матросов А. HTML 4.0 // А. Матросов, А. Сергеев, М. Чаунин. – СПб. : БХВ-Петербург, 2001. – 672 с.

8.Симонович С. Специальная информатика / С. Симонович, Г. Евсеев, А. Алексеев. – М. : АСТ-ПРЕСС Инфорком – Пресс Москва, 2000. – 480 с.

9.Симонович С. Интернет у вас дома / С. Симонович, В. Мураховский. – М. : АСТ-ПРЕСС Инфорком – Пресс Москва, 2001. – 431 с.

10.Старыгин А. XML. Разработка Web / А. Старыгин. – СПб.: БХВ-Петербург, 2003. – 592 с.

11.Шапошников И. Интернет–программирование / И. Шапошников. – СПб.: БХВ-Петербург, 2003. - 544 с.

12.Шапошников И. Web-сайт своими руками / И. Шапошников. – СПб.: БХВ-Петербург, 2003. – 224 с.

13.Феррара А. Программирование web-сервисов для NET / Алекс Феррара, Мэтью Мак-Дональд. – СПб. : БХВ-Петербург, 2003. – 432 с.

14.Бумфрей Ф. XML. Новые преспективы WWW / Ф. Бумфрей, О. Дирецо, Й. Дакетт и др. ; пер. с англ. – М. : ДМК, 2000. – 688 с.

15.Зайден М. XML для электронной коммерции / Марк Зайден. М. : Изд-во: „Бином. Лаборатория знаний”, 2003. – 480 с.

16.Тейлор Р. Реализация XML Web-служб на платформе Microsoft .NET Microsoft .NET XML Web Services / Роберт Тейлор. – М. : Вильямс, 2002. – 464 с.

17.Сироткин С. А. Самоучитель WML и WMLScript / С. А. Сироткин, И. В. Чалышев, С. Е. Воробьев. – СПб. : БХВ-Петербург, 2001. –240 с.

18.Шорт С. Разработка XML Web-сервисов средствами Microsoft. NET (+ CD-ROM) / Скотт Шорт. – СПб. : БХВ-Петербург, 2003. – 480 с.

19.Шапошников И. В. Web–сервисы Microsoft .Net / И. В. Шапошников. – СПб. : БХВ–Петербург, 2002. – 336 с.

20.Храмцов П. Б. Основи WEB–технологий / П. Б. Храмцов, С. А. Брик, А. М. Русак, А. И.Сурин. – М. : ИТУИТ.РУ, 2003. – 512 с.

21.Роджерс Д. Програмування на Microsoft JScript.NET / Д. Роджерс – М. : Вільямс, 2002. – 352 с.

22.Дунаев В. В. JavaScript / В. В.Дунаев. – СПб. : Пітер, 2003. – 394 с.

23.Форту Б. Освой самостійно регулярні вирази. 10 хвилин на урок // Б. Форту / М.: "Вільямс", 2005. – 184 с.

24.Мельтцер К. Разработка CGI -приложений на Perl / К. Мельтцер – М. : Вільямс, 2001. – 395 с.

25.Шапошников И. PHP 5.1 / И. Шапошников : учебный курс. – СПб. : Пітер, 2007. – 192 с.

26.Шеперд Д. Освой самостійно XML за 21 день / Д. Шеперд. – М. : Вільямс, 2002. – 432 с.

27.Старыгин А. XML : разработка Web – приложений / А. Старыгин. – СПб. : БХВ-Петербург, 2003. – 585 с.

28.Троелсен Э. C# і платфома .NET / Э. Троелсен // Бібліотека програміста. – СПб. : Пітер, 2007. – 796 с.

29.Беллиньясо М. Разработка Web–приложений в среде ASP.NET 2.0: с примерами на C# / М. Беллиньясо – Диалектика. – 2007. – 640 с.

30.Шортів С. Разработка XML Web–сервисов способами Microsoft .NET / С. Шортів. – СПб. : БХВ-Петербург, 2003. – 480 с.

31.Ньюкомер Э. Веб-сервисы: XML, WSDL, SOAP і UDDL / Э. Ньюкомер, С. Шортів. – СПб. : БХВ-Петербург, 2003. – 480 с.

32.И. Салмре Программирование мобильніх устройств на платформе .NET Compact Framework / И. Салмре. – М. : Вільямс, 2006. – 736 с.

33.Крейн Д. AJAX в действии: технология - Asynchronous JavaScript and XML = AJAX in Action // Д. Крейн, Э. Паскарелло, Д. Джеймс. – М. : Вільямс, 2006 – 640 с.

34.Wikipedia, the free encyclopedia [Електронний ресурс] / Режим доступу: http://en.wikipedia.org/wiki/Main Page.

35.Вікіпедія [Електронний ресурс]/Режим доступу: http://ru.wikipedia.org/wiki.

36.W3CSchools Online Web Tutorials [Електронний ресурс] / Режим доступу: http://www.w 3schools.com.

37.Інтернет Університет інформаційних технологій [Електронний ресурс] / Режим доступу: http://www.intuit.ru/catalog.

Покажчик

Навчально-методичне видання

Фоменко Андрій Вікторович
Козуб Галина Олександрівна

ТЕХНОЛОГІЯ WEB-ПРОГРАМУВАННЯ PHP, MYSQL

Курс лекцій з дисципліни „Технологія Web-програмування" для студентів 5 курсу спеціальності „Інформатика" та для студентів 3 курсу спеціальності „Комп'ютерна інженерія"

Редактор – Фоменко А. В., Козуб Г. О.
Комп'ютерний макет – Козуб Г. О.
Коректор – Лєсовець Н. М.

Здано до склад. 19.12.2010 р. Підп. до друку 19.01.2011 р.
Формат 60x84 1/16. Папір офсет. Гарнітура Times New Roman.
Друк ризографічний. Ум. друк. арк. 12,56. Наклад 300 прим. Зам. № 169.

Видавець і виготовлювач
Видавництво Державного закладу
„Луганський національний університет імені Тараса Шевченка"
вул. Оборонна, 2, м. Луганськ, 91011. Тел./факс: (0642) 58-03-20
e-mail: alma-mater@list.ru
Свідоцтво суб'єкта видавничої справи ДК № 3459 від 09.04.2009 р.