• Кіндрат П.В. •

РОЗРОБКА ВЕБ-ЗАСТОСУВАНЬ ЗАСОБАМИ PHP

Міністерство освіти і науки України
Рівненський державний гуманітарний університет
Кафедра інформаційно-комунікаційних технологій та методики викладання інформатики

Кіндрат П.В.

Розробка веб-застосувань засобами PHP

Навчально-методичний посібник

Рівне – 2021

УДК 004.777:004.43 (072)
К 41
Р 65

Кіндрат П.В. Розробка веб-застосувань засобами PHP : навчально-методичний посібник для здобувачів вищої освіти – Рівне : Рівненський державний гуманітарний університет, 2021. - 310с.

Рецензенти:

Войтович І.С. – доктор педагогічних наук, професор кафедри інформаційно-комунікаційних технологій та методики викладання інформатики Рівненського державного гуманітарного університету.

Малежик П. М. - доктор педагогічних наук, доцент кафедри програмної інженерії Національниого педагогічного університету імені М.П.Драгоманова.

Навчально-методичний посібник спрямований як на формування теретичного підгрунтя щодо принципів побудови програмних рішень, так і на закріплення отриманих знань у практичній площині.

Посібник може бути використаний як початківцями, так і студентами, що володіють певним досвідом у розробці веб-застосувань.

Навчально-методичний посібник затверджено та рекомендовано до друку на засіданні Вченої Ради рівненського державного гуманітарного університету, протокол № 3 від 28 березня 2019 року.

ЗМІСТ

Передмова

Розвиток мережевих технологій сприяє все більшому їх інтегруванню в наше життя. При цьому розробка і підтримка власних мережевих ресурсів і сервісів стає невід'ємною вимогою сучасного бізнесу. Водночас, незважаючи на стрімкий розвиток нових напрямків та мов програмування, сфера розробки веб-застосувань продовжує залишається під домінуючим впливом однієї мови програмування. Цією мовою є PHP, яка застосовується в порядку 70% існуючих проектів. Така популярність і відданість розробників обумовлена як ґрунтовними функціональними можливостями, які дозволяють реалізовувати складні проекти, так і постійним вдосконаленням мови, що дозволяє враховувати зміни в тенденціях та стилях веб-розробки та залишати мову відповідною сучасним потребам розробників і їх клієнтів.

Попри візуальну простоту і зрозумілість мови PHP, студенти що її вивчають можуть зіштовхнутись з багатьма підводними каменями. Такі колізії будуть особливо чутливими для тих, чий попередній досвід програмування зосереджувався на компільованих мовах програмування, а виконуваний код не передбачав застосування мережевих ресурсів.

Даний навчально-методичний посібник покликаний систематизувати основні положення створення програмних додатків на мові PHP: принципи побудови мережевих застосувань, механізми передачі даних між веб-ресурсами, обробки даних користувача, методи динамічного управління вмістом веб-сторінок, роботи з системою керування базами даних, а також побудови об'єктно-орієнтованої моделі сайту. Це дозволить швидше та ефективніше засвоювати навчальний матеріал, а також розвинути практичні навички зі створення веб-ресурсів.

Засвоєння матеріалу посібника як аудиторно (під керівництвом викладача), так і вдома (самостійно) дозволить сформувати та розвинути навички:

– розробки базових веб-додатків;

– використання методів передачі і обміну даними;

– використання HTML-форм для забезпечення введення, виведення і обробки даних веб-додатків;

– реалізації роботи з файлами і каталогам;

– реалізації підключення веб-додатків до бази даних з метою зберігання та обміну інформацією між базою даних і додатком.

При вивченні матеріалу посібника важливо усвідомлювати, що він не є всеосяжним і не розкриває усіх особливостей мови програмування PHP. Його метою є стимулювання інтересу до веб-розробки як напрямку програмування, а також побудова міцного фундаменту що дозволить впевненіше розбудовувати рівні професійних компетентностей та навичок майбутнього фахівця.

Розділ 1 Виникнення та розвиток веб-застосувань
Тема 1. Основні засоби веб-технологій
1.1 Засоби розмітки веб-документів
HTML - мова розмітки тексту

Всесвітня павутина складається з веб-сторінок. які створено у форматі HTML *(HyperTexJMarkupLanguage,* «мова розмітки гіпертексту»). HTML - це фундаментальна, базова технологія Інтернету.

HTML не є мовою програмування, це мова розмітки тексту, що використовує спеціальні оператори - теги *(tag)* чи інша назва дескриптори *(descriptor)* для розмітки текстового документа. Ці позначки вказують в якому вигляді буде виведено текстовий чи інший елемент у вікні браузера.

HTML дозволяє формувати на сторінці сайту текстові блоки, додавати до них зображення, організовувати таблиці, керувати відтворенням кольору, додавати до дизайну сайту звуковий супровід, організовувати гіперпосилання з переходом до інших розділів сервера або звертатися до інших ресурсів Інтернету і компонувати всі ці елементи між собою. Документи, що створено лише засобами HTMLмають розширення *htm* або*html.*

Однією з основних функціональних особливостей мови HTML, завдяки якої вона і отримала свою назву, є гіперпосилання.

Гіперпосилання (*Hyperlink*) – це базовий функціональний елемент HTML- документу, який реалізовує зв'язок певного об'єкту веб-сторінки з іншим об'єктом. Для гіперпосилання може використовуватися як фрагмент тексту, так і графічний об'єкт, а сам гіперзв'язок можна встановлювати як між об'єктами одного сайту, так і між об'єктами, що розміщені на різних сайтах Інтернету.

HTML є мовою, що лише інтерпретується, тому, для виконання коду, його не потрібно компілювати. Інтерпретатор мови втілено в браузер. і він «компілює» код безпосередньо під час відкривання документа. Якщо в коді сторінки виявлено

помилку, інтерпретатор, зазвичай, не видає відповідного попередження, а просто ігнорує весь «помилковий» рядок, що може зіпсувати зовнішній вигляд завантаженої сторінки. Це є важливим для розробників, тому слід бути уважним під час складання HTML-коду і ретельно тестувати результати своєї роботи.

CSS - каскадна таблиця стилів

CSS (*CascadingStyleSheets)* – це технологія опису зовнішнього вигляду документа, що створено засобами HTML. XML і XHTML.

CSS використовується для завдання кольорів, шрифтів, розташування елементів сторінки тощо. До появи CSSоформлення веб-сторінок вказувалося безпосередньо в HTML-коді сторінки. Проте, з появою CSSстало можливим принципове розділення змісту і представлення документа. За рахунок такого нововведення стало можливим легке застосування єдиного стилю оформлення для кількох сторінок сайту, а також швидка зміна цього оформлення.

Переваги:

1. Застосування кількох варіантів дизайну сторінки для різних пристроїв перегляд)-. Наприклад, для відображення на екран монітора - дизайн буде розраховано на велику ширину. У разі друкування документу не буде роздруковане меню сайту, а у разі перегляду у мобільному комп'ютері чи телефоні, меню буде виведено після вмісту сторінки.

2. Зменшення часу завантаження сторінок сайту- за рахунок перенесення правил представлення даних до окремого CSS-файлу. В цьому випадку браузер завантажує лише структуру документа і дані, що містяться на сторінці. CSS-файл з правилами представлення цих даних завантажується браузером лише один раз і зберігається в кеші браузера.

3. Простота подальшої зміни дизайну. Не потрібно виправляти кожну сторінку, достатньо лише змінити кілька правил у CSS-файлі.

4. Додаткові можливості оформлення. Наприклад, за допомогою CSS- правил можна застосувати обтікання певного

блоку текстом або зробити так. щоб меню фіксовано знаходилося в певному місці прн перегортанні сторінки.

Недоліки:

Різні браузери можуть в різний спосіб інтерпретувати правила CSS і, відповідно, по різному відображати одні і ті ж фрагменти сторінки.

DHTML - динамічна мова розмітки тексту

DHTML (*DynamicHTML)* - це набір засобів, які реалізовують інтерактивність веб-сторінки без звертання до серверу. Тобто, певні дії відвідувача можуть спричинити зміну зовнішнього вигляду і вмісту сторінки.

DHTML побудовано на об'єктній моделі документа DOM (*DocumentObjectModel).* яка розширює традиційний статичний HTML-документ. DOM забезпечує динамічний доступ до вмісту документа, його структури і стилів. В DOM кожен елемент веб-сторінки є об'єктом, який надається до змін. DOM не визначає нових тегівв чи атрибутів, а лише забезпечує можливість програмного управління всіма тегами. атрибутами і каскадними листами стилів CSS.

XML- розширена мова розмітки

Основну увагу в мові XMLзосереджено на даних. Тут, структурна розмітка даних і представлення даних є строго розділеними. Основні причини створення XML:

– Спроба надати могутні засоби форматування і структуризації даних для широкого кола розробників.

– Необхідність в стабільній реалізації мови стркгтуризації документів, де можна легко створювати допоміжні інструменти, що є доступними для звичайних користувачів.

XML є метамовою - спеціальною мовою, якою можна скласти повний опис класу інших мов, якими створено документи. XML містить набір правил, що дозволяють створювати унікальні застосування і підмножини даних.

В багатьох розробників виникають певні труднощі у зв'язку з абстрактністю XML і довільним використанням його методів. Насправді. XML - є досить логічною і добре

організованою структурою, вона має чіткий синтаксис, що змушує строго дотримуватися певних правил. Починати вивчення XML слід із застосування вже отриманих знань про HTML. XML. як і HTML, використовує теги та атрибути.

Як мова розмітки веб-документів XML має свої особливості:

– Гнучкість. XML дозволяє обробляти унікальні даш. і незалежно від їх характеру надає адекватні методи для їх зберігання і обробки.

– Можливість налаштування. Гнучкість XML безпосередньо пов'язана з можливістю визначати власні дескриптори, необхідність в яких виникає в процесі рішення задачі.

– Узгодженість. XML відрізняється синтаксичною цілісністю і строгою структурою.

Практично всі сучасні браузери підтримують XML. Вона здатна цілком замінити HTML, як засіб розмітки веб-сторінок. хоча вивчення XML є складнішим за вивчення HTML.

XSLT - розширена мова перетворення листів стилів

Мова XSLT *(extensibleStylesheetLanguageTransformations)* є транслятором, за допомогою якого можна вільно модифікувати початковий текст. XLST грає вирішальну роль в затвердженні XML як універсальної мови збереження і передачі даних. Область застосування XSLT є широкою - від електронної комерції до безпровідного Веб.

Фактична збірка результуючого документа відбувається, коли початковий документ і лист стилів XSLT передаються до синтаксичного аналізатору XSLT (XSLT-процесора).

При використанні XSLT в середовищі Веб синтаксичний аналіз може відбуватися або з боку користувача (в браузері), або з боку сервера.

Перетворення XSLT засновано на шаблонах. XSLT-процесор аналізує початковий документ і намагається знайти відповідний XSL-шаблон. Якщо такий шаблон знайдено, то виконуються інструкції, що містяться всередині нього.

1.2 Засоби управління вмістом

JavaScript - мова сценаріїв

JavaScript - це мова, що призначена для написання сценаріїв для інтерактивних HTML-сторінок. Мова JavaScript не має жодного відношення до мови Java. Java розроблено фірмою SUN. а JavaScript - фірмою Netscape Communication Corporation. Первинною назвою було «LiveScript», але, з огляду на велику популярність мови Java, назву «LiveScript» з комерційних міркувань було змінено на (JavaScript».

JavaScript не призначено дтя створення автономних застосувань. Програмний код на JavaScript вписується безпосередньо в текст HTML- документа і інтерпретується браузером в міру завантаження цього документа. За допомогою JavaScript можна динамічно змінювати текст завантаженого HTML-документу і реагувати на події, які пов'язані з діями відвідувача або змінами стану документа чи вікна.

Важливою особливістю JavaScript є об'єктна орієнтованість. Програмісту є доступними численні об'єкти, такі, як документи, гіперпосилання. форми, фрейми тощо. Об'єкти характеризуються описовою інформацією (властивостями) і можливими діями (методами).

PHP - мова створення сценаріїв

PHP *(.PersonalHomePage)* - мова створення сценаріїв, яка давно переросла свою назву. Перша версія PHP була створена Рас мусом Лердорфом в 1994 р. і була набором інструментів дтя відстеження поведінки відвідувачів сайту. З часом PHP з набору інструментів перетворилася на повноцінну мову програмування, а її назву було змінено як *PHP HyperTextPreprocessor* (препроцесор гіпертексту PHP).

PHP - це серверна мова.

Конструкції PHP. що вставлено в HTML-текст, виконуються сервером при кожному відвідуванні сторінки. Результат обробки конструкцій разом із звичайним HTML-текстом передається браузеру.

Основними конкурентами PHP є ASP *(ActiveServerPages)* від компанії Microsoft і ColdFusion від компанії Allaire.

У порівнянні з ними PHP має ряд переваг, зокрема:

– Висока продуктивність. PHP-програми працюють швидше, ніж ASP.

– Функціональність. Розробку PHP-програми можна відокремити від власне розробки веб-сторшки. що спрощує працю і для програміста і для дизайнера.

– Ціна. Мова PHP є абсолютно безкоштовною.

– Простота у використанні. Програмісти, що мають досвід програмування на поширених мовах швидко зрозуміють синтаксис PHP.

– Переносимість. Один і той же PHP-код можна використовувати як в середовищі Windows, так і на платформах UNIX.

ASP - активні сторінки сервера

ASP *(ActiveServerPages)*–технологія, що є аналогічною до JavaScript і PHP. Для того, щоб зробити інтерактивну веб-сторінку із застосуванням макромови ASP, необхідно вбудувати в її код відповідний скрипт, що віддалено нагадує Java і CI. Скрипт інтерпретується і виконується безпосередньо на сервері, після чого до браузера відправляється вже готовий HTML-документ з результатами роботи сценарію ASP. Для сторінок, що містять конструкції ASP. не має значення, яке програмне забезпечення встановлено на комп'ютері користувача, але принципове значення має тип сервера, який має підтримувати дану технологію.

Ajax - технологія для взаємодії з сервером без перевантаження сторінок

Ajax (*AsynchronousJavascriptAndXML)* це підхід до створення веб- застосувань за допомогою наступних технологій:

– Стандартизоване представлення засобами XHTMLi CSS.

– Динамічне відображення і взаємодія з користувачем за допомогою DOM.

– Обмін і обробка даних у вигляді XML и XSLT.

– Широке застосування мови сценаріїв JavaScript.

– Асинхронні запити за допомогою об'єкту XMLHttpRequest.

В стандартному веб-застосуванні обробкою всієї інформації займається сервер, браузер відповідає лише за взаємодію з користувачем, передачу запитів і виведення отриманих даних у форматі HTML.

В Ajax-застосуванні між користувачем і сервером з'являється ще один посередник - програмний механізм (рушій) Ajax. Він визначає, які запита можна обробити з боку клієнта, а які необхідно виконувати на сервері.

Поведінка сервера теж змінилася. Якщо раніше на кожен запит сервер видавав нову сторінку, то тепер вш надсилає лише ті дані, які потрібні клієнту, а рушій Ajax в браузері формує з них HTML-код.

Асинхронність виявляється в тому, що далеко не кожен запит користувача скеровується до сервера, причому зворотне теж справедливо - далеко не кожна реакція сервера обумовлена запитом користувача. Велику частину запитів формує рушій Ajax, причому є можливим, щоб він передбачав запита користувача.

Зрозуміло, що за такою схемою робота міняється якісне навантаження на сервер - якщо раніше запитів було мало, але кожен з них вимагав значних ресурсів (серверу потрібно витягнути інформацію з бази даних, сформувати з неї веб-сторінку і відправити до браузера), то тепер завдання сервера спрощується (формувати веб-сторінки не потрібно, та і об'єм даних, що передаються є значно меншим), але доводиться більше обробляти запитів.

Створювати застосування на Ajax є трудомістким і складним завданням. Потрібно написати і відлагодитн на JavaScript рушій з десятьох чи двадцятьох тисяч рядків коду плюс реалізувати сервери) частину.

Тема 2. Архітектура типових веб-застосувань

2.1 Поняття архітектури ПЗ

Архітектура програмного забезпечення (англ. softwarearchitecture) – це структура програми або обчислювальної системи, яка містить програмні компоненти, видимі зовні властивості цих компонентів, а також відносини між ними. Цей термін також відноситься до документування архітектури програмного забезпечення. Документування архітектури ПЗ спрощує процес комунікації між зацікавленими особами , дозволяє зафіксувати прийняті на ранніх етапах проектування рішення про високорівневий дизайн системи і дозволяє використовувати компоненти цього дизайну і шаблони повторно в інших проектах.

Компоненти інформаційної системи по виконуваних функціях можна розділити на три шари: шар представлення, шар бізнес-логіки і шар доступу до даних.

Шар представлення - все, що пов'язано зі взаємодією з користувачем: натиснення кнопок, рух миші, отрисовка зображення, виведення результатів пошуку і так далі

Бізнес логіка - правила, алгоритми реакції додатка на дії користувача або на внутрішні події, правила обробки даних.

Шар доступу до даних - зберігання, вибірка, модифікація і видалення даних, пов'язаних з вирішуваною додатком прикладним завданням

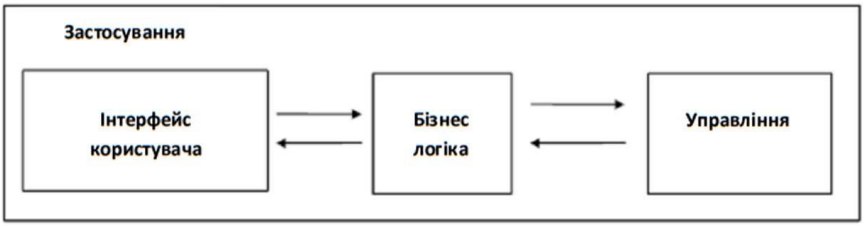


Рисунок 2.1 Компоненти інформаційної системи

2.2 Види архітектур сучасних програмних додатків

Історично першими з'явилися комп'ютерні системи з **нейтралізованою архітектурою** додатків. При використанні

цієї архітектури усе програмне забезпечення автоматизованої системи виконується централізовано на одному комп'ютері, що виконує одночасно багато завдань і що підтримує велику кількість користувачів. На цьому комп'ютері повністю здійснюється процес введення/ виведення інформації, а також її прикладна обробка. В якості центрального комп'ютера для такої системи може застосовуватися або велика ЕОМ(що називається також мейнфреймом) або так звана МІНІ-ЕОМ. До подібного комплексу підключаються периферійні пристрої для введення/виведення інформації від кожного користувача.

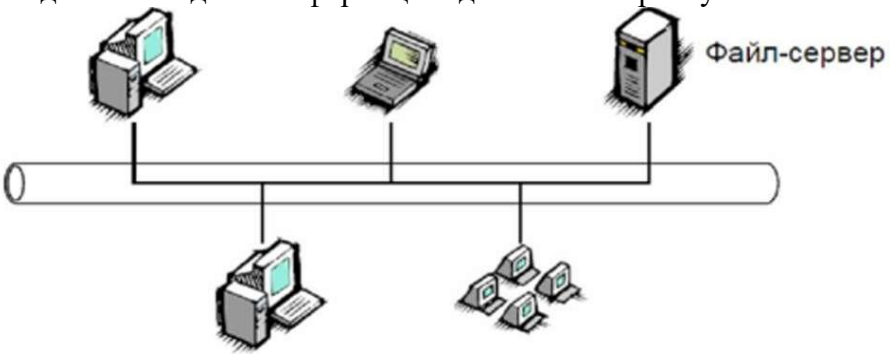


Рисунок 2.2 Файл-серверна архітектура

Наступний етап розвитку архітектури ПЗ – **файл-сервер** – це виділений сервер, призначений для виконання файлових операцій введення- виводу і зберігаючий файли будь-якого типу. Як правило, має великий об'єм дискового простору, реалізованому у формі RAID– масиву для забезпечення безперебійної роботи і підвищеної швидкості запису і читання даних.

Файл-серверні додатки – додатки, схожі по своїй структурі з локальними застосуваннями що використовують мережевий ресурс для зберігання даних у вигляді окремих файлів. Функції сервера у такому разі зазвичай обмежуються зберіганням даних(можливо також зберігання виконуваних файлів), а обробка даних відбувається виключно на стороні клієнта. Кількість клієнтів обмежена десятками зважаючи на неможливість одночасного доступу на запис до одного файлу.

Проте клієнтів може бути в рази більше, якщо вони звертаються до файлів виключно в режимі читання.

Рисунок 2.3 Модель файлового сервера

Переваги:

- низька вартість розробки;
- висока швидкість розробки;
- невисока вартість

оновлення і зміни ПЗ. Недоліки:

- зростання числа клієнтів різко збільшує об'єм трафіку і навантаження на мережі передачі даних;
- високі витрати на модернізацію і супровід сервісів бізнес-логіки на кожній клієнтській робочій станції;
- низька надійність системи.

Архітектура клієнт-сервер

Архітектура є одним із архітектурних шаблонів програмного забезпечення та є домінуючою концепцією у створенні розподілених мережних застосунків і передбачає взаємодію та обмін даними між ними. Вона передбачає такі основні компоненти:

• набір серверів, які надають інформацію або інші послуги програмам, які звертаються до них;

• набір клієнтів, які використовують сервіси, що надаються серверами;

• мережа, яка забезпечує взаємодію між клієнтами та серверами. Сервери є незалежними один від одного. Клієнти також функціонують паралельно і незалежно один від одного. Немає жорсткої прив'язки клієнтів до серверів. Більш ніж типовою є ситуація, коли один сервер одночасно обробляє запити від різних клієнтів; з другого боку, клієнт може звертатися то до одного сервера, то до іншого. Клієнти мають знати про доступні сервери, але можуть не мати жодного уявлення про існування інших клієнтів.

Загальноприйнятим є положення, що клієнти та сервери - це перш за все програмні модулі. Найчастіше вони знаходяться на різних комп'ютерах, але бувають ситуації, коли обидві програми - і клієнтська, і серверна, фізично розміщуються на одній машині; в такій ситуації сервер часто називається локальним

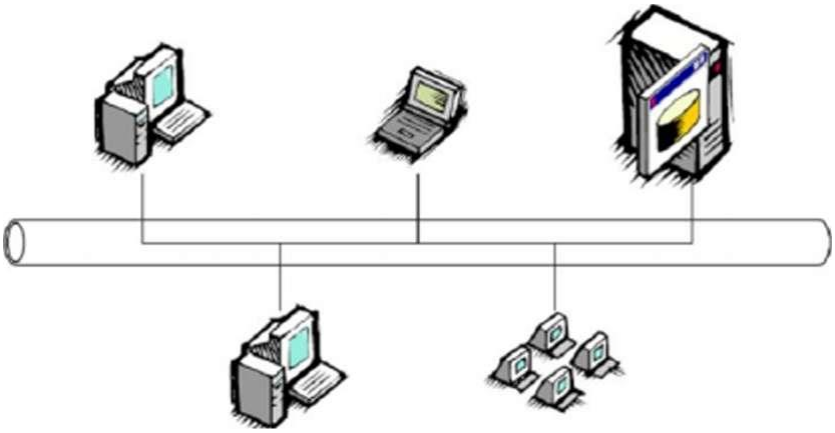


Рисунок 2.4 Архітектура клієнт-сервер

Модель клієнт-серверної взаємодії визначається перш за все розподілом обов'язків між клієнтом та сервером. Логічно можна виокремити три рівні операцій:

• рівень представлення даних, який по суті являє собою інтерфейс користувача і відповідає за представлення даних користувачеві і введення від нього керуючих команд;

• прикладний рівень, який реалізує основну логіку застосунку і на якому здійснюється необхідна обробка інформації;

• рівень управління даними, який забезпечує зберігання даних та доступ до них.

Дворівнева клієнт-серверна архітектура передбачає взаємодію двох програмних модулів - клієнтського та серверного. В залежності від того, як між ними розподіляються наведені вище функції, розрізняють:

• модель тонкого клієнта, в рамках якої вся логіка застосунку та управління даними зосереджена на сервері. Клієнтська програма забезпечує тільки функції рівня представлення;

• модель товстого клієнта, в якій сервер тільки керує даними, а обробка інформації та інтерфейс користувача зосереджені на стороні клієнта. Товстими клієнтами часто також називають пристрої з обмеженою потужністю: кишенькові комп'ютери, мобільні телефони та ін.

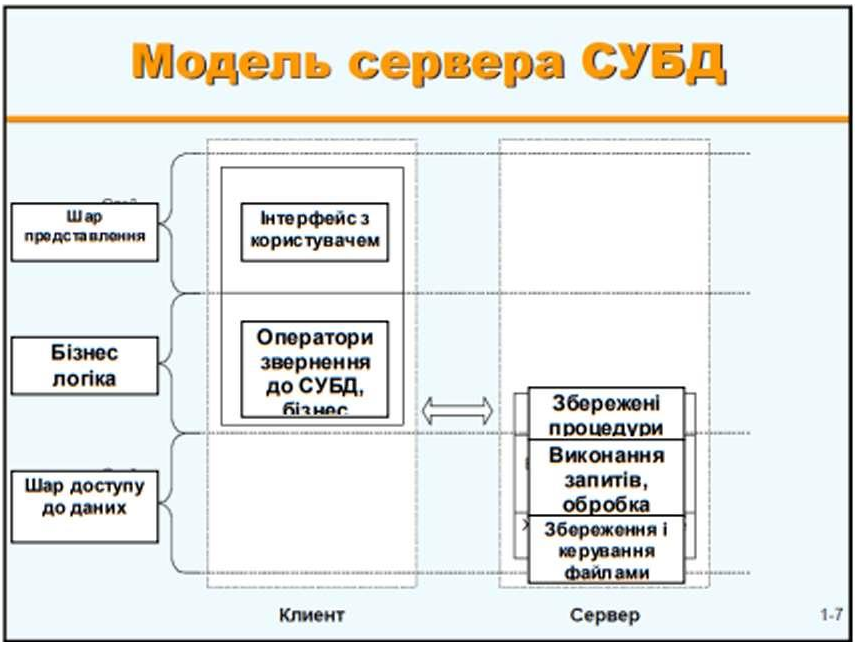


Рисунок 2.5 Архітектура клієнт-сервер

Плюси:

• Повна підтримка розрахованої на багато користувачів роботи

• Гарантія цілісності даних Мінуси:

• Бізнес логіка додатків залишилася в клієнтському ПЗ. При будь-якій зміні алгоритмів, потрібно оновлювати призначене для користувача ПЗ на кожному клієнтові.

• Високі вимоїн до пропускної спроможності комунікаційних каналів з сервером, що перешкоджає використання клієнтських станцій інакше як в локальній мережі.

• Слабкий захист даних від злому, особливо від недобросовісних користувачів системи.

• Висока складність адміністрування і налаштування робочих місць користувачів системи.

• Необхідність використати потужні ПК на клієнтських місцях.

• Висока складність розробки системи із-за необхідності виконувати бізнес-логіку і забезпечувати призначений для користувача інтерфейс в одній програмі

Трирівнева архітектура

Трирівнева архітектура, або триланкова архітектура (англ. three–tierабо англ. Multiticrarchitecture) – архітектурна модель програмного комплексу, що припускає наявність в нім трьох компонентів : клієнтського застосування(що зазвичай називається «тонким клієнтом» або терміналом),сервера додатків, до якого підключено клієнтське застосування, і сервера бази даних, з яким працює сервер додатків.

• *Клієнт* – це інтерфейс ний (зазвичай графічний) компонент, який представляє перший рівень, власне додаток для кінцевого користувача. Перший рівень не повинен мати прямих зв'язків з базою даних(за вимогами безпеки), бути навантаженим основною бізнес-логікою(за вимогами масштабованості) і зберігати стан додатка(за вимогами надійності). На перший рівень може бути винесена і зазвичай виноситься проста бізнес-логіка: інтерфейс авторизації, алгоритми шифрування, перевірка значень, що вводяться, на допустимість і відповідність формату, нескладні операції(сортування, угрупування, підрахунок значень) з даними, вже завантаженими на термінал.

• *Сервер додатків* розташовується на другому рівні. На другому рівні зосереджена більша частина бізнес-логіки. Поза ним залишаються фрагменти, що експортуються на термінали(див. вище), а також занурені в третій рівень процедури, що зберігаються, і тригери.

• *Сервер бази даних* забезпечує зберігання даних і виноситься на третій рівень. Звичайно це стандартна реляційна або об'єктно-орієнтована СУБД. Якщо третій рівень є базою даних разом з процедурами, що зберігаються, тригерами і схемою, що описує додаток в термінах реляційної моделі, то другий рівень будується як програмний інтерфейс, що зв'язує клієнтські компоненти з прикладною логікою бази даних.

У простій конфігурації фізично сервер додатків може бути поєднаний з сервером бази даних на одному комп'ютері, до якого по мережі підключається один або декілька терміналів.

У «правильній»(з крапки зору безпеки, надійності, масштабування) конфігурації сервер бази даних знаходиться на виділеному комп'ютерній кластері), до якого по мережі підключені один або декілька серверів додатків, до яких, у свою чергу, по мережі підключаються термінали.

Переваги

В порівнянні з клієнт-серверною або файл-серверною архітектурою можна виділити наступні перевага трнрівневої архітектури :

- масштабованість
- конфігурується – ізольованість рівнів один від одного дозволяє(при правильному розгортанні архітектури) швидко і простими засобами переконфігурувати систему при виникненні збоїв або при плановому обслуговуванні на одному з рівнів
- висока безпека
- висока надійність
- низькі вимоги до швидкості каналу(мережі) між терміналами і сервером додатків
- низькі вимоги до продуктивності і технічних характеристик терміналів, як наслідок зниження їх вартості. Терміналом може виступати не лише комп'ютер, але і. наприклад, мобільний телефон.

Рисунок 2.6 Архітектура клієнт-сервер

Недоліки

Недоліки витікають з переваг. По порівнянню с кліент-серверної або файл-серверної архітектурою можна виділити наступні недоліки трирівневої архітектури :

- більш висока складність створення додатків:
- складніше в розгортанні і адмініструванні:
- високі вимоги до продуктивності серверів додатків і сервера бази даних, а. означає, і висока вартість серверного устаткування;
- високі вимоги до швидкості каналу(мережі) між сервером бази даних і серверами додатків.

2.3 Розподілені інформаційні системи.

У літературі можна знайти різні визначення розподілених систем, причому жодне з них не є задовільним і не узгоджується з іншими. Для наших завдань вистачить досить вільної характеристики. Розподілена система – це набір незалежних обчислювальних машин, що представляється їх користувачам єдиною об'єднаною системою. У цьому визначенні

обмовляються два моменти. Перший відноситься до апаратури: усі машини автономні.

Другий торкається програмного забезпечення: користувачі думають, що мають справу з єдиною системою. Важливі обидва моменти. Пізніше в цій главі ми до них повернемося, але спочатку розглянемо деякі базові питання, що стосуються як апаратного, так і програмного забезпечення.

Характеристики розподілених систем:

1.Від користувачів приховані відмінності між комп'ютерами і способи зв'язку між ними. Те ж саме відноситься і до зовнішньої організації розподілених систем.

2.Користувачі і додатки однаково працюють в розподілених системах, незалежно від того, де і коли відбувається їх взаємодія.

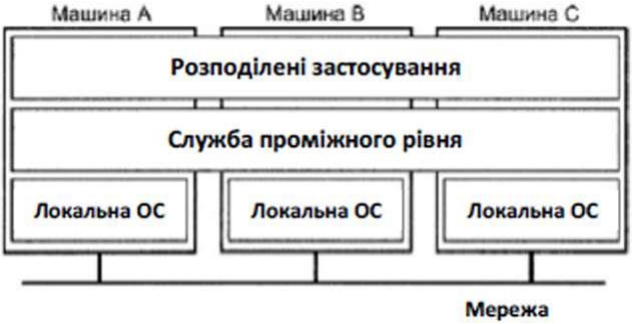


Рисунок 2.7 Модель розподіленої системи

Розподілені системи повинні також відносно легко піддаватися розширенню, або масштабуванню. Ця характеристика е прямим наслідком наявності незалежних комп'ютерів, але в тон же час не вказує, яким чином ці комп'ютери насправді об'єднуються в єдину систему.

Розподілені системи зазвичай існують постійно, проте деякі їх частини можуть тимчасово виходити з ладу. Користувачі і додатки не повинні повідомлятися про те. що частини системи замінені або полагоджені або. що додані нові для підтримки додаткових користувачів.

Для того, щоб підтримати представлення системи в єдиному виді, організація розподілених систем часто включає додатковий рівень програмного забезпечення, що знаходиться між верхнім рівнем, на якому знаходяться користувачі і додатки, і нижнім рівнем, що складається з операційних систем.

Сервісно-орієнтована архітектура

Сервісно-орієнтована архітектура (англ. Service-orientedarchitecture. SOA) архітектурний шаблон програмного забезпечення, модульний підхід до розробки програмного забезпечення, заснований на використанні розподілених, слабко пов'язаних замінних компонентів, оснащених стандартизованими інтерфейсами дія взаємодії за стандартизованими протоколам.

Дуже часто становлення того чи іншого підходу супроводжується появою невірних або хибних трактувань, як це було, наприклад, з концепцією федеративного сховища даних. Не оминуло стороною це і сервіс-орієнтовану архітектуру. Так вважає представник компанії BEA Джерімі Уестерман (JeremyWesterman). Саме тому в одній із своїх статей, присвячених SOA. він спеціально зупиняється на «проблемних місцях» і наводить докладні пояснення:

- SOA не є чимось новим: ГГ-відділи компаній успішно створювали і розгортали додатки, що підтримують сервіс-орієнтовану архітектуру, вже багато років – задовго до появи XML і Web-сервісів.
- SOA– це не технологія, а спосіб проектування і організації інформаційної архітектури та бізнес-функціональності.
- Купівля найновіших продуктів, що реалізують XML і Web-сервісів. не означає побудову додатків згідно з принципами SO A.

Джерімі Уестерман дає наступне визначення SOA: це парадигма, призначена для проектування, розробки та управління дискретних одиниць логіки (сервісів) в обчислювальному середовищі. Застосування цього підходу вимагає від розробників проектування додатків як набору

сервісів, навіть якщо переваги такого рішення відразу неочевидні. Розробники повинні вийти за межі своїх додатків і подумати, як скористатися вже існуючими сервісами, або вивчити, як 'іх сервіси можуть бути використані fe колегами.

SOAпідштовхує до використання альтернативних технологій і підходів (таких як обмін повідомленнями) для побудови додатків за допомогою **зв'язування** сервісів, а не за допомогою написання нового програмного коду. У цьому випадку, при належному проектуванні, застосування повідомлень дозволяє компаніям своєчасно реагувати на зміну ринкових умов – настроювати процес обміну повідомленнями, а не розробляти нові програми. Ще до недавніх пір термін «сервіс-орієнгована архітектура» був синонімом «Web-**сервіс».** SOA– виклик Web-**сервісів** за допомогою засобів і мов управління бізнес-процесами. SOA– це термін, який з'явився для опису виконуваних компонентів – таких як web-сервіси – які можуть викликатися іншими програмами, які виступають у якості клієнтів або споживачів цих сервісів. Ці сервіси можуть бути повністю сучасними – або навіть застарілими – прикладними програмами, які можна активізувати як чорний ящик. Від розробника не потрібно знати, як працює програма, необхідно лише розуміти, які вхідні та вихідні дані потрібні, і як викликати ці програми для виконання. У найзагальнішому вигляді SOA припускає наявність трьох основних учасників: постачальника сервісу, споживача сервісу та реєстру сервісів. Взаємодія учасників виглядає досить просто: постачальник сервісу реєструє свої сервіси в реєстрі, а споживач звертається до реєстру із запитом.

Для використання сервісу необхідно дотримуватися угоди про інтерфейс для звернення до сервісу – інтерфейс повинен не залежати від платформи. SOA реалізує масштабованість сервісів – можливість додавання сервісів, а також їх модернізацію. Постачальник сервісу і його споживач виявляються непов'язаними – вони спілкуються за допомогою повідомлень. Оскільки інтерфейс повинен не залежати від платформи, то і технологія, використовувана для визначення повідомлень,

також повинна не залежати ви платформи. Тому, як правило, повідомлення є XML-документами, які відповідають XML-схемі.

Дійсно, відкриті стандарти, що описують XML і Web-сервсів. дозволяють застосовувати SOA до всіх технологій і додатків, встановлених в компанії. Як відомо. \\ᵀеЬ-сервісн базуються на широко поширених і відкритих протоколах: HTTP. XML. UDDL WSDL і SOAP. Саме пі стандарти реалізують основні вимоги SOA– по-перше, сервіс повинен піддаватися динамічному виявленню і викликом (UDDI. WSDLі SOAP), по-друге, повинен використовуватися незалежний ви платформи інтерфейс (XML). Нарешті. HTTP забезпечує функціональну сумісність.

Переваги використання SOA Перш ніж. перерахувати переваги використання SOA буде доречним нагадати, що переваги бувають різні: стратегічні і тактичні. SOA має ряд переваг як стратегічних, так і тактичних.

Стратегічна цінність SOA:

– Скорочення часу реалізації проектів, або «часу виходу на ринок».

– Підвищення продуктивності.

– Більш швидка і менш дорога інтеграція додатків і інтеграція B2B

– зупинимося більш детально на цьому пункті.

Тактичні переваги SOA:

– Простіша розробка і впровадження додатків.

– Використання поточних інвестицій.

– Зменшення ризику, пов'язаного з впровадженням проектів в області автоматизації послуг і процесів.

– Можливість безперервного поліпшення наданої послуги. Скорочення числа звернень за технічною підтримкою.

– Підвищення показника повернення інвестицій.

2.4 Базові концепції Веб-програмування

Концепція тонкого клієнта

Концепція тонкого клієнта інформаційної системи клієнт-серверної архітектури ґрунтується на організації рівня представлення на клієнтському комп'ютері, рівня обчислень – на сервері програмних додатків, а рівень даних – на сервері баз даних. Рівень представлення інформації користувачу здійснюється за допомогою стандартних програм поступу до джерел інформації, організованих згідно Інтернет технології – Web-браузерів. Рівень обчислень здійснюється на основі серверного програмного забезпечення, що реалізовує функціонування Інтернет вузлів, надаючи доступ до спеціалізованих джерел інформації – Web-сторінок. Таким чином, технологія створення програмних додатків згідно з концепцією тонкого клієнта технології клієнт-сервер зводиться до програмування цих сторінок, яке відоме під назвою Web-програмування.

Web-сторінки створюються за допомогою мови розмітки гіпертексту HTML (Hypertext Markup Lenguage). Він складається з набору позначень – тегів, які використовуються для розмітки тексту, що знаходиться між ними. Теги обмежуються символами <> – початок розмітки та </> – кінець розмітки. Наприклад:

<B> Текст буде виділений жирним шрифтом </B>

Сторінки HTML прийнято розділяти на статичні та динамічні.

Статичні сторінки однаково відображаються для всіх користувачів і не надають можливості змінювати їх вміст. Тобто Web-браузери користувачів мають нагоду відображати тільки той HTML код цих сторінок, який задав розробник. Статичні Web-сторінки є простими файлами, що містять текст розмічений тегами. Їх можна створювати за допомогою практично будь-якого текстового редактора в будь-якій операційній системі. Оскільки статичний продукт не дозволяє користувачу індивідуалізуватися і змінювати дані, його створення не співвідноситься з процесом програмування, а мову HTML не прийнято вважати безпосередньою мовою програмування.

Динамічні сторінки надають користувачам інформацію, яка відрізняється від перегляду до перегляду, і зміст яких залежить від того, кому вони призначені. Їх застосування дозволяє забезпечити двосторонній обмін інформацією між сервером та клієнтом. Динамічні web-сторінки проходять цикл обробки на сервері перед відправкою клієнту. Програмне забезпечення, що реалізує процес цієї обробки, є результатом Web-програмування.

В якості прикладу може бути приведена гіпотетична програма, яка модифікує запрошувані клієнтом статичні сторінки, використовуючи параметри одержаного запиту та сховище даних. Тобто, достатньо підготувати статичні сторінки – шаблони і, перед відправкою клієнтам, програмно модифікувати їх HTML код, підставляючи в нього значення, одержані з бази даних. Розробка подібного програмного забезпечення ґрунтується на застосуванні різних технологічних підходів, розвиток яких відбувався впродовж останнього десятиріччя.

Динамічні Веб-стрінки

Більшість сторінок на ранніх стадіях розвитку Інтернету була статичними. Подальший успішний розвиток «глобальної павутини» відбувся, багато в чому, завдяки розвитку технології динамічних сторінок. Цей розвиток був, здебільшого, стимульований вимогою користувачів Інтернету бути активними дійовими особами інформаційного простору. Наприклад, вони прагнуть замовляти товари в інтернет-магазинах, брати участь в аукціонах, одержувати інформацію про рух засобів на банківських рахунках і т.д.. Динамічні сторінки задовольняють ці потреби завдяки здатності підлаштовуватись під конкретного користувача, а також, реагувати на його дії в браузері.

Початок еволюції мов програмування, здатних динамічно змінювати вміст Web-сторінки, завдячує використанню, так званих, мов скриптів, що виконуються безпосередньо на обчислювальних ресурсах клієнта. Програмний код скриптових мов призначається для виконання тільки під управлінням

відповідного програмного інтерпретатора. Найвідомішими з цих мов вважаються JavaScript і VBScript. Скрипти на цих мовах вбудовуються в код HTML, який сервер посилає браузеру. Сценарії, що виконуються на стороні клієнта, виділяються тегами <SCRIPT> та </SCRIPT>. Браузер інтерпретує цей код і показує користувачу результат.

Слід відмітити, що технологія скриптового програмування не надає можливості реалізувати обчислювальний процес, складність якого перевищує мінімальний рівень. Наприклад, якщо потрібно виконати складний аналіз інформації, що міститься в базі даних, часто не представляється можливим відправити користувачу потрібну кількість її вмісту.

Незважаючи на це, розробка скриптів досягла значного рівня популярності завдяки отриманій можливості успішного розв'язання ряду задач за їх участю. Наприклад, скрипти можуть перевірити правильність запиту, що вводиться в інтерфейс сторінки, і тоді не доведеться перенавантажувати сервер обробкою неправильних запитів. Деякі програмісти створюють на JavaScript анімаційні ефекти. Однак, можливості виконання сценаріїв на стороні клієнта недостатньо для створення повноцінних динамічних сторінок. Навіть якщо на сторінці використовується JavaScript і анімовані картинки .GIF, у наш час її прийнято вважати статичною.

Динамічна Web-сторінка повинна створюватися в потрібний момент часу програмою, що виконується на Інтернет-сервері. Для цього широко застосовується механізм шлюзів CGI (Common Gateway Interface). Спочатку користувач дістає доступ до статичної сторінки з формою, яка реалізує для нього інтерфейс, і задає адресу (URL) виконуваного додатку. Додаток розташовується на відповідній адресі серверної платформи і є одним з виконуваних файлів програм, написаних мовами прикладного програмування (наприклад, на C++/C#). Цей додаток приймає по протоколу HTTP дані з вхідного потоку, проводить необхідні обчислення і, залежно від їх результатів, записує у вихідний потік результуючу Web-сторінку. Тобто,

користувачу, у відповідь на його запит, посилається HTML-код, який Web-додаток згенерував спеціально для нього.

Web-додатки, що викликаються користувачами за технологією CGI, завантажуються в оперативну пам'ять комп'ютера, на якому встановлене серверне програмне забезпечення, а при завершенні – віддаляється з пам'яті. При збільшенні кількості користувачів це негативно позначається на масштабованості. У даному випадку, під масштабованістю розуміється можливість плавного зростання часу затримки відповіді програмної системи на запит у процесі різкого зростання кількості одночасно працюючих користувачів.

Для вирішення цієї проблеми Microsoft була запропонована альтернатива — ISAPI (Internet Server Application Programming Interface). Замість монолітних виконавчих файлів, використовуються DLL-библиотеки. Код DLL знаходиться в пам'яті весь час і, для кожного запиту, замість процесів, створює нитки виконання. Усі нитки використовують єдиний програмний код, який виконується в єдиному процесі Інтернет-серверу. Це дозволяє підвищити продуктивність і масштабованість.

Для реалізації динамічних web-сторінок також застосовуються технології скриптових мов, що виконуються на стороні серверу. Найвідомішими з них є: ASP (Active Server Pages) – активні серверні сторінки та PHP (Personal Home Pages) – персональні домашні сторінки. Принцип їх роботи розглянемо на технології ASP, що була розроблена фірмою Microsoft у 90-х роках 20-го століття.

Виконання скриптового коду файлу ASP підтримується ISAPI-розширенням Інтернет-серверу. У конфігурації Інтернет-серверу визначаються способи обробки файлів з різними розширеннями. Для обробки файлу з розширенням ASP в установках Інтернет-серверу визначається файл Asp.dll. Файли ASP відправляються до нього на обробку: на вхід Asp.dll поступає потік коду ASP, а на виході генерується потік HTML-коду.

Цей процес реалізується за наступною схемою. Код HTML тегів ASP-файлу обробник Asp.dll поміщає у вихідний потік без змін. Код, який заключений у тег <%...%>, сигналізує Asp.dll, що він повинен підлягати обробці. Виконується скрипт на мові, яка вказана у відповідній директиві ASP-файлу – Language. Здебільшого це – JavaScript і VBScript. Результат обробки відповідного до мови інтерпретатора – код HTML додається до вихідного потоку Інтернет-серверу і стає частиною HTML-сторінки, що відправляється на браузер користувача.

Від початку, технологія ASP була обмежена по своїх можливостях, оскільки основувалась на використанні скриптових мов, які поступаються за функціональністю мовам програмування. Крім того, код ASP був вбудований у HTML у вигляді спеціальних тегів, що створювало плутанину, бо HTML-код, зазвичай, створюють дизайнери, які відповідають за оформлення сторінки, а ASP – програмісти, які реалізовують її функціональність.

У ході еволюції технології ASP, ці недоліки були усунені. У 2000 році на конференції розробників в якості складової частини нової технології .NET, Microsoft представив ASP+. З виходом першої версії каркасу Framework.NET вона увійшла до його складу у вигляді структурної компоненти і отримала назву ASP.NET.

Вважається, що технологію ASP.NET не можна розглядати як продовження скриптової технології ASP. Розробники позиціонують її, як концептуально нову технологію, що створена в рамках ідеології відкритого програмування Microsoft – .NET. У ASP.NET закладено все для того, щоб зробити весь цикл розробки Web-додатку швидшим, а підтримку його функціонування – простішою. ASP.NET заснована на об'єктно-орієнтованій технології. Проте, вона зберегла модель використання ASP: готову програму достатньо розмістити в директорії, яка прописана у Web-сервері, і вона працюватиме.

У ASP.NET з'явилося багато нових функцій, а ті, що були раніше в ASP, значно вдосконалені. Зокрема, в ASP.NET використовуються компільовані мови. Під час компіляції

перевіряється синтаксична коректність початкового тексту. Скомпільований у проміжну мову код виконується швидше за некомпільований (скриптовий), незалежно від мови, яка була використана. Крім того, компільовані мови підтримують сувору типізацію.

Компіляція відбувається на сервері в момент першого звернення користувача до сторінки. Якщо програміст змінив текст сторінки, програма перекомпілюється автоматично. При написанні коду можна використовувати набір компонентів, що поставляються разом з Framework.NET .

Платформа Framework.NET надає додаткам середовище виконання і, у той же час, сама безпосередньо взаємодіє з операційною системою. Середовище виконання реалізує інтерфейс ASP.NET-додатків, на якому, у свою чергу, базуються Web-форми – ASP.NET-сторінки. Інтерфейс Framework.NET дозволяє стандартизувати звернення до програмної системи і надає середовище для швидшої та зручнішої її розробки. CLR забезпечує єдиний набір сервісів для всіх мов.

Завдяки наведеним якостям, ASP .NET сьогодні розглядається не як мова програмування, а як технологія, що дозволяє програмувати на різних мовах, компілятори яких підтримує каркас Framework.NET. Тому для її освоєння необов'язкове попереднє знання якоїсь певної мови програмування, так само як і знання ASP. Проте, оскільки мова C# була спеціально створена для платформи .NET, її використання дозволить у більш повній мірі застосувати її концепції та методи.

Контрольні питання

1. Розкрийте суть еволюції епох Веб.
2. Основні принципи Веб 2.0
3. Переваги і недоліки Веб 2.0
4. Наведіть приклади сайтів Веб 2.0
5. Порівняйте Веб 2.0 та Веб 3.0
6. Переваги і недоліки Веб 3.0
7. Наведіть приклади сайтів Веб 3.0

8. Назвіть основні засоби веб-технологій.
9. Що таке архітектура програмного забезпечення
10. З чого складається інформаційна система
11. Які є види сучасного ГГЗ?
12. Порівняйте файл-серверну та клієнт-серверну архітектуру.
13. Переваги і недоліки трирівневої архітектури.
14. Принцип роботи розподілених систем.
15. SOA. суть і особливості.

Розділ 2 Основи PHP
Тема 3 Основні поняття PHP

3.1 Що таке PHP

PHP - це широко використовувана мова сценаріїв загального призначення з відкритим вихідним кодом.

Говорячи простіше, PHP це мова програмування, спеціально розроблена для написання web-додатків (сценаріїв), що виконуються на Web-сервері.

Абревіатура PHP означає **"HypertextPreprocessor (Препроцесор Гіпертексту)"**. Синтаксис мови бере початок з C, Java і Perl. PHP досить простий для вивчення. Перевагою PHP є надання web-розробникам можливості швидкого створення динамічних web-сторінок.

Важливою **перевагою** мови PHP перед такими мовами, як мов Perl і C полягає в можливості створення HTML документів із вбудованими командами PHP.

Значною відзнакою PHP від якого-небудь коду, що виконується на стороні клієнта, наприклад, JavaScript, є те, що PHP-скрипти виконуються на стороні сервера. Ви навіть можете конфігурувати свій сервер таким чином, щоб HTML-файли оброблялися процесором PHP, так що клієнти навіть не зможуть дізнатися, чи отримують вони звичайний HTML-файл або результат виконання скрипта.

PHP дозволяє створювати якісні Web-додатки за дуже короткі терміни, отримуючи продукти, що легко модифікуються і підтримуються в майбутньому.

PHP простий для освоєння, і разом з тим здатний задовольнити запити професійних програмістів.

Мова PHP постійно удосконалюється, і їй, напевно забезпечене довге домінування в області мов web-програмування, принаймні, найближчим часом.

3.2 Можливості PHP

PHP може все. **Головна область застосування** PHP - це написання скриптів, що працюють на стороні сервера; таким чином, PHP здатний виконувати все те, що виконує будь-яка

інша програма CGI, наприклад, обробляти дані форм, генерувати динамічні сторінки або відсилати й приймати cookies. Але PHP здатний виконувати й багато інших завдань.

Існують три основні області застосування PHP.

- *Створення скриптів для виконання на стороні сервера.*

PHP найбільш широко використовується саме таким чином. Все, що вам знадобиться, це інтерпретатор PHP (у вигляді програми CGI або серверного модуля), веб-сервер і браузер. Для того щоб можна було переглядати результати виконання PHP-скриптів в браузері, потрібен працюючий веб-сервер і встановлений PHP. У випадку, якщо ви просто експериментуєте, ви цілком можете використовувати свій домашній комп'ютер замість сервера.

- *Створення скриптів для виконання в командному рядку.*

Ви можете створити PHP-скрипт, здатний запускатися незалежно від веб-сервера та браузера. Все, що вам буде потрібно - **парсер** PHP. Такий спосіб використання PHP ідеально підходить для скриптів, які повинні виконуватися регулярно, наприклад, за допомогою cron (на платформах * nix або Linux) або за допомогою планувальника завдань (TaskScheduler) на платформах Windows. Ці скрипти також можуть бути використані в задачах простої обробки текстів.

- *Створення віконних додатків, що виконуються на стороні клієнта.* Можливо, PHP є не найкращою мовою для створення подібних додатків, але, якщо ви дуже добре знаєте PHP і хотіли б використати деякі його можливості у своїх клієнт-додатках, ви можете використовувати PHP-GTK для створення таких додатків. Подібним чином ви можете створювати і крос-платформні додатки. PHP-GTK є розширенням PHP і не поставляється разом з дистрибутивом PHP.

PHP доступний для більшості операційних систем, включаючи **Linux**, багато модифікації **Unix** (такі, як HP-UX, Solaris і OpenBSD), Microsoft Windows, Mac OS X, RISC OS, та багатьох інших. Також в PHP включена підтримка більшості сучасних веб-серверів, таких, як Apache, Microsoft InternetInformation Server, PersonalWeb Server, серверів Netscape

і iPlanet, сервера OreillyWebsitePro, Caudium, Xitami, OmniHTTPd та багатьох інших. Для більшості серверів PHP поставляється в якості модуля, для інших, що підтримують стандарт CGI, PHP може функціонувати як процесор CGI.

Таким чином, вибираючи PHP, ви отримуєте свободу вибору операційної системи і веб-сервера. Крім того, у вас з'являється вибір між використанням процедурного або об'єктно-орієнтованого програмування або ж їх поєднання. Багато бібліотек коду і великі програми (включаючи бібліотеку PEAR) написані тільки з використанням ООП.

3.3 Інструментарій

Мінімальна програма

Традиційно, знайомство з мовою програмування починають з горезвісної програми "Hello, World!". Що ж, ми не будемо відступати від цієї традиції, і напишемо нашу першу програму на PHP!

Отже, беремо редактор PHP-коду, і напишемо наступний PHP код:

```
<? Php
echo "Hello, World!";
?>
```

Перш, ніж запустити програму, її потрібно встановити на сервері. Для цього збережіть написаний PHP-скрипт під назвою **start.php**. Потім скопіюйте його в каталог **DocumentRoot** вашого сервера. За умовчанням, в Linux таким каталогом є **/ var / www / html** (в старих версіях Linux - / home / httpd / html /). У Windows розташування каталогу залежить від типу встановленого web-сервера і його налаштувань. Тепер наберіть в адресному рядку вашого браузера **http://localhost/start.php** і, якщо все встановлено і налаштовано правильно, ви побачите текст Hello, World!

3.4 Синтаксис

Ми приступаємо до вивчення основних елементів синтаксису мови PHP. Розглянемо способи поділу інструкцій і створення коментарів, змінні, константи, типи даних і оператори.

Основний синтаксис

Перше, що потрібно знати щодо синтаксису PHP, - це те, як він вбудовується в HTML-код, як інтерпретатор дізнається, що це код на мові PHP. В прикладах ми найчастіше будемо використовувати варіант <? Php?>, І іноді скорочений варіант <? ?>.

Поділ інструкцій

Програма на **PHP** (та й на будь-якій іншій мові програмування) - це набір команд (інструкцій). Оброблювачу програми (парсер) необхідно якось відрізняти одну команду від іншої. Для цього використовуються спеціальні символи - роздільники. У PHP інструкції поділяються так само, як і у Сі або Perl, - кожен вираз закінчується крапкою з комою.

Закриваючий тег «?>» Також має на увазі кінець інструкції, тому перед ним крапку з комою не ставлять. Наприклад, два наступних фрагмента коду еквівалентні:

```
<? Php
echo "Hello, world!"; / / крапка з комою
/ / В кінці команди
/ / Обов'язкове
?>
<? Php
echo "Hello, world!" ?>
<! - Крапка з комою
опускається з-за "?>" ->
```

Коментарі

Часто при написанні програм виникає необхідність робити будь-які коментарі до коду, які ніяк не впливають на сам код, а тільки пояснюють його. Це важливо при створенні великих програм і у випадку, якщо кілька людей працюють над однією

програмою. При наявності коментарів у програмі в її коді розібратися набагато простіше. Крім того, якщо вирішувати задачу по частинах, недороблені частини рішення також зручно коментувати, щоб не забути про них надалі. В усіх мовах програмування передбачена можливість включати коментарі в код програми. PHP підтримує кілька видів коментарів: у стилі Сі, С + + і оболонки Unix. Символи / / і # позначають початок однорядкових коментарів, / * і * / - відповідно початок і кінець багаторядкових коментарів.

Приклад . Використання коментарів в PHP

```
<? Php
echo "Мене звуть Вася";
/ / Це однорядковий коментар
/ / У стилі С + +
echo "Прізвище моя Петров";
/ * Це багаторядковий коментар.
Тут можна написати кілька рядків.
При виконанні програми все, що
знаходиться тут (закоментоване),
буде проігноровано. * /
echo "Я вивчаю PHP";
# Це коментар в стилі
# Оболонки Unix
?>
```

3.5 Змінні, константи й оператори

Важливим елементом кожної мови є змінні, константи й оператори, застосовувані до цих змінним і констант. Розглянемо, як виділяються і обробляються ці елементи в PHP.

Змінні

Змінна у PHP позначається знаком долара, за яким слідує її ім'я. Наприклад: $ My_var

Ім'я змінної чутливо до регістру, тобто змінні **$ my_var** і **$ My_var різні**.

Імена змінних відповідають тим же правилам, що й інші найменування в PHP: правильне ім'я змінної має починатися з

букви або символу підкреслення з подальшим в будь-якій кількості літерами, цифрами або символами підкреслення.

У PHP 3 змінні завжди присвоювалися за значенням. Тобто коли ви привласнюєте вираз змінної, всі значення оригінальному вираження копіюються в цю змінну. Це означає, наприклад, що після присвоєння однієї змінної значення іншої, зміна однієї з них не впливає на значення іншої.

```
<? Php
$ First = 'Text'; / / Надаємо $ first
/ / Значення
/ / 'Text'
$ Second = $ First; / / Надаємо $ second
/ / Значення
/ / Змінної $ first
$ First = 'Newtext'; / / Змінюємо
/ / Значення
/ / $ First
/ / На 'Newtext'
echo "Змінна з ім'ям first".
"Дорівнює $ first ";
/ / Виводимо значення $ first
echo "Змінна з ім'ям second".
"Дорівнює $ second";
/ / Виводимо значення $ second
?>
```

Приклад. Присвоєння за значенням

Результат роботи цього скрипта буде наступним:

Змінна з ім'ям **first** дорівнює **Newtext**

Змінна з ім'ям **second** дорівнює **Text**

3.6 Константи

Для зберігання постійних величин, тобто таких величин, значення яких не змінюється в ході виконання скрипта, використовуються константи. Такими величинами можуть бути математичні константи, паролі, шляхи до файлів і т.п. Основна відмінність константи від змінної полягає в тому, що їй не

можна присвоїти значення більше одного разу і її значення не можна анулювати після її оголошення. Крім того, у константи немає приставки у вигляді знаку долара і її не можна визначити простим привласненням значення. Як же тоді можна визначити константу? Для цього існує спеціальна функція define (). Її синтаксис такий:

define ("Імя_константи",
"Значення_константи",
[Нечутливість _до_регістру])

За замовчуванням імена констант чутливі до регістру. Для кожної константи це можна змінити, вказавши в якості значення аргументу Нечутливість _до_регістру значення True. Існує угода, за яким імена констант завжди пишуться у верхньому регістрі.

Отримати значення константи можна, вказавши її ім'я. На відміну від змінних, не потрібно випереджати ім'я константи символом $. Крім того, для отримання значення константи можна використовувати функцію constant () з ім'ям константи в якості параметра.

Приклад. Константи в PHP

```
<? Php
/ / Визначаємо константу
/ / PASSWORD
define ("PASSWORD", "qwerty");
/ / Визначаємо регістро незалежну
/ / Константу PI зі значенням 3.14
define ("PI", "3.14", True);
/ / Виведемо значення константи PASSWORD,
/ / Тобто qwerty
echo (PASSWORD);
/ / Теж виведе qwerty
echoconstant ("PASSWORD");
echo (password);
/ * Виведе password і попередження,
оскільки ми ввели регістру
константу PASSWORD * /
```

echopi;
/ / Виведе 3.14, оскільки константа PI
/ / Регістронезалежна за визначенням
?>

Крім змінних, які декларуються користувачем, про які ми тільки що розповіли, в PHP існує ряд констант, що визначаються самим інтерпретатором. Наприклад, константа **_FILE_** зберігає ім'я файлу програми (і шлях до нього), яка виконується в даний момент, **_FUNCTION_** містить ім'я функції, **_CLASS_** - ім'я класу, **PHP_VERSION** - версія інтерпретатора PHP. Повний список зумовлених констант можна отримати, прочитавши посібник з PHP.

3.7 Оператори

Оператори дозволяють виконувати різні дії з змінними, константами і виразами. Вираз можна визначити як все, що завгодно, що має значення. Змінні і константи - це основні і найбільш прості форми виразів. Існує безліч операцій (і відповідних їм операторів), які можна робити з виразами. Розглянемо деякі з них докладніше.

Таблиця 3.1. Арифметичні оператори

Позначення	Назва	Приклад
+	Додавання	$a + $b
-	Віднімання	$a - $b
*	Множення	$a * $b
/	Ділення	$a / $b
%	Залишок від ділення	$a % $b

Таблиця 3.2. Строкові оператори

Позначення	Назва	Приклад
.	Конкатенація (додавання рядків)	$c = $a . $b (це рядок, що складається з $ a і $ b)

Таблиця 3.3. Оператори присвоювання

Позначення	Назва	Опис	Приклад
=	Присвоєння	Змінній ліворуч від оператора буде привласнене значення, отримане в результаті виконання яких-небудь операцій або змінній/константи із правої сторони	$a = ($b = 4) +5; ($a буде дорівнює 9, $b буде дорівнює 4)
+=	Скорочення	Додає до змінного число й потім привласнює їй отримане значення	$a += 5; (еквівалентно $a = $a + 5;)
.=	Назва	Скорочено позначає комбінацію операцій конкатенації й присвоювання (спочатку додається рядок, потім отриманий рядок записується в змінну)	$b = "Привіт "; $b .= "всім"; (еквівалентно $b = $b . "всім";) У результаті: $b="Привіт всім"

Таблиця 3.4. Логічні оператори

Позначення	Назва	Опис	Приклад
And	І	$a і $b щирі (True)	$a and $b
&&	І		$a && $b
Or	Або	Хоча б одна зі змінних $a або $b істина (можливо, що й обидві)	$a or $b

||	Або		$a || $b
Xor	Що виключає або	Одна зі змінних істина. Випадок, коли вони обидві істині, виключається	$a xor $b
!	Інверсія (NOT)	Якщо $a=True, то !$a=False і навпаки	! $a

Таблиця 3.5. Оператори порівняння

Позначення	Назва	Опис	Приклад
==	Рівність	Значення змінних рівні	$a == $b
===	Еквівалентність	Рівні значення й типи змінних	$a === $b
!=	Нерівність	Значення змінних не рівні	$a != $b
<>	Нерівність		$a <> $b
!==	Нееквівалентність	Змінні не еквівалентні	$a !== $b
<	Менше		$a < $b
>	Більше		$a > $b
<=	Менше або дорівнює		$a <= $b
>=	Більше або дорівнює		$a >= $b

Таблиця 3.6. Оператори інкремента й декремента

Позначення	Назва	Опис
++$a	Пре-Інкремент	Збільшує $a на одиницю й повертає $a
$a++	Пост-	Повертає $a, потім збільшує $a на

	інкремент	одиницю
--$a	Пре-декремент	Зменшує $a на одиницю й повертає $a
$ a-a--	Пост-декремент	Повертає $a, потім зменшує $a на одиницю

3.8 Типи даних

PHP підтримує вісім простих типів даних.

Чотири скалярних типи:

- boolean (двійкові дані)
- integer (цілі числа)
- float (число з плаваючою крапкою або 'double')
- string (рядки)

Два змішаних типу:

- array (масиви)
- object (об'єкти)

І два спеціальних типи:

- resource (ресурси)
- NULL (порожній тип)

Існують також кілька **псевдотипів:**

- mixed (змішаний тип)
- number (числа)
- callback (зворотного виклику)

У PHP не прийнято явне оголошення типів змінних. Переважно, щоб це робив сам інтерпретатор під час виконання програми в залежності від контексту, в якому використовується змінна. Розглянемо по порядку всі перераховані типи даних.

3.8.1 Тип boolean (логічний тип)

Цей найпростіший тип, що висловлює істинність значення, тобто змінна цього типу може мати лише два значення - істина TRUE або брехня FALSE.

Щоб визначити логічний тип, використовують ключове слово TRUE або FALSE. Обидва регістронезалежні.

Приклад. Логічний тип

```
<? Php
$ Test = True;
?>
```

Логічні змінні використовуються в різних управляючих конструкціях (циклах, умовах тощо). Мати логічний тип, тобто приймати тільки два значення, чи істину, чи брехню, можуть також і деякі оператори (наприклад, оператор рівності). Вони також використовуються в керуючих конструкціях для перевірки будь-яких умов. Наприклад, в умовній конструкції перевіряється істинність значення оператора або змінної і залежно від результату перевірки виконуються ті чи інші дії. Тут умова може бути істинно або хибно, що якраз і відображає змінна і оператор логічного типу.

Приклад . Використання логічного типу

```
<? Php
/ / Оператор '= =' перевіряє рівність
/ / І повертає
/ / Булеве значення
if ($ know = = False) {/ / якщо $ know
/ / Має значення
/ / False
echo "Вивчай PHP!";
}
if (! $ know) {/ / те ж саме, що
/ / І вище, тобто перевірка
/ / Чи має $ know значення
/ / False
echo "Вивчай PHP!";
}
/ * Оператор = = перевіряє, чи збігається
значення змінної $ know з рядком
"Вивчити PHP". Якщо співпадає, то
повертає true, інакше - false.
```

```
Якщо повернуто true, то виконується
те, що всередині фігурних дужок * /
if ($ know = = "Вивчити PHP")
{Echo "Почав вивчати";}
?>
```

3.8.2 Тип integer (цілі)

Цей тип задає число з безлічі цілих чисел Z = {..., -2, -1, 0, 1, 2, ...}. Цілі можуть бути вказані у десятковій, шістнадцятковій або вісімковій системі числення, за бажанням з попереднім знаком «-» або «+».

Якщо ви використовуєте вісімкову систему числення, ви повинні перед числом ставити 0 (нуль), для використання шістнадцяткової системи потрібно поставити перед числом 0x.

```
<? Php
# Десяткове число
$ A = 1234;
# Від'ємне число
$ A = -123;
# Вісімкове число (еквівалентно
# 83 у десятковій системі)
$ A = 0123;
# Шістнадцяткове число (еквівалентно
# 26 у десятковій системі)
$ A = 0x1A;
?>
```

Розмір цілого залежить від платформи, хоча, як правило, максимальне значення близько двох мільярдів (це 32-бітове знакова). Беззнакові цілі PHP не підтримує.

Якщо ви визначите число, що перевищує межі цілого типу, воно буде інтерпретовано як число з плаваючою крапкою. Також якщо ви використовуєте оператор, результатом роботи якого буде число, що перевищує межі цілого, замість нього буде повернуто число з плаваючою крапкою.

У PHP не існує оператора ділення цілих. Результатом 1/2 буде число з плаваючою крапкою 0.5. Ви можете навести

значення до цілого, що завжди округлює його в меншу сторону, або використовувати функцію round (), округлюються значення за стандартними правилами. Для перетворення змінної до конкретного типу потрібно перед змінною вказати в дужках потрібний тип. Наприклад, для перетворення змінної $ a = 0.5 до цілого типу необхідно написати (integer) (0.5) або (integer) $ a або використовувати скорочений запис (int) (0.5). Можливість явного приведення типів за таким принципом існує для всіх типів даних (звичайно, не завжди значення одного типу можна перевести в інший тип). Ми не будемо заглиблюватися у всі тонкощі приведення типів, оскільки PHP робить це автоматично залежно від контексту.

3.8.3 Тип float (числа з плаваючою крапкою)

Числа з плаваючою крапкою (вони ж числа подвійної точності або дійсні числа) можуть бути визначені за допомогою будь-якого з наступних синтаксисів:

```
<? Php
$ A = 1.234;
$ B = 1.2e3;
$ C = 7E-10;
?>
```

Розмір числа з плаваючою крапкою залежить від платформи, хоча максимум, як правило, ~ 1.8e308 з точністю близько 14 десяткових цифр.

3.8.4 Тип string (рядки)

Рядок - це набір символів. У PHP символ - це те ж саме, що байт, це означає, що існує рівно 256 різних символів. Це також означає, що PHP не має вбудованої підтримки Unicode. У PHP практично не існує обмежень на розмір рядків, тому немає абсолютно ніяких причин турбуватися про їх довжину.

Рядок у PHP може бути визначений **трьома різними** способами:

- за допомогою одинарних лапок;
- за допомогою подвійних лапок;

- heredoc-синтаксисом.

Одинарні лапки

Найпростіший спосіб **визначити рядок** - це укласти його в одинарні лапки «'». Щоб використовувати одинарні лапки всередині рядка, як і в багатьох інших мовах, перед нею необхідно поставити символ зворотної косої межі «\», тобто екранувати її. Якщо зворотній слеш повинен йти перед одинарними лапками або бути в кінці рядка, необхідно продублювати його «\ \ '».

Якщо всередині рядка, укладеного в одинарні лапки, зворотній слеш «\» зустрічається перед будь-яким іншим символом (відмінним від «\» і «'»), то він розглядається як звичайний символ і виводиться, як і всі інші. Тому зворотну косу риску необхідно екранувати, тільки якщо вона знаходиться в кінці рядка, перед останньою лапками.

У PHP існує ряд комбінацій символів, які починаються з символу зворотнього косого слеша. Їх називають управляючими послідовностями, і вони мають спеціальні значення, про які ми розповімо трохи пізніше. Так от, на відміну від двох інших синтаксисів, змінні і керуючі послідовності для спеціальних символів, що зустрічаються в рядках, взятих в одинарні лапки, не обробляються.

Приклад . Використання керуючих послідовностей

```
<? Php
echo 'Також ви можете додавати до рядка
символ нового рядка таким чином,
бо це нормально ';
/ / Виведе: Щоб вивести 'треба
/ / Перед нею поставити \
echo 'Щоб вивести \' треба перед '.
'Нею поставити \ \';
/ / Виведе: Ви хочете видалити C: \ *.*?
echo 'Ви хочете видалити C: \ \ *.*?';
/ / Виведе: Це не вставить: \ n новий рядок
```

```
echo 'Це не вставить: \ n новий рядок';
/ / Виведе: Змінні $ expand також
/ / $ Either не підставляються
echo 'Змінні $ expand також $ either'.
'Не підставляються';
?>
```

Подвійні лапки

Якщо рядок помістити у подвійні лапки «" », то PHP розпізнає більшу кількість керуючих послідовностей для спеціальних символів. Деякі з них наведені в таблиці 7.

Послідовність	Значення
\n	Новий рядок (LF або 0x0A (10) в ASCII)
\r	Повернення каретки (CR або 0x0D (13) в ASCII)
\t	Горизонтальна табуляція (HT або 0x09 (9) в ASCII)
\\	Зворотній слеш
\$	Знак долара
\"	Подвійна лапки

Найважливішою властивістю рядків у подвійних лапках є обробка змінних.

Heredoc

Інший спосіб визначення рядків - це використання **heredoc-синтаксису**. У цьому випадку рядок повинен починатися з символу <<<, після якого йде ідентифікатор. Закінчується рядок цим самим ідентифікатором. Закриваючий ідентифікатор повинен починатися в першому стовпці рядка. Крім того, ідентифікатор повинен відповідати тим же правилам іменування, що і всі інші позначки в PHP: містити тільки буквено-цифрові символи і знак підкреслення і починатися з цифри або знака підкреслення.

Heredoc-текст веде себе так само, як і рядок в подвійних лапках, при цьому їх не маючи. Це означає, що вам немає необхідності екранувати лапки в heredoc, але ви як і раніше можете використовувати перераховані вище керуючі послідовності. Змінні всередині heredoc теж обробляються.

Приклад . Використання heredoc-синтаксису

```
<? Php
$ Str = <<
Приклад рядка, що охоплює кілька
рядків, з використанням
heredoc-синтаксису
EOD;
/ / Тут ідентифікатор - EOD. Нижче
/ / Ідентифікатор EOD
$ Name = 'Вася';
echo<<
Мене звуть "$ name".
EOD;
/ / Це виведе "Мене звуть" Вася ".
?>
```

3.8.5 Тип array (масив)

Масиви (arrays) - це впорядковані набори даних, що представляють собою список однотипних елементів.

Існує два типи масивів, що розрізняються за способом ідентифікації елементів.

1. У масивах першого типу елемент визначається індексом у послідовності. Такі масиви називаються простими масивами.
2. Масиви другого типу мають асоціативну природу, і для звернення до елементів використовуються ключі, логічно пов'язані зі значеннями. Такі масиви називають асоціативними масивами.

Важливою особливістю PHP є те, що PHP, на відміну від інших мов, дозволяє створювати масиви будь-якої складності безпосередньо в тілі програми (скрипта).

Масиви можуть бути як одновимірними, так і багатовимірними.

Прості масиви та списки в PHP

При зверненні до елементів простих індексованих масивів використовується цілочисельний індекс, що визначає позицію заданого елемента.

Прості одномірні масиви:

Узагальнений синтаксис елементів простого одновимірного масиву:

$ Ім'я [індекс];

Масиви, індексами яких є числа, які починаються з нуля - це **списки**:

```
<? Php
/ / Простий спосіб ініціалізації масиву
$ Names [0] = "Апельсин";
$ Names [1] = "Банан";
$ Names [2] = "Груша";
$ Names [3] = "Помідор";
/ / Тут: names - ім'я масиву, а 0, 1, 2, 3 - індекси масиву
?>
```

Доступ до елементів простих масивів (списків) здійснюється наступним чином:

```
<? Php
/ / Простий спосіб ініціалізації масиву
$ Names [0] = "Апельсин";
$ Names [1] = "Банан";
$ Names [2] = "Груша";
$ Names [3] = "Помідор";
/ / Тут: names - ім'я масиву, а 0, 1, 2, 3 - індекси масиву

/ / Виводимо елементи масивів в браузер:
```

```
echo $ names [0]; / / Висновок елемента масиву names з індексом 0
echo "
";
echo $ names [3]; / / Висновок елемента масиву names з індексом 3
/ / Виводить:
/ / Апельсин
/ / Помідор
?>
```

З технічної точки зору різниці між простими масивами і списками немає.

Прості масиви можна створювати, не вказуючи індекс нового елемента масиву, це за вас зробить PHP. Ось приклад:

```
<? Php
/ / Простий спосіб ініціалізації масиву, без вказівки індексів
$ Names [] = "Апельсин";
$ Names [] = "Банан";
$ Names [] = "Груша";
$ Names [] = "Помідор";
/ / PHP автоматично присвоїть індекси елементів масиву, починаючи з 0

/ / Виводимо елементи масивів в браузер:
echo $ names [0]; / / Висновок елемента масиву names з індексом 0
echo "
";
echo $ names [3]; / / Висновок елемента масиву names з індексом 3
/ / Виводить:
/ / Апельсин
/ / Помідор
?>
```

У розглянутому прикладі ви можете додавати елементи масиву names простим способом, тобто не вказуючи індекс елемента масиву:

$ Names [] = "Яблуко";

Новий елемент простого масиву (списку) буде додано в кінець масиву. Надалі, з кожним новим елементом масиву, індекс буде збільшуватися на одиницю.

Прості багатовимірні масиви:

Узагальнений синтаксис елементів багатовимірного простого масиву:

$ Ім'я [індекс1] [індекс2] .. [індексN];

Приклад простого багатовимірного масиву:

```
<? Php
/ / Багатомірний простий масив:
$ Arr [0] [0] = "Овочі";
$ Arr [0] [1] = "Фрукти";
$ Arr [1] [0] = "Абрикос";
$ Arr [1] [1] = "Апельсин";
$ Arr [1] [2] = "Банан";
$ Arr [2] [0] = "Огірок";
$ Arr [2] [1] = "Помідор";
$ Arr [2] [2] = "Гарбуз";

/ / Виводимо елементи масиву:
echo "<h3>". $ arr [0] [0 ].":</ h3> ";
for ($ q = 0; $ q <= 2; $ q + +) {
echo $ arr [2] [$ q]. "
";
}
echo "<h3>". $ arr [0] [1 ].":</ h3> ";
for ($ w = 0; $ w <= 2; $ w + +) {
echo $ arr [1] [$ w]. "<br>";
}
?>
```

Асоціативні масиви в PHP

У PHP індексом масиву може бути не тільки число, але і рядок. Причому на такий рядок не накладаються ніякі обмеження: вони можуть містити пробіли, довжина такого рядка може бути будь-яка.

Асоціативні масиви особливо зручні в ситуаціях, коли елементи масиву зручніше пов'язувати зі словами, а не з числами.

Отже, масиви, індексами яких є рядки, називаються асоціативними масивами.

Одномірні асоціативні масиви:

Одномірні асоціативні масиви містять тільки один ключ (елемент), відповідний конкретному індексу асоціативного масиву. Наведемо **приклад:**

```
<? Php
/ / Асоціативний масив
$ Names ["Іванов"] = "Іван";
$ Names ["Сидоров"] = "Микола";
$ Names ["Петров"] = "Петро";
/ / У даному прикладі: прізвища - ключі асоціативного масиву
/ /, А імена - елементи масиву names
?>
```

Доступ до елементів одновимірних асоціативних масивів здійснюється так само, як і до елементів звичайних масивів, і називається доступом по ключу:

```
echo $ names ["Іванов"];
```

Багатовимірні асоціативні масиви

Багатовимірні асоціативні масиви можуть містити кілька ключів, які відповідають конкретним індексам асоціативного масиву. Розглянемо приклад багатовимірного асоціативного масиву:

```
<? Php
/ / Багатомірний масив
```

$ A ["Ivanov"] = array ("name" => "Іванов І.І.", "age" => "25", "email" => "ivanov@ukr.net");

$ A ["Petrov"] = array ("name" => "Петров П.П.", "age" => "34", "email" => "petrov@ukr.net");

$ A ["Sidorov"] = array ("name" => "Сидоров С.С.", "age" => "47", "email" => "sidorov@ukr.net");

?>

Багатовимірні масиви схожі на записи у мові Pascal або структури в мові С.

Доступ до елементів багатовимірного асоціативного масиву здійснюється наступним чином:

echo $ A ["Ivanov"] ["name"]; / / Виводить Іванов І.І.

echo $ A ["Petrov"] ["email"]; / / Виводить petrov@ukr.net

Як ви вже помітили, для створення багатовимірного асоціативного масиву ми використовували спеціальну функцію array, ми її розглянемо пізніше, коли будемо розглядати операції над масивами.

Асоціативні багатовимірні масиви можна створювати і класичним способом, хоча це не так зручно:

<? Php

/ / Багатомірний асоціативний масив

$ A ["Ivanov"] ["name"] = "Іванов І.І.";

$ A ["Ivanov"] ["age"] = "25";

$ A ["Ivanov"] ["email"] = "ivanov@ukr.net";

$ A ["Petrov"] ["name"] = "Петров П.П.";

$ A ["Petrov"] ["age"] = "34";

$ A ["Petrov"] ["email"] = "petrov@ukr.net";

$ A ["Sidorov"] ["name"] = "Сидоров С.С.";

$ A ["Sidorov"] ["age"] = "47";

$ A ["Sidorov"] ["email"] = "sidorov@ukr.net";

/ / Отримуємо доступ до ключів багатовимірного асоціативного масиву

echo $ A ["Ivanov"] ["name "]."
"; / / Виводить Іванов І.І.

```
echo $ A ["Sidorov"] ["age "]."<br>"; / / Виводить 47
echo $ A ["Petrov"] ["email "]."<br>"; / / Виводить petrov@ukr.net
?>
```

3.8.6 Тип object (об'єкти)

Об'єкти - тип даних, що прийшов з об'єктно-орієнтованого програмування (ООП). Згідно з принципами ООП, клас - це набір об'єктів, що володіють певними властивостями і методами роботи з ним, а об'єкт відповідно - екземпляр класу. Наприклад, програмісти - це клас людей, які пишуть програми, вивчають комп'ютерну літературу і, крім того, як всі люди, мають ім'я та прізвище. Тепер, якщо взяти одного конкретного програміста, Васю Іванова, то можна сказати, що він є об'єктом класу програмістів, має ті ж властивості, що й інші програмісти, теж має ім'я, пише програми і т.п.

У PHP для доступу до методів об'єкта використовується оператор ->. Для ініціалізації об'єкту використовується вираз new, що створює в змінної екземпляр об'єкта.

```
<? Php
/ / Створюємо клас людей
classPerson
{
/ / Метод, який навчає людину PHP
functionknow_php ()
{
echo "Тепер я знаю PHP";
}
}
$ Bob = newPerson; / / створюємо об'єкт
/ / Класу людина
$ Bob ->know_php (); / / навчаємо його PHP
?>
```

3.8.7 Тип resource (ресурси)

Ресурс - це спеціальна змінна, що містить посилання на зовнішній ресурс (наприклад, з'єднання з базою даних). Ресурси створюються та використовуються спеціальними функціями (наприклад, mysql_connect (), pdf_new () і т.п.).

3.8.8 Тип Null

Спеціальне значення NULL говорить про те, що змінна не має значення.

Змінна вважається NULL, якщо:

- їй була присвоєна константа NULL ($ var = NULL);
- їй ще не було присвоєно будь-яке значення;
- вона була вилучена за допомогою unset ().

Існує тільки одне значення типу NULL - регістронезалежне ключове слово NULL.

Тема 4 Основи клієнт-серверних технологій

4.1 Зміст (поняття) клієнт-серверних технологій

PHP - це скриптова мова, оброблювана сервером. Якщо мова йде про сервер, мимоволі спливає в пам'яті поняття клієнта. Все тому, що ці два поняття нерозривно пов'язані. Об'єднує їх комп'ютерна архітектура клієнт-сервер. Зазвичай, коли говорять «сервер», мають на увазі сервер в архітектурі клієнт-сервер, а коли говорять «клієнт» - мають на увазі клієнт в цій же архітектурі. Так що ж це за архітектура? Суть її в тому, щоб розділити функції між двома підсистемами: клієнтом, який відправляє запит на виконання будь-яких дій, і сервером, який виконує цей запит. Взаємодія між клієнтом і сервером відбувається за допомогою стандартних спеціальних протоколів, таких як TCP / IP і z39.50. Насправді протоколів дуже багато, вони розрізняються за рівнями. Ми розглянемо тільки протокол прикладного рівня HTTP (трохи пізніше), оскільки для вирішення наших програмістських завдань потрібен тільки він. А поки повернемося до клієнт-серверної архітектури і розберемося, що ж таке клієнт і що таке сервер.

Сервер являє собою набір програм, які контролюють виконання різних процесів. Відповідно, цей набір програм встановлений на якомусь комп'ютері. Часто комп'ютер, на якому встановлено сервер, і називають сервером. Основна функція комп'ютера-сервера - по запиту клієнта запустити який-небудь певний процес і відправити клієнту результати його роботи.

Клієнтом називають будь-який процес, який користується послугами сервера. Клієнтом може бути як користувач, так і програма. Основне завдання клієнта - виконання програми та здійснення зв'язку з сервером, коли цього потребує програма. Тобто клієнт повинен надавати користувачеві інтерфейс для роботи з додатком, реалізовувати логіку його роботи і при необхідності відправляти завдання серверу.

Взаємодія між клієнтом і сервером починається з ініціативи клієнта. Клієнт запитує вид обслуговування, встановлює сеанс,

отримує потрібні йому результати і повідомляє про закінчення роботи.

Послугами одного сервера найчастіше користується декілька клієнтів одночасно. Тому кожен сервер повинен мати досить велику продуктивність і забезпечувати безпеку даних.

Логічно встановлювати сервер на комп'ютері, що входить в яку-небудь мережу, локальну або глобальну. Однак можна встановлювати сервер і на окремому комп'ютері (тоді він буде одночасно і клієнтом і сервером).

Існує безліч типів серверів. Ось лише деякі з них.

- **Відеосервер** Такий сервер спеціально пристосований до обробки зображень, зберігання відеоматеріалів, відеоігор і т.п. У зв'язку з цим комп'ютер, на якому встановлений відеосервер, повинен мати високу продуктивність і велику пам'ять.
- **Пошуковий сервер** призначений для пошуку інформації в Internet.
- **Поштовий сервер** надає послуги у відповідь на запити, надіслані електронною поштою.
- **Сервер WWW** призначений для роботи в Internet.
- **Сервер баз даних** виконує обробку запитів до баз даних.
- **Сервер захисту даних** призначений для забезпечення безпеки даних (містить, наприклад, засоби для ідентифікації паролів).
- **Сервер додатків** призначений для виконання прикладних процесів. З одного боку взаємодіє з клієнтами, одержуючи завдання, а з іншого - працює з базами даних, підбираючи необхідні для обробки дані.
- **Сервер віддаленого доступу** забезпечує колективний віддалений доступ до даних.
- **Файловий сервер** забезпечує функціонування розподілених ресурсів, надає послуги пошуку, зберігання, архівування даних і можливість одночасного доступу до них декількох користувачів.

Зазвичай на комп'ютері-сервері працює відразу кілька програм-серверів. Одна займається електронною поштою, інша розподілом файлів, третя надає web-сторінки.

З усіх типів серверів нас в основному цікавить сервер WWW. Часто його називають web-сервером, http-сервером або навіть просто сервером. Що являє собою web-сервер? По-перше, це сховище інформаційних ресурсів. По-друге, ці ресурси зберігаються і надаються користувачам відповідно до стандартів Internet (такими, як протокол передачі даних HTTP). Як надаються дані у відповідності з цим протоколом, ми розглянемо трохи пізніше. Робота з документами web-серверу здійснюється за допомогою браузера (наприклад, IE, Opera або Mozilla), який відсилає серверу запити, створені відповідно до протоколу HTTP. У процесі виконання завдання сервер може зв'язуватися з іншими серверами.

В якості прикладів web-серверів можна навести сервер Apache групи Apache, InternetInformation Server (IIS) компанії Microsoft, SunOne фірми SunMicrosystems, WebLogic фірми BEA Systems, IAS (InpriseApplication Server) фірми Borland, WebSphere фірми IBM, OAS (OracleApplication Server).

Все, що ми коли-небудь будемо говорити про web-серверах, орієнтоване на Apache, якщо не вказано інший. Про те, як встановити його на свій комп'ютер, ми вже розповідали в самій першій лекції. А тепер, як було обіцяно, звернемося до протоколу HTTP.

4.2 Протокол HTTP і способи передачі даних на сервер

Internet побудований за багаторівневим принципом, від фізичного рівня, пов'язаного з фізичними аспектами передачі двійкової інформації, і до прикладного рівня, що забезпечує інтерфейс між користувачем і мережею.

HTTP (HyperTextTransferProtocol, протокол передачі гіпертексту) - це протокол прикладного рівня, розроблений для обміну гіпертекстової інформацією в Internet.

HTTP надає набір методів для вказівки цілей запиту, що відправляється серверу. Ці методи засновані на дисципліні

посилань, де для вказівки ресурсу, до якого має бути застосований даний метод, використовується універсальний ідентифікатор ресурсів (UniversalResourceIdentifier) у вигляді місцезнаходження ресурсу (UniversalResourceLocator, URL) або у вигляді його універсального імені (UniversalResourceName , URN).

Повідомлення по мережі при використанні протоколу HTTP передаються у форматі, схожому з форматом поштового повідомлення Internet (RFC-822) або з форматом повідомлень MIME (MultipurposeInternetMail Exchange).

HTTP використовується для комунікацій між різними користувацькими програмами та програмами-шлюзами, що надають доступ до існуючих Internet-протоколів, таких як SMTP (протокол електронної пошти), NNTP (протокол передачі новин), FTP (протокол передачі файлів), Gopher і WAIS. HTTP розроблений для того, щоб дозволяти таким шлюзам через проміжні програми-сервери (proxy) передавати дані без втрат.

Протокол реалізує принцип запит / відповідь. Запитуюча програма-клієнт ініціює взаємодію з відповідною програмою-сервером, і надсилає запит, який містить:

- метод доступу;
- адресу URI;
- версію протоколу;
- повідомлення (схоже за формою на MIME) з інформацією про тип переданих даних, інформацією про клієнта, що послав запит, і, можливо, із змістовною частиною (тілом) повідомлення.

Відповідь сервера містить:

- рядок стану, в яку входить версія протоколу і код повернення (успіх або помилка);
- повідомлення (у формі, схожій на MIME), до якого входить інформація сервера, метаінформація (тобто інформація про зміст повідомлення) і тіло повідомлення.

У протоколі не вказується, хто повинен відкривати і закривати з'єднання між клієнтом і сервером. На практиці

з'єднання, як правило, відкриває клієнт, а сервер після відправки відповіді ініціює його розрив.

4.3 Форма запиту клієнта

Клієнт відсилає серверу запит в одній з двох форм: у повній або скороченій. Запит у першій формі називається відповідно повним запитом, а в другій формі - простим запитом.

Простий запит містить метод доступу та адресу ресурсу. Формально це можна записати так:

<Простий-Запит>: = <Метод><символ пробіл>

<Запитуваний-URI><символ нового рядка>

В якості методу можуть бути вказані GET, POST, HEAD, PUT, DELETE та інші. Про найбільш поширені з них ми поговоримо трохи пізніше. В якості запитуваної URI найчастіше використовується URL-адресу ресурсу.

Приклад простого запиту:

GET http://phpbook.info/

Тут **GET** - це метод доступу, тобто метод, який повинен бути застосований до запитуваного ресурсу, а http://phpbook.info/ - це URL-адреса запитуваного ресурсу.

Повний запит містить рядок стану, кілька заголовків (заголовок запиту, загальний заголовок або заголовок запису) і, можливо, тіло запиту. Формально загальний вигляд повного запиту можна записати так:

<Повний запит>: = <Рядок Стану>

(<Загальний заголовок> | <Тема запиту> |

<Заголовок змісту>)

<Символ нового рядка>

[<Зміст запиту>]

Квадратні дужки тут позначають необов'язкові елементи заголовка, через вертикальну риску перераховані альтернативні варіанти. Елемент <Рядок стану> містить метод запиту та URI ресурсу (як і простий запит) і, крім того, використовувану версію протоколу HTTP. Наприклад, для виклику зовнішньої програми можна задіяти наступний рядок стану:

POST http://phpbook.info/cgi-bin/test HTTP/1.0

У даному випадку використовується метод POST і протокол HTTP версії 1.0.

В обох формах запиту важливе місце займає URI запитуваного ресурсу. Найчастіше URI використовується у вигляді URL-адреси ресурсу. При зверненні до сервера можна застосовувати як повну форму URL, так і спрощену.

Повна форма містить тип протоколу доступу, адреса сервера ресурсу та адресу ресурсу на сервері (рис 4.1).

У скороченій формі опускають протокол і адресу сервера, вказуючи лише місце розташування ресурсу від кореня сервера. Повну форму використовують, якщо можливе пересилання запиту іншого сервера. Якщо ж робота відбувається тільки з одним сервером, то частіше застосовують скорочену форму.

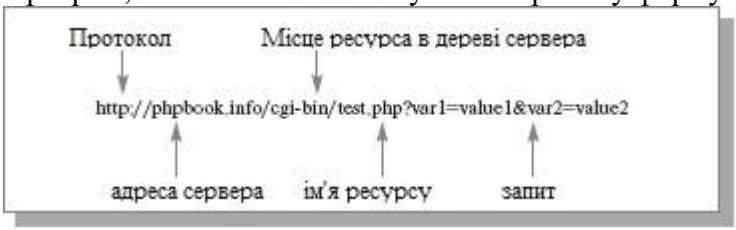


Рис. 4.1. Повна форма URL

4.4 Методи

Метод повідомляє про мету запиту клієнта. Протокол HTTP підтримує досить багато методів, але реально використовуються тільки три: POST, GET і HEAD. Метод GET дозволяє отримати будь-які дані, ідентифіковані за допомогою URI в запиті ресурсу. Якщо URI вказує на програму, то повертається результат роботи програми, а не її текст (якщо, звичайно, текст не є результат її роботи). Додаткова інформація, необхідна для обробки запиту, вбудовується в сам запит (у рядок статусу). При використанні методу GET у поле тіла ресурсу повертається власне затребувана інформація (текст HTML-документа, наприклад).

Існує різновид методу GET - **умовний GET.** Цей метод повідомляє серверу про те, що на запит потрібно відповісти,

тільки якщо виконуються умови, що міститься в полі if-Modified-Since заголовка запиту. Якщо говорити більш точно, то тіло ресурсу передається у відповідь на запит, якщо цей ресурс змінювався після дати, зазначеної в if-Modified-Since.

Метод **HEAD** аналогічний методу GET, тільки не повертає тіло ресурсу і не має умовного аналога. Метод HEAD використовують для отримання інформації про ресурс. Це може стати в нагоді, наприклад, при вирішенні завдання тестування гіпертекстових посилань.

Метод **POST** розроблений для передачі на сервер такої інформації, як анотації ресурсів, новині і поштові повідомлення, дані для додавання в базу даних, тобто для передачі інформації великого обсягу і досить важливої. На відміну від методів GET і HEAD, в POST передається тіло ресурсу, яке і є інформацією, одержуваної з полів форм або інших джерел введення.

До цих пір ми тільки теоретизували, знайомилися з основними поняттями. Тепер варто навчитися використовувати все це на практиці. Далі в лекції ми розглянемо, як посилати запити серверу і як обробляти його відповіді.

4.5 Використання HTML-форм для передачі даних на сервер

Як передавати дані серверу? Для цього в мові HTML є спеціальна конструкція - форми**.** Форми призначені для того, щоб отримувати від користувача інформацію. Наприклад, вам потрібно знати логін і пароль користувача для того, щоб визначити, на які сторінки сайту його можна допускати. Або вам необхідні особисті дані користувача, щоб була можливість з ним зв'язатися. Форми якраз і застосовуються для введення такої інформації. У них можна вводити текст або вибирати підходящі варіанти зі списку. Дані, записані у форму, відправляються для обробки спеціальною програмою (наприклад, скрипту на PHP) на сервері. Залежно від введених користувачем даних ця програма може формувати різні web-сторінки, відправляти запити до бази даних, запускати різні додатки і т.п.

Розберемося з синтаксисом HTML-форм. Можливо, багато хто з ним знайомі, але ми все ж повторимо основні моменти, оскільки це важливо.

Отже, для створення форми в мові HTML використовується тег FORM. Всередині нього знаходиться одна або декілька команд INPUT. За допомогою атрибутів action і methodтега FORM задаються ім'я програми, яка буде обробляти дані форми, і метод запиту, відповідно. Команда INPUT визначає тип і різні характеристики запитуваної інформації. Надсилання даних форми відбувається після натискання кнопки input типу submit. Створимо форму для реєстрації учасників заочної школи програмування.

```
<h2> Форма для реєстрації учасників </ h2>
<formaction="1.php" method=POST>
відправці запиту буде використаний метод POST ->
Ім'я <br><input type = text name = "first_name"
value = "Ваше ім'я"><br>
Прізвище <br><input type=textname="last_name"><br>
E-mail<br><input type=textname="email"><br>
<p>
Виберіть курс, який ви б хотіли відвідувати: <br>
<input type=radio name="kurs" value="PHP"> PHP <br>
<input type=radio name="kurs" value="Lisp">Lisp<br>
<input type=radio name="kurs" value="Perl">Perl<br>
<input type=radio name="kurs" value="Unix">Unix<br>
<P> Що ви хочете, щоб ми знали про вас? <BR>
<textarea name="comment" cols=32 rows=5>
<P><Input name = "confirm" type = checkbox
checked> Підтвердити отримання <br>
<input type=submit value="Відправити">
<input type=reset value="Відмініти">
```

Після обробки браузером цей файл буде виглядати приблизно так:

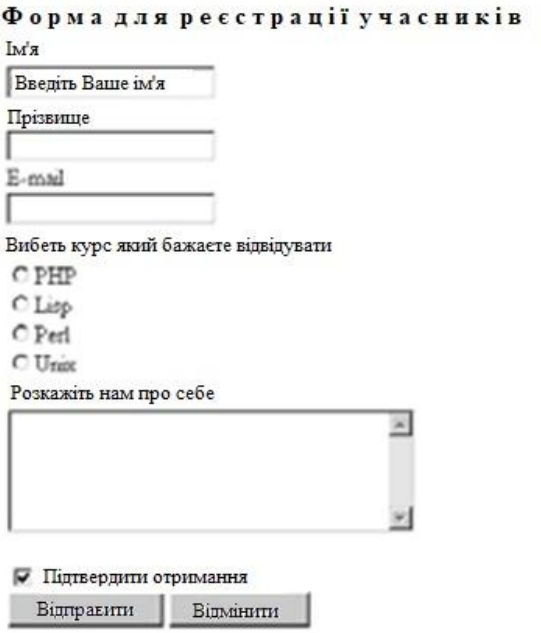


Рис. 4.2. Приклад html-форми

Ось так створюються і виглядають HTML-форми. Будемо вважати, що ми навчилися чи згадали, як їх створювати. Як ми бачимо, у формі можна вказувати метод передачі даних. Подивимося, що буде відбуватися, якщо вказати метод GET або POST, і в чому буде різниця.

4.5.1 Дія методу GET

Для методу GET При відправці даних форми за допомогою методу GET вміст форми додається до URL після знака запитання у вигляді пар імен = значення, об'єднаних за допомогою амперсанда&:

action? name1 = value1 & name2 = value2 & name3 = value3

Тут action - це URL-адрес програми, яка повинна обробляти форму (це або програма, задана в атрибуті actionтегаform, або сама поточна програма, якщо цей атрибут опущений). Імена name1, name2, name3 відповідають іменам елементів форми, а value1, value2, value3 - значення цих елементів. Всі спеціальні

символи, включаючи = і &, в іменах або значеннях цих параметрів будуть закодовані. Тому не варто використовувати в назвах або значеннях елементів форми ці символи і символи кирилиці в ідентифікаторах.

Якщо в полі для введення ввести який-небудь службовий символ, то він буде переданий в його шістнадцятковому коді, наприклад, символ $ заміниться на% 24. Так само передаються і російські літери.

Для полів введення тексту і пароля (це елементи input з атрибутом type = text і type = password), значенням буде те, що введе користувач. Якщо користувач нічого не вводить в таке поле, то в рядку запиту буде присутній елемент name =, де name відповідає імені цього елемента форми.

Для кнопок типу **checkbox** і **radiobutton** значення **value** визначається атрибутом VALUE в тому випадку, коли кнопка відзначена. Не зазначені кнопки при складанні рядка запиту ігноруються повністю. Кілька кнопок типу checkbox можуть мати один атрибут NAME (і різні VALUE), якщо це необхідно. Кнопки типу radiobutton призначені для одного з усіх запропонованих варіантів і тому повинні мати однаковий атрибут NAME і різні атрибути VALUE.

У принципі створювати HTML-форму для передачі даних методом GET не обов'язково. Можна просто додати рядок URL потрібні змінні та їх значення.

Приклад . Передача даних в URL

http://phpbook.info/test.php?id=10&user=pit

У зв'язку з цим у передачі даних методом GET є один істотний недолік - будь-хто може підробити значення параметрів. Тому не радимо використовувати цей метод для доступу до захищених паролем сторінок, для передачі інформації, що впливає на безпеку роботи програми або сервера. Крім того, не варто застосовувати метод GET для передачі інформації, яку не дозволено змінювати користувачеві.

Незважаючи на всі ці недоліки, використовувати метод GET досить зручно при налагодженні скриптів (тоді можна бачити значення й імена переданих змінних) і для передачі параметрів, які не впливають на безпеку.

4.5.2 Дія методу POST

Вміст форми кодується точно так само, як для методу **GET**, але замість додавання рядка до **URL** вміст запиту надсилається блоком даних як частина операції POST. Якщо присутній атрибут **ACTION**, то значення **URL**, яке там знаходиться, визначає, куди посилати цей блок даних. Цей метод, як уже зазначалося, рекомендується для передачі великих за обсягом блоків даних.

Інформація, введена користувачем і відправлена серверу за допомогою методу POST, подається на стандартне введення програми, зазначеної в атрибуті action, чи поточного скрипту, якщо цей атрибут опущений. Довжина посилається файлу передається у змінній оточення CONTENT_LENGTH, а тип даних - у змінній CONTENT_TYPE.

Передати дані методом POST можна тільки за допомогою HTML-форми, оскільки дані передаються в тілі запиту, а не в заголовку, як у GET. Відповідно і змінити значення параметрів можна, тільки змінивши значення, введене в форму. При використанні POST користувач не бачить чи передаються серверу дані.

Основна перевага POST запитів - це їхня велика безпека і функціональність у порівнянні з GET-запитами. Тому метод POST частіше використовують для передачі важливої інформації, а також інформації великого обсягу. Тим не менше не варто цілком покладатися на безпеку цього механізму, оскільки дані POST запиту також можна підробити, наприклад створивши html-файл на своїй машині і заповнивши його потрібними даними. Крім того, не всі клієнти можуть застосовувати метод POST, що обмежує варіанти його використання.

При відправці даних на сервер будь-яким методом передаються не тільки самі дані, введені користувачем, але і ряд перемінних, які називаються змінними середовища, що характеризують клієнта, історію його роботи, шляхи до файлів і т.п. Ось деякі з змінних оточення:

- REMOTE_ADDR - IP-адреса хоста (комп'ютера), що відправляє запит;
- REMOTE_HOST - ім'я хоста, з якого надіслано запит;
- HTTP_REFERER - адреса сторінки, що посилається на поточний скрипт;
- REQUEST_METHOD - метод, який був використаний при відправці запиту;
- QUERY_STRING - інформація, яка перебуває в URL після знака питання;
- SCRIPT_NAME - віртуальний шлях до програми, яка повинна виконуватися;
- HTTP_USER_AGENT - інформація про браузер, який використовує клієнт

4.6 Обробка запитів за допомогою PHP

До цих пір ми згадували тільки, що запити клієнта обробляються на сервері за допомогою спеціальної програми. Насправді цю програму ми можемо написати самі, в тому числі і на мові PHP, і вона буде робити з отриманими даними все, що ми захочемо. Для того, щоб написати цю програму, необхідно познайомитися з деякими правилами і інструментами, запропонованими для цих цілей PHP.

Усередині PHP-скрипта є декілька способів отримання доступу до даних, переданим клієнтом по протоколу HTTP. Починаючи з PHP 4.1.0 для звернення до змінних, переданих за допомогою HTTP-запитів, задіюють спеціальний масив - $ _REQUEST. Цей масив містить дані, передані методами POST і GET, а також за допомогою HTTP cookies. Це Суперглобальний асоціативний масив, тобто його значення можна отримати в будь-якому місці програми, використовуючи як ключ ім'я відповідної змінної (елементу форми).

Приклад . Припустимо, ми створили форму для реєстрації учасників заочної школи програмування, як у наведеному вище прикладі. Тоді у файлі 1.php, що обробляє цю форму, можна написати наступне:

```
<? Php
$ Str = "Здрастуйте,
". $ _REQUEST [" First_name "]."
". $ _REQUEST [" Last_name "]."! <br> ";
$ Str .= "Ви обрали для вивчення курс по
". $ _REQUEST [" Kurs "];
echo $ str;
?>
```

Приклад. Файл 1.php, обробний форму form.html

Тоді, якщо до форми ми ввели ім'я «Вася», прізвище «Петров» і вибрали серед усіх курсів курс по PHP, на екрані браузера отримаємо таке повідомлення:

Здравствуйте, Вася Петров!
Ви обрали для вивчення курс по PHP

Після введення масиву $ _REQUEST масиви $ HTTP_POST_VARS і $ HTTP_GET_VARS для однорідності були перейменовані в $ _POST і $ _GET відповідно, але самі вони з ужитку не зникли з міркувань сумісності з попередніми версіями PHP. На відміну від своїх попередників, масиви $ _POST і $ _GET стали суперглобальними, тобто доступними безпосередньо і всередині функцій і методів.

Наведемо приклад використання цих масивів. Припустимо, нам потрібно обробити форму, що містить елементи введення з іменами first_name, last_name, kurs (наприклад, форму form.html, наведену вище). Дані були передані методом POST, і дані, передані іншими методами, ми обробляти не хочемо. Це можна зробити наступним чином:

```
<? Php
$ Str = "Здрастуйте,
". $ _POST [" First_name "]."
```

". $ _POST [" Last_name "]."!
 ";
$ Str .= "Ви обрали для вивчення курс по".
$ _POST ["Kurs"];
echo $ str;
?>

Тоді на екрані браузера, якщо ми ввели ім'я «Вася», прізвище «Петров» і вибрали серед усіх курсів курс по PHP, побачимо повідомлення, як у попередньому прикладі:

Здравствуйте, Вася Петров!
Ви обрали для вивчення курс по PHP

Іноді виникає необхідність дізнатися значення якої-небудь змінної оточення, наприклад метод, що використовувався при передачі запиту або IP-адреса комп'ютера, що відправив запит. Отримати таку інформацію можна за допомогою функції getenv (). Вона повертає значення змінної оточення, ім'я якої передано їй як параметр.

<?
getenv ('REQUEST_METHOD');
/ / Поверне використаний метод
echogetenv ('REMOTE_ADDR');
/ / Виведе IP-адресу користувача,
/ / Послав запит
?>

Приклад . Використання функції getenv ()

Все, що записано в URL після знака запитання, можна отримати за допомогою команди

getenv ('QUERY_STRING');

Завдяки цьому можна методом GET передавати дані в якому-небудь іншому вигляді. Наприклад, вказувати тільки значення декількох параметрів через знак плюс, а в скрипті розбивати рядок запиту на частини або можна передавати значення лише одного параметра. У цьому випадку в масиві $ _GET з'явиться порожній елемент з ключем, рівним цьому значенню (всього

рядка запиту), причому символ «+», що зустрівся в рядку запиту, буде замінений на підкреслення «_».

Методом POST дані передаються тільки за допомогою форм, і користувач (клієнт) не бачить, які саме дані відправляються серверу. Щоб їх побачити, хакер повинен підмінити нашу форму на власну. Тоді сервер відправить результати обробки неправильної форми не туди, куди потрібно. Щоб цього уникнути, можна перевіряти адресу сторінки, з якої були надіслані дані. Це можна зробити знову ж за допомогою функції getenv ():

getenv ('HTTP_REFERER');

4.7 Приклад обробки запиту за допомогою PHP

Нагадаємо, в чому полягало завдання, і уточнимо її формулювання. Потрібно написати форму для реєстрації учасників заочної школи програмування і після реєстрації відправити учаснику повідомлення. Ми назвали це повідомлення універсальним листом, але воно буде трохи відрізнятися від того листа, який ми склали на попередній лекції. Тут ми також не будемо відправляти що-небудь по електронній пошті, щоб не уподібнюватися спамерам, а просто згенеруємо це повідомлення і виведемо його на екран браузера. Початковий варіант форми реєстрації ми вже наводили вище. Змінимо його таким чином, щоб кожен хто реєструється, міг вибрати скільки завгодно курсів для відвідування, і не будемо підтверджувати отримання реєстраційної форми.

```
<h2> Форма для реєстрації студентів </ h2>
<formaction="1.php" method=POST>
Ім'я <br><input type = text name = "first_name"
value = "Ваше ім'я"><br>
Прізвище <br><input type=text name="last_name"><br>
E-mail<br><input type=text name="email">lt;br>
<li><p> Виберіть курс, який ви б хотіли відвідувати: </li>br>
<li input type=checkbox name='kurs[]' value='PHP'> PHP </li>br>
<li input type=checkbox name='kurs[]' value='Lisp'>Lisp</li>br>
```

```
<li input type=checkbox name='kurs[]' value='Perl'>Perl</li>br>
<li input type=checkbox name='kurs[]' value='Unix'>Unix</li>br>
<li> <p> Що ви хочете, щоб ми знали про вас? </li>BR>
<li><textarea name="comment" cols=32 rows=5></ textarea>
<li>input type= submit value="Відправте">
<li>input type=reset value="Відмініть">
</ form>
```

Приклад. form_final.html

Тут все досить просто і зрозуміло. Єдине, що можна відзначити, - це спосіб передачі значень елемента checkbox. Коли ми пишемо в імені елемента kurs [], це означає, що перший зазначений елемент checkbox буде записаний в перший елемент масиву kurs, другий зазначений checkbox - у другий елемент масиву і т.д. Можна, звичайно, просто дати різні імена елементів checkbox, але це ускладнить обробку даних, якщо курсів буде багато.

Скрипт, який все це буде розбирати і обробляти, називається 1.php (форма посилається саме на цей файл, що записано в її атрибуті action). За замовчуванням використовується для передачі метод GET, але ми вказали POST. За отриманими даними від зареєстрованої людини, скрипт генерує відповідне повідомлення. Якщо людина вибрала якісь курси, то йому виводиться повідомлення про час їх проведення та про лекторів, які їх читають. Якщо людина нічого не вибрав, то виводиться повідомлення про наступні збори заочної школи програмістів.

```
<?
/ / Створимо масиви відповідностей курс - час його
/ / Проведення та курс - його лектор
$ Times = array ("PHP" => "14.30", "Lisp" => "12.00",
"Perl" => "15.00", "Unix" => "14.00");
$ Lectors = array ("PHP" => "Василь Васильович",
"Lisp" => "Іван Іванович", "Perl" => "Петро Петрович", "Unix" => "Семен Семенович");
define ("SIGN", "З повагою, адміністрація");
/ / Визначаємо підпис листи як константу
```

```
define ("MEETING_TIME", "18.00");
/ / Задаємо час зборів студентів
$ Date = "12 травня"; / / задаємо дату проведення лекцій
/ / Починаємо складати текст повідомлення
$ Str = "Здрастуйте, шановний". $ _POST ["First_name"]
. "". $ _POST ["Last_name "]."!<br>";
$ Str .= "<br> Повідомляємо Вам, що";
$ Kurses = $ _POST ["kurs"]; / / збережемо в цій змінній
/ / Список вибраних курсів
if (! isset ($ kurses)) {/ / якщо не обраний жоден курс
$ Event = "наступні збори студентів";
$ Str .= "$ event відбудеться $ date". MEETING_TIME. "<br>";
} Else {/ / якщо хоча б один курс вибраний
$ Event = "обрані Вами лекції відбудуться $ date<ul>";
/ / Функція count обчислює число елементів у масиві
$ Lect = "";
for ($ i = 0; $ i <count ($ kurses); $ i + +) {
/ / Для кожного обраного курсу
$ K = $ kurses [$ i]; / / запам'ятовуємо назву курсу
$ Lect = $ lect. "<li> Лекція з $ k в $ times [$ k]";
/ / Складаємо повідомлення
$ Lect .= "(Ваш лектор, $ lectors [$ k])";
}
$ Event = $ event. $ Lect. "</ Ul>";
$ Str .= "$ event";
}
$ Str .= "<br>". SIGN; / / додаємо підпис
echo $ str; / / виводимо повідомлення на екран
?>
```

Приклад. Скрипт 1.php, обробний форму form_final.html

4.8 Суперглобальні масиви

Розглянемо роботу суперглобальних змінних в PHP.

У PHP існує кілька суперглобальний змінних, а точніше суперглобальних масивів:

- $ _SERVER

- $ _GET
- $ _POST
- $ _FILES
- $ _COOKIE
- $ _SESSION
- $ _REQUEST
- $ _ENV

4.8.1 Суперглобальний масив $ _SERVER

Масив являє собою інформацію про заголовки, шляхи та розміщення скриптів. Записи в цьому масиві створюються веб-сервером. Не існує гарантій, що веб-сервер сформує цей масив з усіма параметрами. Даний масив містить такі елементи:

PHP_SELF: ім'я файлу в даний час виконується PHP-скриптом. Наприклад при виконанні скрипта http://phpprogs.ru/test/guestbook2/ даний елемент буде приймати значення / test/guestbook2/index.php.

argv: список аргументів, переданих скрипту. Якщо використовує в командному рядку, то отримуєте масив значень, якщо використовується $ _GET, то буде містити рядок запиту.

argc: містить число параметрів переданих сценарієм (якщо запуск був з командного рядка).

GATEWAY_INTERFACE: параметр повертає версію CGI, яку використовує веб-сервер.

SERVER_ADDR: елемент містить IP адреса сервера, де виконується скрипт.

SERVER_NAME: елемент містить ім'я веб-сервера, де виконується скрипт.

SERVER_SOFTWARE: ідентифікаційний рядок веб-сервера, який повертається у відповідь при запитах.

SERVER_PROTOCOL: ім'я та версія використовуваного протоколу HTTP.

REQUEST_METHOD: використовуваний метод запиту до веб-сервера (POST, GET, HEAD, PUT).

REQUEST_TIME: відмітка про час початку запиту (починаючи з PHP 5.1.0).

QUERY_STRING: рядок запит до веб-сторінки, якщо вона існує, за допомогою якого був здійснений доступ до сторінки

DOCUMENT_ROOT: коренева директорія, з якої виконується скрипт.

HTTP_ACCEPT: зміст заголовка ACCEPT, якщо він є.

HTTP_ACCEPT_CHARSET: зміст заголовка ACCEPT-CHARSET, якщо він є. Наприклад 'iso-8859-1, *, utf-8'.

HTTP_ACCEPT_ENCODING: зміст заголовка ACCEPT-ENCODING, якщо він є. Наприклад 'gzip'.

HTTP_ACCEPT_LANGUAGE: зміст заголовка ACCEPT-LANGUAGE, якщо він є. Наприклад 'en'.

HTTP_ACCEPT_CONNECTION: зміст заголовка ACCEPT-CONNECTION, якщо він є. Наприклад 'Keep-Alive'.

HTTP_HOST: зміст заголовка HOST, тобто він є.

HTTP_REFERER: адреса сторінки, яку на поточну сторінку передало програмне забезпечення користувача. Не всі ПЗ користувача передають цей параметр, а деякий ПЗ навіть змінюють його. Отже, даному параметру довіряти не можна.

HTTP_USER_AGENT: цей параметр містить інформацію про клієнта користувача (ПО користувача), яка звертається до сторінки. Наприклад 'Mozilla/4.5 [RU] (X11; U; Linux 2.2.9 i586). Також цю інформацію Ви можете отримати з функції get_browser ().

HTTPS: параметр містить інформацію, якщо запити був зроблений через HTTP.

REMOTE_ADDR: IP-адреса користувача, з якого він переглядає сторінку.

REMOTE_HOST: ім'я хоста користувача, з якого він переглядає цю сторінку.

REMOTE_POST: порт, який використовується для з'єднання з веб-сервером.

SCRIPT_FILENAME: абсолютний шлях до поточного скрипта.

SERVER_ADMIN: значення SERVER_ADMIN, взяте з конфігураційного файлу Apache.

SERVER_PORT: порт веб-сервера, використовуваний для передачі даних по HTTP. За замовчуванням 80.

SERVER_SIGNATURE: рядок, що містить версію веб-сервера і ім'я віртуального хоста.

PATH_TRANSLATED: базовий шлях до поточного сценарію.

SCRIPT_NAME: містить шлях та ім'я поточного скрипта.

REQUEST_URI: URI для поточної сторінки.

PHP_AUTH_DIGEST: якщо PHP працює як модуль Apache, то параметр використовується як аутентифікації по протоколу HTTP для перевірки автентичності.

PHP_AUTH_USER: якщо PHP працює як модуль Apache або IIS, то параметр містить ім'я користувача при аутентифікації по протоколу HTTP.

PHP_AUTH_PW: якщо PHP працює як модуль Apache або IIS, то параметр містить пароль користувача при аутентифікації по протоколу HTTP.

AUTH_TYPE: якщо PHP працює як модуль Apache або IIS, то параметр містить тип аутентифікації по протоколу HTTP.

4.8.2 Суперглобальний масив $ _GET

Масив $ _GET представляє собою асоціативний масив елементів, переданих за допомогою HTTP GET запитів поточному PHP-скрипту. Немає необхідності оголошувати масив $ _GET всередині користувача функцій командою "global $ _GET;", тому що даний масив є суперглобальний.

4.8.3 Суперглобальний масив $ _POST

Масив $ _POST представляє собою асоціативний масив елементів, переданих за допомогою HTTP POST запитів поточному PHP-скрипту. Немає необхідності оголошувати масив $ _POST всередині користувача функцій командою "global $ _POST;", тому що даний масив є суперглобальний.

4.8.4 Суперглобальний масив $ _FILES

Масив $ _FILES представляє собою асоціативний масив елементів, переданих за допомогою HTTP POST запитів поточному PHP-скрипту. Немає необхідності оголошувати масив $ _FILES всередині користувача функцій командою "global $ _FILES;", тому що даний масив є суперглобальний.

4.8.5 Суперглобальний масив $ _COOKIE

Масив $ _COOKIE представляє собою асоціативний масив елементів, переданих за допомогою HTTP COOKIE запитів поточному PHP-скрипту. Немає необхідності оголошувати масив $ _COOKIE всередині користувача функцій командою "global $ _COOKIE;", тому що даний масив є суперглобальний.

4.8.6 Суперглобальний масив $ _SESSION

Даний асоціативний масив містить змінні сесії, доступні для даного скрипта. Немає необхідності оголошувати масив $ _SESSION всередині користувача функцій командою "global $ _SESSION;", тому що даний масив є суперглобальний.

4.8.7 Суперглобальний масив $ _REQUEST

Масив $ _REQUEST є об'єднаним асоціативним масивом, який включає в себе масиви $ _GET, $ _POST, $ _FILES. Немає необхідності оголошувати масив $ _REQUEST всередині користувача функцій командою "global $ _REQUEST;", тому що даний масив є суперглобальний.

4.8.8 Суперглобальний масив $ _ENV

$ _ENV Представляє собою асоціативний масив, який містить значення змінних з середовища, в якій працює інтерпретатор PHP. Немає необхідності оголошувати масив $ _ENV всередині користувача функцій командою "global $ _ENV;", тому що даний масив є суперглобальний.

4.9 PHP і Cookies

4.9.1 Поняття Cookies

Cookies - це механізм зберігання даних браузером віддаленого комп'ютера для ідентифікації відвідувачів і зберігання параметрів веб-сторінок (наприклад, змінних).

Наведемо приклад використання Cookies на конкретному прикладі.

Припустимо, нам потрібно написати лічильник відвідування сайту. Нам потрібно знати, яке число відвідувань сайту здійснювалося кожним конкретним відвідувачем.

Дану задачу можна вирішити двома способами. Перший з них полягає у веденні обліку IP-адрес користувачів. Для цього потрібна база даних всього з однієї таблиці, приблизна структура якої така:

IP-адреса Число відвідувань
210.124.134.203 7
212.201.78.207 14
83.103.203.73 3

Коли користувач заходить на сайт, нам потрібно визначити його IP-адресу, знайти в базі даних інформацію про його відвідини, збільшити лічильник і вивести його в браузер відвідувача. Написати обробник (скрипт) подібної процедури нескладно. Проте при використанні такого методу у нас з'являються проблеми наступного характеру:

- Для кожного IP-адреси потрібно вести облік в одній таблиці, яка може бути дуже великою. А з цього випливає, що ми нераціонально використовуємо процесорний час і дисковий простір;
- У більшості домашніх користувачів IP-адреси є динамічними. Тобто, сьогодні у нього адрес 212.218.78.124, а завтра - 212.218.78.137. Таким чином, велика вірогідність ідентифікувати одного користувача кілька разів.

Можна використовувати другий спосіб, який набагато легший в реалізації і є більш ефективним. Ми встановлюємо в Cookie змінну, яка буде зберігатися на диску віддаленого користувача. Ця змінна і буде зберігати інформацію про

відвідування. Вона буде зчитуватися скриптом при зверненні відвідувача до сервера. Вигода такого методу ідентифікації очевидна. По-перше, нам не потрібно зберігати безліч непотрібної інформації про IP-адреси. По-друге, нас не цікавлять динамічні IP-адреси, оскільки дані про відвідини зберігаються конкретно у кожного відвідувача сайту.

Тепер зрозуміло, для чого ми можемо використовувати Cookie - для зберігання невеликої за обсягом інформації у клієнта (відвідувача) сайту, наприклад: налаштування сайту (колір фону сторінок, мова, оформлення таблиць і.т.д.), а також іншої інформації.

Файли Cookies представляють собою звичайні текстові файли, які зберігаються на диску у відвідувачів сайтів. Файли Cookies і містять ту інформацію, яка була в них записана сервером.

4.9.2 Програмування Cookies

Приступимо до програмування Cookies.

Для установки Cookies використовується функція SetCookie (). Для цієї функції можна вказати шість параметрів, один з яких є обов'язковим:

- name - задає ім'я (рядків), закріплене за Cookie;
- value - визначає значення змінної (рядок);
- expire - час "життя" змінної (ціле число). Якщо цей параметр не вказати, то Cookie будуть "жити" до кінця сесії, тобто до закриття браузера. Якщо час вказано, то, коли воно настане, Cookie самознищиться.
- path - шлях до Cookie (рядок);
- domain - домен (рядок). Як значення встановлюється ім'я хоста, з якого Cookie був встановлений;
- secure - передача Cookie через захищене HTTPS-з'єднання.

Зазвичай використовуються тільки три перші параметра.

Приклад установки Cookies:

<? Php

/ / Встановлюємо Cookie до кінця сесії:

SetCookie ("Test", "Value");

/ / Встановлюємо Cookie на одну годину після установки:

SetCookie ("My_Cookie", "Value", time () +3600);

?>

При використанні Cookies необхідно мати на увазі, що Cookies повинні встановлюватися до першого висновку інформації в браузер (наприклад, оператором echo або висновком якої-небудь функції). Тому бажано встановлювати Cookies на самому початку скрипта. Cookies встановлюються з допомогою певного заголовка сервера, а якщо скрипт виводить що-небудь, то це означає, що починається тіло документа. У результаті Cookies не будуть встановлені і може бути виведено попередження. Для перевірки успішності установки Cookies можна використовувати такий метод:

<? Php

/ / Встановлюємо Cookie до кінця сесії:

/ / В разі успішного встановлення Cookie, функція SetCookie повертає TRUE:

if (SetCookie ("Test", "Value")) echo "<h3>Cookies успішно встановлені! </ h3>";

?>

Функція SetCookie () повертає TRUE у випадку успішної установки Cookie. У випадку, якщо Cookie встановити не вдається SetCookie () поверне FALSE і можливо, попередження (залежить від налаштувань PHP).

Приклад невдалого встановлення Cookie:

<? Php

/ / Cookies встановити не вдасться, оскільки перед відправкою

/ / Заголовка Cookie ми виводимо в браузер рядок 'Hello':

echo "Hello";

/ / Функція SetCookie поверне FALSE:

if (SetCookie ("Test", "Value")) echo "<h3>Cookie успішно встановлено! </ h3>";

elseecho "<h3>Cookie встановити не вдалося! </ h3>";

/ / Виводить 'Cookie встановити не вдалося!'.

?>

Cookie встановити не вдалося, оскільки перед посилкою заголовка Cookie ми вивели в браузер рядок "Hello".

4.9.3 Читання значень Cookies

Отримати доступ до Cookies та їх значенням досить просто. Вони зберігаються в суперглобальних масивах і $ _COOKIE і $ HTTP_COOKIE_VARS.

Доступ до значень здійснюється на ім'я встановлених Cookies, наприклад:

```
echo $ _COOKIE ['my_cookie'];
/ / Виводить значення встановленої Cookie 'My_Cookie'
```

Приклад встановлення Cookie і подальшого його читання:

```
<? Php
/ / Встановлюємо Cookie 'test' зі значенням 'Hello' на одну годину:
setcookie ("test", "Hello", time () +3600);
/ / При наступному запиті скрипта виводить 'Hello':
echo @ $ _COOKIE ['test'];
?>
```

У розглянутому прикладі при першому зверненні до скрипта встановлюється Cookie "test" із значенням "hello". При повторному зверненні до скрипта буде виведено значення Cookie "test", тобто рядок "Hello".

При читанні значень Cookies звертайте увагу на перевірку існування Cookies, наприклад, використовуючи оператор isset (). Або шляхом придушення виводу помилок оператором @

А ось приклад, як побудувати лічильник числа завантажень сторінки за допомогою Cookies:

```
<? Php
/ / Перевіряємо, чи був вже встановлений Cookie 'Mortal',
/ / Якщо так, то читаємо його значення,
/ / І збільшуємо значення лічильника звернень до сторінки:
if (isset ($ _COOKIE ['Mortal'])) $ cnt = $ _COOKIE ['Mortal'] +1;
else $ cnt = 0;
/ / Встановлюємо Cookie 'Mortal' із значенням лічильника,
```

/ / З часом "життя" до 18/07/29,
/ / Тобто на дуже довгий час:
setcookie ("Mortal", $ cnt, 0x6FFFFFFF);
/ / Виводить число відвідувань (завантажень) цієї сторінки:
echo "<p> Ви відвідували цю сторінку <b> ".@$_ COOKIE ['Mortal']."</ b> раз </ p>";
?>

4.9.4 Видалення Cookies

Іноді виникає необхідність видалення Cookies. Зробити це нескладно, необхідно лише знову встановити Cookie з ідентичним ім'ям і порожнім параметром. Наприклад:

<? Php
/ / Видаляємо Cookie 'Test':
SetCookie ("Test ","");
?>

4.9.5 Установка масиву Cookies і його читання

Ми може встановити масив Cookies, використовуючи квадратні дужки в іменах Cookies [], а потім прочитати масив Cookies та значення цього масиву:

<? Php
/ / Встановлюємо масив Cookies:
setcookie ("cookie [1]", "Перший");
setcookie ("cookie [2]", "Другий");
setcookie ("cookie [3]", "Третій");
/ / Після перезавантаження сторінки ми відобразимо
/ / Склад масиву Cookies 'cookie':
if (isset ($ _COOKIE ['cookie'])) {
foreach ($ _COOKIE ['cookie'] as $ name => $ value) {
echo "$ name: $ value
";
}
}
?>

Переваги використання Cookies незаперечні. Проте існують і деякі проблеми їх використання. Перша з них полягає в тому,

що відвідувач може блокувати Cookies браузером або просто видалити всі Cookies або їх частину. Таким чином, ми можемо мати деякі проблеми в ідентифікації таких відвідувачів.

4.10 Робота з сесією

Етапи роботи з сесіями

1. Установка імені сесії (не обов'язково).
2. Створення сесії.
3. Реєстрація змінних сесії і їх використання.
4. Видалення змінних сесії.
5. Знищення сесії.

4.10.1 Створення сесії

Перше, що потрібно зробити для роботи з сесіями (якщо вони вже налаштовані адміністратором сервера), це запустити механізм сесій. Якщо в налаштуваннях сервера змінна session.auto_start встановлена в значення "0" (якщо session.auto_start = 1, то сесії запускаються автоматично), то будь-який скрипт, в якому потрібно використовувати дані сесії, повинен починатися з команди

session_start ();

Отримавши таку команду, сервер створює нову сесію або відновлює поточну, ґрунтуючись на ідентифікаторі сесії, переданому по запиту. Як це робиться? Інтерпретатор PHP шукає змінну, в якій зберігається ідентифікатор сесії (за умовчанням це PHPSESSID) спочатку в cookies, потім в змінних, переданих за допомогою POST-і GET-запитів. Якщо ідентифікатор знайдений, то користувач вважається ідентифікованим, проводиться заміна всіх URL і виставлення cookies. В іншому випадку користувач вважається новим, для нього генерується новий унікальний ідентифікатор, потім проводиться заміна URL і виставлення cookies.

Команду session_start () потрібно викликати у всіх скриптах, у яких доведеться використовувати змінні сесії, причому до

виведення яких-небудь даних в браузер. Це пов'язано з тим, що cookies виставляються тільки до виведення інформації на екран.

Отримати ідентифікатор поточної сесії можна за допомогою функції session_id ().

Для наочності сесії можна задати ім'я за допомогою функції session_name ([імя_сессії]). Робити це потрібно ще до ініціалізації сесії. Отримати ім'я поточної сесії можна за допомогою цієї ж функції, викликаної без параметрів: session_name ();

Ім'я сесії - це ім'я параметра, в якому зберігається ідентифікатор сесії.

Перейменуємо наш файл index.html, щоб оброблялися php-скрипти, наприклад в Index.php, створимо сесію і подивимося, як вона отримає ідентифікатор та ім'я.

```
<?
session_start ();
/ / Створюємо нову сесію або
/ / Відновлюємо поточну
echo session_id ();
/ / Виводимо ідентифікатор сесії
?>
<html>
<head><title>Myhomepage</ title></ head>
... / / Домашня сторінка
</ Html>
<?
echo session_name ();
/ / Виводимо ім'я поточної сесії.
/ / У даному випадку це PHPSESSID
?>
```

Приклад. Створення сесії

Якщо проробити те ж саме з файлом authorize.php, то значення змінних, що виводяться (id сесії та її ім'я) будуть такими ж, якщо перейти на нього з index.php і не закривати перед цим вікно браузера, тоді ідентифікатор сесії зміниться.

4.10.2 Реєстрація змінних сесії

Однак від самого ідентифікатора та імені сесії нам користі для вирішення наших завдань небагато. Ми ж хочемо передавати і зберігати протягом сесії наші власні змінні (наприклад, логін і пароль). Для того щоб цього досягти, потрібно просто зареєструвати свої змінні:

session_register (ім'я_змінної1,
ім'я_змінної2, ...);

Увага! session_register є застарілою і у версії PHP 6.0 взагалі вилучена. Її використання вкрай не бажано. Замість session_register слід використовувати $ _SESSION.

Зауважимо, що реєструються не значення, а імена змінних. Зареєструвати змінну достатньо один раз на будь-якій сторінці, де використовуються сесії. Імена змінних передаються функції session_register () без знаку $. Усі зареєстровані таким чином змінні стають глобальними (тобто доступними з будь-якої сторінки) протягом даної сесії роботи з сайтом.

Зареєструвати змінну також можна, просто записавши її значення в асоціативний масив $ _SESSION, тобто написавши

$ _SESSION ['Ім'я_змінної'] =
'Значення_змінної';

У цьому масиві зберігаються всі зареєстровані (тобто глобальні) змінні сесії.

Доступ до таких змінних здійснюється за допомогою масиву $ _SESSION ['ім'я_змінної']. Якщо ж в налаштуваннях php включена опція register_globals, то до сесійним змінним можна звертатися ще й як до звичайних змінних, наприклад так: $ ім'я_змінної.

Якщо register_globals = off (відключені), то користуватися session_register () для реєстрації змінних переданих методами POST або GET, не можна, тобто це просто не працює. І взагалі, не рекомендується одночасно використовувати обидва методи реєстрації змінних, $ _SESSION і session_register ().

Приклад. Реєстрація змінних

Зареєструємо логін і пароль, що вводяться користувачем на сторінці авторизації.

```
<?
session_start ();
/ / Створюємо нову сесію або
/ / Відновлюємо поточну
if (! isset ($ _GET ['go'])){
echo "<form>
Login: <input type=textname=login>
Password: <input type = password
name = passwd>
<input type=submitname=govalue=Go>
</ Form> ";
} Else {
$ _SESSION ['Login']=$_ GET [' login '];
/ / Реєструємо змінну login
$ _SESSION ['Passwd']=$_ GET [' passwd '];
/ / Реєструємо змінну passwd
/ / Тепер логін і пароль - глобальні
/ / Змінні для цієї сесії
if ($ _GET ['login']==" pit "&&
$ _GET ['Passwd']==" 123 ") {
Header ("Location: secret_info.php");
/ / Перенаправляємо на сторінку
/ / Secret_info.php
} Elseecho "Невірний введення,
спробуйте ще раз <br> ";
}
print_r ($ _SESSION);
/ / Виводимо всі змінні сесії
?>
```

Приклад. authorize.php

Тепер, потрапивши на сторінку secret_info.php, та й на будь-яку іншу сторінку сайту, ми зможемо працювати з введеними користувачем логіном і паролем, які будуть зберігатися в масиві

$ _SESSION. Таким чином, якщо змінити код секретної сторінки (зауважте, ми перейменували її в secret_info.php) так:

```
<? Php
session_start ();
/ / Створюємо нову сесію або
/ / Відновлюємо поточну
print_r ($ _SESSION);
/ / Виводимо всі змінні сесії
?>
<html>
<head><title>Secretinfo</ title></ head>
<body>
<p> Тут я хочу ділитися секретами
з одним Петром.
</ body>
</ html>
```

Приклад. secret_info.php

То ми отримаємо в браузері на секретній сторінці наступне:

```
Array ([login] =>pit [passwd] => 123)

Тут я хочу ділитися секретами
з одним Петром.
```

У результаті отримаємо список змінних, зареєстрованих на authorize.php і, власне, саму секретну сторінку.

Що це нам дає? Припустимо, хакер хоче прочитати секрети Васі і Петі. І він як-то дізнався, як називається секретна сторінка (або сторінки). Тоді він може спробувати просто ввести її адресу в рядку браузера, минаючи сторінку авторизації (введення пароля). Щоб уникнути такого проникнення в наші таємниці, потрібно дописати всього пару рядків у код секретних сторінок:

```
<? Php
session_start ();
/ / Створюємо нову сесію або
/ / Відновлюємо поточну
```

```
print_r ($ _SESSION);
/ / Виводимо всі змінні сесії
if (!($_ SESSION ['login'] == pit&&
$ _SESSION ['Passwd'] == 123))
/ / Перевіряємо правильність
/ / Пароля-логіна
Header ("Location: authorize.php");
/ / Якщо помилка, то перенаправляємо на
/ / Сторінку авторизації
?>
<html>
<head><title>Secretinfo</ title></ head>
... / / Тут розташовується
/ / Секретна інформація:)
</ Html>
```

Приклад. 2-я версія secret_info.php

4.10.3 Видалення змінних сесії

Крім вміння реєструвати змінні сесії (тобто робити їх глобальними протягом всього сеансу роботи), корисно також вміти видаляти такі змінні і сесію в цілому.

Функція **session_unregister** (ім'я_змінної) видаляє глобальну змінну з поточної сесії (тобто видаляє її з списку зареєстрованих змінних). Якщо реєстрація проводилася за допомогою $ _SESSION, то використовують мовну конструкцію unset (). Вона не повертає ніякого значення, а просто знищує зазначені змінні.

Увага! session_unregister застаріла! Слід використовувати unset ($ _SESSION ['ім'я змінної']);

Де це може стати в нагоді? Наприклад, для знищення даних про відвідувача (зокрема, логіна і пароля) після його відходу з секретної сторінки. Якщо правильні логін і пароль збережуться і вікно браузера після відвідування сайту не закрили, то будь-який інший користувач цього комп'ютера зможе прочитати закриту інформацію.

Приклад. Знищення змінних сесії

У файл secret_info.phpдодамо рядок для виходу на головну сторінку:

```
<? Php
/ / ... php код
?>
<html>
<head><title>Secretinfo</ title></ head>
... / / Тут розташовується
/ / Секретна інформація:)
<a href="index.php"> На головну </ a>
</ Html>
```

Приклад. secret_info.php

У Index.php знищимо логін і пароль, введені раніше:

```
<?
session_start ();
session_unregister ('passwd');
/ / Знищуємо пароль
unset ($ _SESSION ['login']);
/ / Знищуємо логін
print_r ($ _SESSION);
/ / Виводимо глобальні змінні сесії
?>
<html>
<head><title>Myhomepage</ title> / head>
..../ / Домашня сторінка
</ Html>
```

Приклад. Index.php

Тепер, щоб потрапити на секретну сторінку, потрібно буде знову вводити логін і пароль.

Для того щоб скинути значення всіх змінних сесії, можна використовувати функцію session_unset ();

Знищити поточну сесію цілком можна командою session_destroy (); Вона не скидає значення глобальних змінних сесії і не видаляє cookies, а знищує всі дані, асоційовані з поточною сесією.

```
<?
```

```
session_start (); / / ініціюємо сесію
$ Test = "Мінлива сесії";
$ _SESSION ['Test'] = $ test;
/ / Реєструємо змінну $ test.
/ / Якщо register_globals = on,
/ / То можна використовувати
/ / Session_register ('test');

print_r ($ _SESSION);
/ / Виводимо всі глобальні змінні

echo session_id ();
/ / Виводимо ідентифікатор сесії

echo "<hr>";
session_unset ();
/ / Знищуємо всі глобальні
/ / Змінні сесії
print_r ($ _SESSION);
echo session_id ();
echo "<hr>";
session_destroy (); / / знищуємо сесію
print_r ($ _SESSION);
echo session_id ();
?>
```

Приклад Знищення сесії і глобальних змінних

У результаті роботи цього скрипта будуть виведені три рядки: у першому - масив з елементом test і його значенням, а також ідентифікатор сесії, у другому - порожній масив і ідентифікатор сесії, в третьому - порожній масив. Таким чином, видно, що після знищення сесії знищується і її ідентифікатор, і ми більше не можемо ні реєструвати змінні, ні взагалі проводити які-небудь дії з сесією.

Розділ 2 Обробка даних в PHP
Тема 5 Математичні функції в PHP

Математичні функції призначені для роботи змінними чисельних типів. Це означає, що операндами будуть змінні типу int і float.

Функції можна поділити на кілька типів:

0. Загальні функції
1. Тригонометричні функції
2. Зворотні тригонометричні функції
3. Логарифмічні функції
4. Функції перетворення підстав обчислення
5. Решта (які важко класифікувати)

5.1 Загальні математичні функції

До загальних функцій відносяться функції:

abs
floor
ceil
max
min
round
rand
sqrt

Опишемо їх більш докладно.

abs

Синтаксис:

numberabs (mixednumber)

Функція обчислює модуль числа. Нагадаю, що | x | = x, якщо x> 0 або x = 0, і | x | =- x, якщо x <0.

floor

Синтаксис:

floatfloor (floatvalue)

Ця функція округляє дріб в меншу сторону. Але не варто думати, що це - "відкидання" дробової частини. Тому що, по-перше, результат все одно буде не цілочисловим, а речовим. До

того ж для негативних чисел найближчим меншим цілим числом буде дріб, доповнена до нього.

ceil

Синтаксис:

floatceil (floatvalue)

Ця функція також округлює аргумент до найближчого цілого, але вже до найближчого більшого цілого. Функції floor і ceil повертають речовинний результат з однієї простої причини - діапазон дійсних чисел в PHP більше, ніж цілих.

max

Синтаксис:

mixedmax (number arg1, number arg2 [, number ...])

mixedmax (arraynumbers)

Ця функція має 2 описи синтаксису. Вона повертає максимальне значення серед аргументів, якщо вони передані у вигляді повного списку (1-й варіант синтаксису), або ж найбільший елемент масиву, у разі, якщо був переданий масив (2-й варіант синтаксису). Цікаво ось що: аргументи можуть бути різних типів. Скажімо,

< ? PHP

$ Test = max ('рядок', array (0, 1), 4, 7);

/ / Повернеться array (0, 1) - масив завжди буде вважатися більший решти, хоча таке "порівняння" виглядає досить безглуздо

?>

Я не можу уявити, навіщо таке може знадобитися, але тим не менш, в PHP це працює саме так.

Повністю аналогічна функція min, її можна навіть не описувати. Різниця в тому, що повертається мінімум.

round

Синтаксис:

floatround (floatval [, intprecision])

Округлює дійсне число за арифметичними правилами. Можна вказати точність заокруглення. У цьому випадку буде

вестися округлення до вказаного числа знаків дробу. Цю точність можна вказати рівною 0 (рівносильно викликом функції без вказівки точності). Крім цього, можна вказати і негативне значення точності. Результат аналогічний.

rand

Синтаксису:

intrand ([intmin, intmax])

Повертає випадкове ціле число в діапазоні від min до max включно. Ці параметри не обов'язкові. Якщо їх не вказувати, повернеться випадкове число в діапазоні від 0 до константи RAND_MAX.

На операційних системах сімейства Windows значення RAND_MAX всього лише 32767

Якщо потрібно випадкове дробове число, то можна скористатися, наприклад, таким кодом:

<? PHP

/ / Генеруємо випадкове дійсне число в діапазоні $ a.. $ B у припущенні $ a <$ b

$ IMaxRand = 30000; / / по суті, можна задавати і більше для більшої точності

$ IRand = rand (1, $ iMaxRand);

$ FRand = $ a + ($ b-$ a) * $ iRand / $ iMaxRand;

?>

sqrt

Синтаксис:

floatsqrt (floatarg)

Функція обчислює квадратний корінь аргументу. І аргумент і результат - речові.

5.2 Тригонометричні і зворотні тригонометричні функції

До тригонометричним функціям я відніс функції:

acos

acosh
asin
asinh
atan2
atan
atanh
cos
cosh
sin
sinh
tan
tanh

Всі ці функції приймають в якості аргументу параметри типу float, і повертають значення цього ж типу. З тієї причини, що для їх опису потрібно лише знання, що конкретно кожна з них робить, не буду наводити тут всі описи. (А призначення цих функцій відомо з геометрії та алгебри).

acos

Синтаксис:

floatacos (floatarg)

Обчислює арккосинус числа. У разі, якщо аргумент неприпустимий (тобто | arg |> 1), повертає NAN.

5.3 Логарифмічні функції

До логарифмічним функцій я відніс функції:

log10
log1p
log

Ці функції також не вимагають окремого опису. Окремо скажу лише про функції log1p. Справа в тому, що, в силу особливості самого логарифма, при значеннях аргументу, близького до 1, він сильно прагне до нуля. Це може бути перешкодою для "тонких" обчисленнях. Ця функція обчислює значення більш точним способом.

Окрім іншого, наведу приклад корисної функції, яка обчислює логарифм з довільною основою:

```
<? PHP
functionLog ($ base, $ arg) {
if ($ base == 1 | | $ base<= 0)
{ returnnull; }
if ($ arg<= 0) {
returnnull; }
returnlog ($ arg) / log ($ base);
}
?>
```

Так, до речі, питання для читачів - я назвав функцію Log - чи не викличе це конфлікту з назвою вже існуючої функції log?

5.4 Функції перетворення підстав обчислення

У php є спеціальний ряд функцій, які працюють з перетворенням систем числення. Якщо врахувати, що bin - двійкова система, oct - восмирічна, dec - десяткова, а hex – шістнадцатирічна, то призначення багатьох з них стає інтуїтивно зрозумілим:

```
base_convert
bindec
decbin
dechex
decoct
hexdec
octdec
```

Інтерес же представляє функція base_convert:

```
octdecbase_convert
octdec Синтаксис:
octdecstringbase_convert (stringnumber, intfrombase, inttobase)
```

Повертає рядок, що містить число number, надане з базою tobase.База, в якій number дається, специфікується в frombase. I frombase, і tobase повинні бути в діапазоні від 2 до 36 включно.

Чому таке обмеження? Та тому, що цифри, великі 10, записуються за допомогою символів латинського алфавіту. Тобто від a до z. Разом з першими 9-у підставами вийде якраз 35 можливих значень.

5.5 Некласифіковані функції

На закінчення наведемо ряд корисних, не класифікованих окремо, функцій:

is_finite
is_infinite
is_nan

is_finite
Синтаксис:
boolis_finite (floatval)
Ця функція визначає, звичайно чи число з точки зору діапазону чисел з плаваючою крапкою. Цей діапазон береться виходячи з поточної платформи.

is_finite

Синтаксис:
boolis_infinite (floatval)
Ця функція визначає, нескінченно чи число з точки зору діапазону чисел з плаваючою крапкою. При цьому, звичайно, враховуються можливості платформи, але, скажімо, результат log (0) не буде таким ні на якій платформі (можете подумати, чому)
is_nan

Синтаксис:
boolis_nan (floatval)
Показує, чи є аргумент дійсним числом. З точки зору алгебри, в категорію не дійсних чисел потраплять як не числа зовсім (на кшталт арккосинуса від двійки), так і комплексні числа (на

кшталт квадратного кореня з -1). В обох випадках функція поверне false.

5.6 Функції для роботи з датою і часом

Ці функції дозволяють отримати поточний час на сервері, на якому виконується скрипт. Крім того, що отримати, час можна представити в різних форматах, порахувати різницю між двома моментами часу і навіть дізнатися час сходу сонця в певній місцевості в той чи інший день!

boolcheckdate (int $ month, int $ day, int $ year)

Повертає TRUE якщо дата, визначена аргументами, є правильною; інакше повертає FALSE. Дата вважається правильною, якщо:

рік у діапазоні від 1 до 32767 включно

місяць в діапазоні від 1 до 12 включно

day є допустимим номером дня для місяця, заданого аргументом month, беручи до уваги, що year може задавати високосний рік.

arraydate_parse (string $ date) Повертає асоціативний масив з інформацією про дату $ date. Масив містить рік, день, місяць, години, хвилини, секунди і ще щось цікаве.

arraydate_sun_info (int $ time, float $ latitude, float $ longitude) Повертає масив з часом сходу, заходу сонця, тривалості світлового дня та ін.

Аргументи - мітка часу, яку можна, наприклад з рядка отримати функцією datetotime ($ str). Або функцією time () - поточний час.

stringdate (string $ format [, int $ timestamp])

Повертає час, відформатований відповідно до аргументом format, використовуючи мітку часу, задану аргументом timestamp або поточне системний час, якщо timestamp не заданий.

Формат - це рядок, що містить символи форматування. І звичайні символи теж. Звичайні виводяться як є, а символи форматування замінюються відповідними значеннями:

Символ у рядку format Опис

Приклад значення, що повертається

a Antemeridiem або Postmeridiem в нижньому регістрі am або pm

A Antemeridiem або Postmeridiem у верхньому регістрі AM або PM

B Час в стандарті SwatchInternet Від 000 до 999

c Дата в форматі ISO 8601 (додано в PHP 5) 2004-02-12T15: 19:21 +00:00

d День місяця, 2 цифри з провідними нулями від 01 до 31

D Скорочене найменування дня тижня, 3 символи від Mon до Sun

F Повне найменування місяця, наприклад January або March від January до December

g Годинник у 12-годинному форматі без ведучих нулів Від 1 до 12

G Годинник у 24-годинному форматі без ведучих нулів Від 0 до 23

h Годинник у 12-годинному форматі з провідними нулями Від 01 до 12

H Годинник у 24-годинному форматі з провідними нулями Від 00 до 23

i Хвилини з провідними нулями 00 to 59

I (заголовна i) Ознака літнього часу 1, якщо дата відповідає літньому часу, інакше 0 otherwise.

j День місяця без ведучих нулів Від 1 до 31

l (мала 'L') Повне найменування дня тижня Від Sunday до Saturday

L Ознака високосного року 1, якщо рік високосний, інакше 0.

m Порядковий номер місяця з провідними нулями Від 01 до 12

M Скорочене найменування місяці, 3 символи Від Jan до Dec

n Порядковий номер місяця без ведучих нулів Від 1 до 12

O Різниця з часом по Грінвічу в годинах Наприклад: +0200

r Дата в форматі RFC 2822

Наприклад: Thu, 21 Dec 2000 16:01:07 +0200

s Секунди з провідними нулями Від 00 до 59

S Англійська суфікс порядкового числівника дня місяця, 2 символи st, nd, rd або th. Застосовується спільно з j

t Кількість днів у місяці Від 28 до 31

T Тимчасова зона на сервері Приклади: EST, MDT ...

U Кількість секунд, що пройшли з початку Епохи Unix (TheUnixEpoch, 1 січня 1970, 00:00:00 GMT) Див також time ()

w Порядковий номер дня тижня Від 0 (неділя) до 6 (субота)

W Порядковий номер тижня року по ISO-8601, перший день тижня - понеділок Наприклад: 42 (42-й тиждень року)

Y Порядковий номер року, 4 цифри Приклади: 1999, 2003

y Номер року, 2 цифри Приклади: 99, 03

z Порядковий номер дня в році (нумерація з 0) Від 0 до 365

Z Зсув тимчасової зони в секундах. Для тимчасових зон захід UTC це негативне число, на схід UTC - позитивне. Від -43200 до 43200

inttime (void)

Повертає кількість секунд, що пройшли з початку Епохи Unix (TheUnixEpoch, 1 січня 1970, 00:00:00 GMT) до поточного часу.

Тема 6 Керуючі конструкції

6.1 Умовні оператори

6.1.1 Оператор if

Це один з найважливіших операторів багатьох мов, включаючи PHP. Він дозволяє виконувати фрагменти коду в залежності від умови. Структуру оператора if можна представити наступним чином:

if (вираз) блок_виконання

Тут вираз є будь-який правильний PHP-вираз (тобто все, що має значення). У процесі обробки скрипта вираз перетвориться до логічного типу. Якщо в результаті перетворення значення виразу істинно (True), то виконується блок_виконання. В іншому випадку блок_виконання ігнорується. Якщо блок_виконання містить кілька команд, то він повинен бути укладений у фігурні дужки {}.

Правила перетворення виразу до логічного типу:

1. Правила перетворення виразу до логічного типу:
   - логічне False
   - цілий нуль (0)
   - дійсний нуль (0.0)
   - порожній рядок і рядок "0"
   - масив без елементів
   - об'єкт без змінних (детально про об'єкти буде розказано в одній з наступних лекцій)
   - спеціальний тип NULL
2. Всі інші значення перетворюються в TRUE.

Приклад. Умовний оператор if

```
<? $ Names = array ("Іван", "Петро", "Семен");
if ($ names [0] == "Іван") {
echo "Привіт, Ваня!";
$ Num = 1;
$ Account = 2000;
}
if ($ num) echo "Іван перший у списку!";
```

```
$ Bax = 30;
if ($ account> 100 * $ bax +3)
echo "Цей рядок не з'явиться
на екрані, так як умова не виконана ";
?>
```

6.1.2 Оператор else

Ми розглянули тільки одну, основну частину оператора if. Існує кілька розширень цього оператора. Оператор else розширює if на випадок, якщо вираз, що перевіряється в if є невірним, і дозволяє виконати будь-які дії за таких умов.

Структуру оператора if, розширеного за допомогою оператора else, можна представити таким чином:

```
if (вираз) блок_виконання
  else блок_виконання1
```

Цю конструкцію if ... else можна інтерпретувати приблизно так: якщо виконана умова (тобто вираз = true), то виконуємо дії з блоку_виконання, інакше - дії з блоку_виконання1. Використовувати оператор else не обов'язково.

Подивимося, як можна змінити попередній приклад, з огляду на необхідність здійснення дій та в разі невиконання умови.

Приклад. Оператор else

```
<?
$ Names = array ("Іван", "Петро", "Семен");
if ($ names [0] == "Іван") {
echo "Привіт, Ваня!";
$ Num = 1;
$ Account = 2000;
} Else {
echo "Привіт, $ names [0].
А ми чекали Ваню: (";
}
if ($ num) echo "Іван перший у списку!";
else echo "Іван НЕ перший у списку?!";
$ Bax = 30;
if ($ account> 100 * $ bax +3)
```

```
echo "Цей рядок не з'явиться на екрані,
так як умова не виконана ";
else echo "Зате з'явиться цей рядок!";
?>
```

6.1.3 Оператор elseif

Ще один спосіб розширення умовного оператора if - використання оператора elseif. Elseif - це комбінація else і if. Як і else, він розширює if для виконання різних дій у тому випадку, якщо умова, що перевіряється в if, невірно. Але на відміну від else, альтернативні дії будуть виконані, тільки якщо elseif-умова є вірним. Структуру оператора if, розширеного за допомогою операторів else і elseif, можна представити таким чином:

```
if (вираз) блок_виконання
elseif (вираз1) блок_виконання1
...
else блок_ виконання N
```

Операторів elseif може бути відразу кілька в одному if-блоці. Elseif-твердження буде виконано, тільки якщо попереднє if-умова є False, всі попередні elseif-умови є False, а дане elseif-умова - True.

Приклад. Оператор elseif

```
<?
$ Names = array ("Іван", "Петро", "Семен");
if ($ names [0] == "Іван") {
/ / Якщо перше ім'я в масиві Іван
echo "Привіт, Ваня!";
} Elseif ($ names [0] == "Петро") {
/ / Якщо перше ім'я
/ / Не Іван, а Петро
echo "Привіт, Петя!";
} Elseif ($ names [0] == "Семен") {
/ / Якщо перше ім'я не
/ / Іван, не Петро, а Семен
echo "Привіт, Сеня!";
} Else {
```

```
/ / Якщо перше ім'я не Іван,
/ / Не Петро І не Семен
echo "Привіт, $ names [0]. А ти хто такий?";
}
?>
```

6.1.4 Оператор switch

Ще одна конструкція, що дозволяє перевіряти умови і виконувати в залежності від цього різні дії, - це **switch**. У залежності від того, яке значення має змінна, він перемикається між різними блоками дії. switch дуже схожий на оператор if ... elseif ... else або набір операторів if. Структуру switch можна записати наступним чином:

```
switch (вираз чи змінна) {
case значення1:
блок_дій1
break;
case значення2:
блок_дій2
break;
...
default:
блок_дій_при_замовчуванні
}
```

На відміну від if, тут значення виразу не приводиться до логічного типу, а просто порівнюється зі значеннями, перерахованими після ключових слів case (значення1, значення2 і т.д.). Якщо значення виразу співпало з якимсь варіантом, то виконується відповідний блок_дій - від двокрапки після значення, що співпало до кінця switch або до першого оператора break, якщо такий знайдеться. Якщо значення виразу не співпало з жодним із варіантів, то виконуються дії за умовчанням (блок_дій_при_замовчуванні), що знаходяться після ключового слова default. Вираз в switch обчислюється тільки один раз, а в операторі elseif - кожен раз, тому, якщо вираз досить складний, то switch працює швидше.

Приклад можна переписати з використанням **switch** наступним чином:

```
<?
$ Names = array ("Іван", "Петро", "Семен");
switch ($ names [0]) {
case "Іван":
echo "Привіт, Ваня!";
break;
case "Петро":
echo "Привіт, Петя!";
break;
case "Семен":
echo "Привіт, Сеня!";
break;
default:
echo "Привіт, $ names [0].
А як Вас звати? ";
}
?>
```

Якщо в цьому прикладі опустити оператор break, наприклад, в case "Петро":, то, якщо змінна виявиться рівною рядку "Петро", після виведення на екран повідомлення "Привіт, Петя!" програма піде далі і виведе також повідомлення "Привіт, Сеня!" і тільки потім, зустрівши break, продовжить своє виконання за межами switch.

Для конструкції switch, як і для if, можливий альтернативний синтаксис, де відкриває switch фігурна дужка замінюється двокрапкою, а закриває - endswitch; відповідно.

6.2 Цикли

У PHP існує кілька конструкцій, що дозволяють виконувати повторювані дії в залежності від умови. Це цикли while, do .. while, foreach та for. Розглянемо їх більш докладно.

6.2.1 while

Структура:

while (вираз) {блок_виконання}
або
while (вираз): блок_виконання endwhile;

while - простий цикл. Він наказує PHP виконувати команди блоку_виконання до тих пір, поки вираз обчислюється як True (тут, як і в if, відбувається приведення вислови до логічного типу). Значення виразу перевіряється щоразу на початку циклу, так що, навіть якщо його значення змінилося в процесі виконання блоку_виконання, цикл не буде зупинено до кінця ітерації (тобто поки всі команди блоку_виконання не будуть виконані).

Приклад. Оператор while

```
<?
/ / Ця програма надрукує всі парні цифри
$ I = 1;
while ($ i <10) {
if ($ i% 2 == 0) print $ i;
/ / Друкуємо цифру, якщо вона парна
$ I + +;
/ / I збільшуємо $ i на одиницю
}
?>
```

6.2.2 do ... while

Цикли do .. while дуже схожі на цикли while, з тією лише різницею, що істинність висловлювання перевіряється наприкінці циклу, а не на початку. Завдяки цьому блок_виконання циклу **do ... while** гарантовано виконується хоча б один раз.

Структура:

do {блок_виконання} while (вираз);

Приклад. Оператор do .. while

```
/ / Ця програма надрукує число 12, незважаючи на те
/ / Що умова циклу не виконано
$ I = 12;
```

```
do {
if ($ i% 2 == 0) print $ i;
/ / Якщо число парне, то друкуємо його
$ I + +;
/ / Збільшуємо число на одиницю
} While ($ i <10)
?>
```

6.2.3 for

Це найскладніші цикли в PHP. Вони нагадують відповідні цикли C.

Структура:

```
for (вираз1; вираз2; вираз3) {блок_виконання}
або
for (вираз1; вираз2; вираз3): блок_виконання endfor;
```

Тут, як ми бачимо, умова складається одразу з трьох виразів. Перший вираз вираз1 обчислюється безумовно один раз на початку циклу. На початку кожної ітерації обчислюється вираз2. Якщо він є True, то цикл продовжується і виконуються всі команди блоку_виконання. Якщо вираз2 обчислюється як False, то виконання циклу зупиняється. В кінці кожної ітерації (тобто після виконання всіх команд блоку_виконання) обчислюється вираз3.

Кожне з виразів 1, 2, 3 може бути порожнім. Якщо вираз2 є порожнім, то це значить, що цикл повинен виконуватися невизначений час (у цьому випадку PHP вважає це вираз завжди істинним). Це не так марно, як здається, адже цикл можна зупиняти, використовуючи оператор break.

Наприклад, всі парні цифри можна вивести з використанням циклу for таким чином:

```
<? Php
for ($ i = 0; $ i <10; $ i + +) {
if ($ i% 2 == 0) print $ i;
/ / Друкуємо парні числа
}
```

```
?>
```

Якщо опустити другий вираз (умова $ i <10), то таку ж задачу можна вирішити, зупиняючи цикл оператором break.

```
< ? Php
for ($ i = 0;; $ i + +) {
if ($ i> = 10) break;
/ / Якщо $ i більше або дорівнює 10,
/ / То припиняємо роботу циклу
if ($ i% 2 == 0) print $ i;
/ / Якщо число парне,
/ / То друкуємо його
}
?>
```

Можна опустити всі три вирази. У цьому випадку просто не буде задано початкове значення лічильника $ i і воно не буде змінюватися кожного разу наприкінці циклу. Всі ці дії можна записати у вигляді окремих команд або в блоці_виконання, або перед циклом:

```
<? Php
$ I = 2; / / задаємо початкове значення лічильника
for (;;) {
if ($ i> = 10) break;
/ / Якщо $ i більше або дорівнює 10,
/ / То припиняємо роботу циклу
if ($ i% 2 == 0) print $ i;
/ / Якщо число парне,
/ / То друкуємо його
$ I + +; / / збільшуємо лічильник на одиницю
}
?>
```

У третє вираз конструкції for можна записувати через кому відразу кілька найпростіших команд. Наприклад, якщо ми хочемо просто вивести всі цифри, то програму можна записати зовсім просто:

```
<? Php
for ($ i = 0; $ i <10; print $ i, $ i + +)
```

/ * Якщо блок_виконання не містить команд
або містить тільки одну команду,
фігурні дужки, в які він укладений,
можна опускати * /
?>

6.2.4 foreach

Ще одна корисна конструкція. Вона з'явилася тільки в PHP4 і призначена виключно для роботи з масивами.

Синтаксис:

foreach ($ array as $ value) {блок_виконання}
або
foreach ($ array as $ key => $ value)
{Блок_виконання}

У першому випадку формується цикл по всіх елементах масиву, заданого змінною $ array. На кожному кроці циклу значення поточного елемента масиву записується в змінну $ value, і внутрішній лічильник масиву пересувається на одиницю (так що на наступному кроці буде видно наступний елемент масиву). Усередині блоку_виконання значення поточного елемента масиву може бути отримано за допомогою змінної $ value. Виконання блоку_виконання відбувається стільки разів, скільки елементів в масиві $ array.

Друга форма запису на додаток до перерахованого вище на кожному кроці циклу записує ключ поточного елемента масиву в змінну $ key, яку теж можна використовувати в блоці_виконання.

Коли foreach починає виконання, внутрішній покажчик масиву автоматично встановлюється на перший елемент.

Приклад. Оператор foreach

<? Php
$ Names = array ("Іван", "Петро", "Семен");
foreach ($ names as $ val) {
echo "Привіт, $ val
";
/ / Виведе всім вітання

```
}
foreach ($ names as $ k => $ val) {
/ / Крім привітання,
/ / Виведемо номера в списку, тобто ключі
echo "Привіт, $ val!
Ти в списку під номером $ k <br> ";
}
?>
```

6.3 Оператори передачі управління

Іноді потрібно негайно завершити роботу циклу або окремої його ітерації. Для цього використовують оператори break та continue.

6.3.1 Break

Приклад. Оператор break

```
<? Php
$ I = 1;
while ($ i) {
$ N = rand (1,10);
/ / Генеруємо довільне число
/ / Від 1 до 10
echo "$ i: $ n";
/ / Виводимо номер ітерації і
/ / Згенероване число
if ($ n == 5) break;
/ * Якщо було створене число 5,
то припиняємо роботу циклу. У цьому випадку
все, що знаходиться після цього рядка
всередині циклу, не буде виконана * /
echo "Цикл працює <br>";
$ I + +;
}
echo "<br> Число ітерацій циклу $ i";
?>
```

Результатом роботи цього скрипта буде приблизно наступне:
1:7 Цикл працює
2:2 Цикл працює
3:5
Число ітерацій циклу 3
Трохи змінимо наш скрипт:

```
<? Php
$ I = 1;
while ($ i) {
$ N = rand (1,10);
/ / Генеруємо довільне число
/ / Від 1 до 10
switch ($ n) {
case 5:
echo "<font color=blue>
Вихід з switch (n = $ n) ";
break 1;
/ / Припиняємо роботу switch
/ / (Перший містить break циклу)
case 10:
echo "<font color=red>
Вихід з switch i
while (n = $ n) ";
break 2;
/ / Припиняємо роботу switch i while
/ / (Два містять break циклів)
default:
echo "switch працює (n = $ n),";
}
echo "while працює - крок $ i <br>";
$ I + +;
}
echo "<br> Число ітерацій циклу $ i";
?>
```

6.3.2 continue

Іноді потрібно не повністю припинити роботу циклу, а тільки почати його нову ітерацію. Оператор continue дозволяє пропустити подальші інструкції з блоку_виконання будь-якого циклу і продовжити виконання з нового кола. continue можна використовувати з числовим аргументом, який вказує, скільки його керуючих конструкцій повинні завершити роботу.

Замінимо в прикладі попереднього параграфа оператор break на continue. Крім того, обмежимо кількість кроків циклу трьома.

```
<? Php
$ I = 1;
while ($ i <= 4) {
$ N = rand (1,10);
/ / Генеруємо довільне число
/ / Від 1 до 10
echo "$ i: $ n";
/ / Виводимо номер ітерації і
/ / Згенероване число
if ($ n == 5) {
echo "Нова ітерація <br>";
continue;

/ * Якщо було створене число 5,
то починаємо нову ітерацію циклу,
$ I не збільшується * /
}
echo "Цикл працює <br>";
$ I + +;
}
- $ I;
echo "<br> Число ітерацій циклу $ i";
?>
```

Результатом роботи цього скрипта буде

```
1:10 Цикл працює
2:5 Нова ітерація
2:1 Цикл працює
```

3:1 Цикл працює
Число ітерацій циклу 4

Зауважимо, що після виконання оператора continue робота циклу не закінчується. У прикладі лічильник циклу не змінюється в разі отримання числа 5, оскільки він перебуває після оператора continue. Фактично за допомогою continue ми намагаємося уникнути ситуації, коли буде створене число 5. Тому можна було просто написати, замінивши оператор continue на перевірку істинності висловлювання:

```
<? Php
$ I = 1;
while ($ i <4) {
$ N = rand (1,10);
/ / Генеруємо довільне число
/ / Від 1 до 10
if ($ n! == 5) {
echo "$ i: $ n <br>";
/ / Виводимо номер ітерації
/ / I згенероване число
$ I + +;
}
}
?>
```

У PHP існує одна особливість використання оператора continue - в конструкціях switch він працює так само, як і break. Якщо switch знаходиться всередині циклу й треба почати нову ітерацію циклу, слід використовувати continue 2.

6.4 Оператори включення

6.4.1 include

Оператор include дозволяє включати код, що міститься у вказаному файлі, і виконувати його стільки разів, скільки програма зустрічає цей оператор. Включення може здійснюватися будь-яким з перерахованих способів:

include 'ім'я_файлу';
include $ file_name;

include ("ім'я_файлу");

Приклад. Нехай у файлі params.inc у нас зберігається набір якихось параметрів і функцій. Кожного разу, коли нам потрібно буде використовувати ці параметри (функції), ми будемо додавати до тексту нашої основної програми команду include 'params.inc'.

```
params.inc
<? Php
$ User = "Вася";
$ Today = date ("d.m.y");
/ * Функція date () повертає дату
і час (тут - дату в форматі
день.місяць.рік) * /
?>

include.php <? Php
include ("params.inc");
/ * Змінні $ user і $ today задані у файлі
params.inc. Тут ми теж можемо ними
користуватися завдяки команді
include ("params.inc") * /

echo "Привіт, $ user! <br>";
/ / Виведе "Привіт, Вася!"
echo "Сьогодні $ today";
/ / Виведе, наприклад, "Сьогодні 24.01.11"
?>
```

Приклад. Використання оператора включення include

Зауважимо, що використання оператора include еквівалентно простий вставці змістовної частини файлу params.inc в код програми include.php. Може бути, тоді можна було в params.inc записати простий текст без жодних тегів, які вказують на те, що це php-код? Не можна! Справа в тому, що в момент вставки файлу відбувається перемикання з режиму обробки PHP в режим HTML. Тому код всередині включається у файл, який

потрібно обробити як PHP-скрипт, повинен бути укладений у відповідні теги.

Пошук файлу для вставки відбувається за такими правилами.

1. Спочатку ведеться пошук файлу в include_path щодо поточної робочої директорії.
2. Якщо файл не знайдений, то пошук виконується в include_path щодо директорії поточного скрипта.
3. Параметр include_path, визначається у файлі налаштувань PHP, задає імена директорій, в яких потрібно шукати файли, що включаються.

Наприклад, ваш include_path це. (Тобто поточна директорія), поточна робоча директорія це / www /. В основний файл include.php ви включаєте файл my_dir / a.php, який у свою чергу включає b.php. Тоді парсер насамперед шукає файл b.php в директорії / www /, і якщо такого немає, то в директорії / www / my_dir /.

Якщо файл включений з допомогою include, то що міститься в ньому код успадковує область видимості змінних рядки, де з'явився include. Будь-які змінні викликаного файлу будуть доступні в файлі з цього рядка і далі. Відповідно, якщо include з'являється всередині функції файла, який викликає, то код, що міститься в викликаному файлі, буде вести себе так, як ніби він був визначений всередині функції. Таким чином, він успадкує область видимості цієї функції. Хоча ми і не знайомилися ще з поняттям функції, все ж наводимо тут ці відомості в розрахунку на інтуїтивне його розуміння.

Приклад. Нехай файл для вставки params.inc залишиться таким же, а include.php буде наступним:

```
<? Php
function Footer () {
/ / Оголошуємо функцію з ім'ям Footer
include ("params.inc");
/ * Включаємо файл params.inc.
Тепер його змінними можна користуватися,
```

але тільки всередині функції * /
$ Str = "Сьогодні: $ today
";
$ Str .= "<a
href = 'mailto: help@intuit.ru'> Сторінку
створив $ user ";
echo "$ str";
}
Footer ();
/ / Викликаємо функцію Footer (). Отримаємо:
/ / Сьогодні: 08.07.05
/ / Сторінку створив Вася

echo "$ user, $ today";
/ / Виведе кому, так як
/ / Ці змінні видно тільки
/ / Всередині функції
?>

Приклад. Область видимості при використанні include

Крім локальних файлів, за допомогою include можна включати і зовнішні файли, вказуючи їх url-адреси. Дана можливість контролюється директивою url_fopen_wrappers у файлі налаштувань PHP і за замовчуванням, як правило, включена.

include () - це спеціальна мовна конструкція, тому при використанні всередині умовних блоків її потрібно укладати у фігурні дужки.

<? Php
/ * Це невірний запис. Отримаємо помилку.
Ми ж вставляємо не одну команду,
а декілька, вони тільки записані
в іншому файлі * /
if ($ condition) include ("first.php");
else include ("second.php");
/ / А ось так правильно.
if ($ condition) {include ("first.php");}
else {include ("second.php");}

?>

Приклад. Використання include ()

При використанні include можливо два види помилок - помилка вставки (наприклад, не можна знайти вказаний файл, невірно написана сама команда вставки тощо) або помилка виконання (якщо помилка міститься у файлі, що вставляється). У будь-якому випадку при помилці в команді include виконання скрипта не завершується.

6.4.2 require

Цей оператор діє приблизно так само, як і # include в C + +. Все, що ми говорили про include, лише за деякими винятками, справедливо і для require. require також дозволяє включати в програму і зробити який-небудь файл. Основна відмінність require і include полягає в тому, як вони реагують на виникнення помилки. Як вже говорилося, include видає попередження, і робота скрипта триває. Помилка в require викликає фатальну помилку роботи скрипта і припиняє його виконання.

Умовні оператори на require () не впливають. Хоча, якщо рядок, в якій з'являється цей оператор, не виконується, то жоден рядок коду з файлу теж не виконується. Цикли також не впливають на require (). Хоча код, що міститься у файлі, що вставляється, є об'єктом циклу, але вставка сама по собі відбувається тільки один раз.

У реалізаціях PHP до версії 4.0.2 використання require () означало, що інтерпретатор обов'язково спробує прочитати файл.

require, як і include, при використанні всередині умовних блоків потрібно укладати у фігурні дужки.

6.4.3 Альтернативний синтаксис

PHP пропонує альтернативний синтаксис для деяких своїх керуючих структур, а саме для if, while, for, foreach і switch. У кожному разі відкриваючу дужку потрібно замінити на двокрапку (:), а закриваючу - на endif;, endwhile, і т.д. відповідно.

Наприклад, синтаксис оператора if можна записати таким чином:

if (вираз): блок_виконання endif;

Сенс залишається тим же: якщо умова, що записана в круглих дужках оператора if, виявилась істиною, буде виконуватися весь код, від двокрапки ":" до команди endif;. Використання такого синтаксису корисно при вбудовуванні php в html-код.

```
<? Php
$ Names = array ("Іван", "Петро", "Семен");
if ($ names [0] == "Іван"):
?>
Привіт, Ваня!
<? Php
endif;?>
```

Прикла. Використання альтернативного синтаксису

Якщо використовуються конструкції else і elseif, то також можна задіяти альтернативний синтаксис:

```
<? Php
$ A = 1;
if ($ a == 5):
print "a одно 5";
print "...";
elseif ($ a == 6):
print "a дорівнює 6";
print "!!!";
else:
print "a не дорівнює ні 5, ні 6";
endif;
?>
```

6.5 Функції, визначені користувачем

Для чого потрібні функції? Щоб відповісти на це питання, потрібно зрозуміти, що взагалі являють собою функції. У програмуванні, як і в математиці, функція є відображення безлічі її аргументів на безліч її значень. Тобто функція для

кожного набору значень аргументу повертає якісь значення, що є результатом її роботи. Навіщо потрібні функції, спробуємо пояснити на прикладі. Класичний приклад функції у програмуванні - це функція, що обчислює значення факторіала числа. Тобто ми ставимо їй число, а вона повертає нам його факторіал. При цьому не потрібно для кожного числа, факторіал якого ми хочемо отримати, повторювати один і той самий код - досить просто викликати функцію з аргументом, рівним цьому числу.

Функція обчислення факторіала натурального числа

```
<? Php
function fact ($ n) {
if ($ n == 0) return 1;
else return $ fact = $ n * fact ($ n-1);
}
echo fact (3);
/ / Можна було б написати echo (3 * 2);
/ / Але якщо число велике,
echo fact (50);
/ / То зручніше користуватися функцією,
/ / Чим писати echo (50 * 49 * 48 *...* 3 * 2);
?>
```

Таким чином, коли ми здійснюємо дії, в яких простежується залежність від будь-яких даних, і при цьому, можливо, нам знадобиться виконувати такі ж дії, але з іншими вихідними даними, зручно використовувати механізм функцій - оформити блок дій у вигляді тіла функції, а змінні дані - як її параметр.

Подивимося, як у загальному вигляді виглядає завдання (оголошення) функції. Функція може бути визначена за допомогою наступного синтаксису:

```
function Імя_функції (параметр1, параметр2,
... параметрN) {
Блок_дій
return "значення повертається функцією";
}
```

Якщо прямо так написати в php-програмі, то працювати нічого не буде. По-перше, Імя_функції і імена параметрів функції (параметр1, параметр2 і т.д.) повинні відповідати правилам найменування в PHP (і російських символів у них краще не використовувати). Імена функцій нечутливі до регістру. По-друге, параметри функції - це змінні мови, тому перед назвою кожної з них повинен стояти знак $. Ніяких крапок ставити в списку параметрів не можна. По-третє, замість слів блок_дій в тілі функції повинен знаходитися будь-який правильний PHP-код (не обов'язково залежати від параметрів). І нарешті, після ключового слова return має йти коректний php-вираз (що-небудь, що має значення). Крім того, у функції може і не бути параметрів, як і значення, що повертається. Приклад правильного оголошення функції - функція обчислення факторіала, наведена вище.

Як відбувається виклик функції? Вказується ім'я функції і в круглих дужках список значень її параметрів, якщо такі є:

```
<? Php
Імя_функції ("значеніе_для_параметра1",
"Значеніе_для_параметра2 ",...);
/ / Приклад виклику функції - виклик функції
/ / Обчислення факторіала наведено вище,
/ / Там для обчислення факторіала числа 3
/ / Ми писали: fact (3);
/ / Де fact - ім'я викликається функції,
/ / А 3 - значення її параметра з ім'ям $ n
?>
```

Коли можна викликати функцію? Здавалося б, дивне запитання. Функцію можна викликати після її визначення, тобто в будь-якому рядку програми нижче блоку function f_name (){...}. У PHP3 це було дійсно так. Але вже в PHP4 такої вимоги немає. Вся справа в тому, як інтерпретатор обробляє одержуваний код. Єдиний виняток становлять функції, які визначаються умовно (всередині умовних операторів або інших

функцій). Коли функція визначається таким чином, її визначення повинно передувати її викликом.

```
<? Php
$ Make = true;
/ * Тут не можна викликати Make_event ();
тому що вона ще не існує, але можна
викликати Save_info () * /

Save_info ("Вася", "Іванов",
"Я вибрав курс по PHP");

if ($ make) {
/ / Визначення функції Make_event ()
function Make_event () {
echo "<p> Хочу вивчати Python <br>";
}
}
/ / Тепер можна викликати Make_event ()
Make_event ();
/ / Визначення функції Save_info
function Save_info ($ first, $ last, $ message) {
echo "<br> $ message <br>";
echo "Ім'я:". $ First. "". $ Last. "<br>";
}
Save_info ("Федя", "Федоров",
"А я вибрав Lisp");
/ / Save_info можна викликати і тут
?>
```

Приклад. Визначення функції всередині умовного оператора

Якщо функція одного разу визначена в програмі, то перевизначити або видалити її пізніше не можна. Незважаючи на те, що імена функцій нечутливі до регістру, краще викликати функцію з того ж імені, яким вона була задана у визначенні.

```
<? Php
/ * Не можна зберегти дані, тобто викликати
функцію DataSave () до того, як виконана
```

```
перевірка їх правильності, тобто викликана
функція DataCheck () * /

DataCheck ();
DataSave ();

function DataCheck () {
/ / Перевірка правильності даних
function DataSave () {
/ / Зберігаємо дані
}
}
?>
```

Приклад. Визначення функції усередині функції

Розглянемо докладніше аргументи функцій, їх призначення та використання.

6.5.1 Аргументи функцій

У кожній функції може бути список аргументів. За допомогою цих аргументів у функцію передається різна інформація (наприклад, значення числа, факторіал якого треба підрахувати). Кожен аргумент являє собою змінну або константу.

За допомогою аргументів дані у функцію можна передавати трьома різними способами. Це передача аргументів за значенням (використовується за замовчуванням), по посиланню й завдання значення аргументів за замовчуванням. Розглянемо ці способи докладніше.

Коли аргумент передається у функцію за значенням, зміна значення аргументу всередині функції не впливає на його значення поза функції. Щоб дозволити функції змінювати її аргументи, їх потрібно передавати по посиланню. Для цього у визначенні функції перед ім'ям аргументу слід написати знак амперсанд «&».

```
<? Php
/ / Напишемо функцію, яка б додавала
```

```
/ / К рядку слово checked
function add_label (& $ data_str) {
$ Data_str .= "checked";
}
$ Str = "<input type = radio name = article";
/ / Нехай є такий рядок
echo $ str. "><br>";
/ / Виведе елемент форми -
/ / Не зазначену радіо кнопку
add_label ($ str);
/ / Викличемо функцію
echo $ str. "><br>";
/ / Це виведе вже зазначену
/ / Радіо кнопку
?>
```

Приклад. Передача аргументів за посиланням

У функції можна визначати значення аргументів, які використовуються за замовчуванням. Саме значення за умовчанням має бути константним виразом, а не змінною і не представником класу або викликом іншої функції.

У нас є функція, яка створює інформаційне повідомлення, підпис до якого змінюється в залежності від значення переданого їй параметра. Якщо значення параметра не задано, то використовується підпис "Оргкомітет".

```
<? Php
function Message ($ sign = "Оргкомітет.") {
/ / Тут параметр sign має за замовчуванням значення "Оргкомітет"
echo "Наступне зібрання відбудеться завтра. <br>";
echo $ sign. "<br>";
}
Message ();
/ / Викликаємо функцію без параметра.
/ / У цьому випадку підпис - це Оргкомітет
Message ("З повагою, Вася");
/ / У цьому випадку підпис
```

/ / Буде "З повагою, Вася."
?>

Приклад. Значення аргументів за замовчуванням

Результатом роботи цього скрипта буде:

Наступне зібрання відбудеться завтра.
Оргкомітет.
Наступне зібрання відбудеться завтра.
З повагою, Вася.

Якщо у функції декілька параметрів, то ті аргументи, для яких задаються значення за замовчуванням, повинні бути записані після всіх інших аргументів у визначенні функції. В іншому випадку з'явиться помилка, якщо ці аргументи будуть опущені при виклику функції.

Наприклад, ми хочемо внести опис статті в каталог. Користувач повинен ввести такі характеристики статті, як її назва, автор та короткий опис. Якщо користувач не вводить ім'я автора статті, вважаємо, що це Іванов Іван.

```
<? Php
function Add_article ($ title, $ description,
$ Author = "Іванов Іван") {
echo "Заносимо в каталог статтю: $ title,";
echo "автор $ author";
echo "<br> Короткий опис:";
echo "$ description <hr>";
}
Add_article ("Інформатика і ми",
"Це стаття про інформатику ...",
"Петров Петро");
Add_article ("Хто такі хакери",
"Це стаття про хакерів ...");
?>
```

У результаті роботи скрипта одержимо наступне
Заносимо в каталог статтю: Інформатика і ми,
автор Петров Петро.

```
Короткий опис:
Це стаття про інформатику ...
Заносимо в каталог статтю: Хто такі хакери,
автор Іванов Іван.
Короткий опис:
Це стаття про хакерів ...

Якщо ж ми напишемо ось так:
<? Php
function Add_article ($ author = "Іванов Іван",
$ Title, $ description) {
/ / ... Дії як у попередньому прикладі
}
Add_article ("Хто такі хакери",
"Це стаття про хакерів ...");
?>

То в результаті отримаємо:
Warning: Missing argument 3 for
add_article () in
c: \ users \ nina \ tasks \ func \ def_bad.php
on line 2
```

6.5.2 Списки аргументів змінної довжини

В PHP4 можна створювати функції зі змінним числом аргументів. Тобто ми створюємо функцію, не знаючи заздалегідь, зі скількома аргументами її викличуть. Для написання такої функції ніякого спеціального синтаксису не потрібно. Все робиться за допомогою вбудованих функцій func_num_args (), func_get_arg (), func_get_args ().

Функція func_num_args () повертає число аргументів, переданих в поточну функцію. Ця функція може використовуватися тільки усередині визначення користувацької функції. Якщо вона з'явиться поза функцією, то інтерпретатор видасть попередження.

```
<? Php
```

```
function DataCheck () {
$ N = func_num_args ();
echo "Кількість аргументів функції $ n";
}
DataCheck ();
/ / Виведе рядок
/ / "Число аргументів функції 0"
DataCheck (1,2,3);
/ / Виведе рядок
/ / "Число аргументів функції 3"
?>
```

Приклад. Використання функції func_num_args ()

Функція func_get_arg (ціле номер_аргументу) повертає аргумент зі списку переданих у функцію аргументів, порядковий номер якого заданий параметром номер_аргумента. Аргументи функції вважаються починаючи з нуля. Як і func_num_args (), ця функція може використовуватися тільки усередині визначення який-небудь функції.

Номер_аргумента не може перевищувати число аргументів, переданих у функцію. Інакше буде згенероване попередження, і функція func_get_arg () поверне False.

Створимо функцію для перевірки типу даних її аргументів. Вважаємо, що перевірка пройшла успішно, якщо перший аргумент функції - ціле число, другий - рядок.

```
<?
function DataCheck () {
$ Check = true;
$ N = func_num_args ();
/ / Число аргументів,
/ / Переданих у функцію
/ * Перевіряємо, чи є першим
переданий аргумент цілим числом * /
if ($ n> = 1) if (! is_int (func_get_arg (0)))
$ Check = false;
/ * Перевіряємо, чи є друга
переданий аргумент рядком * /
```

```
if ($ n> = 2)
if (! is_string (func_get_arg (1)))
$ Check = false;
return $ check;
}

if (DataCheck (123, "text"))
echo "Перевірка пройшла успішно <br>";
else echo "Дані не задовольняють
умовам <br> ";
if (DataCheck (324))
echo "Перевірка пройшла успішно <br>";
else echo "Дані не задовольняють умовам <br>";
?>
```

Приклад. Функція для перевірки типу даних, її аргументів

Результатом роботи буде наступне.

Перевірка пройшла успішно

Перевірка пройшла успішно

Функція func_get_args () повертає масив, що складається зі списку аргументів, переданих функції. Кожен елемент масиву відповідає аргументу, переданому функції. Якщо функція використовується поза визначенням користувача функції, то генерується попередження.

Перепишемо попередній приклад, використовуючи цю функцію. Будемо перевіряти, чи є цілим числом кожен парний аргумент, переданий функції:

```
<?
function DataCheck () {
$ Check = true;
$ N = func_num_args ();
/ / Число аргументів,
/ / Переданих у функцію

$ Args = func_get_args ();
/ / Масив аргументів функції
for ($ i = 0; $ i <$ n; $ i + +) {
```

```
$ V = $ args [$ i];
if ($ i% 2 == 0) {
if (! is_int ($ v)) $ check = false;
/ / Перевіряємо,
/ / Чи є парний аргумент цілим
}
}
return $ check;
}
if (DataCheck (array ("text", 324)))
echo "Перевірка пройшла успішно <br>";
else echo "Дані не задовольняють
умовам <br> ";
?>
```

Як бачимо, комбінації функцій func_num_args (), func_get_arg () і func_get_args () використовується для того, щоб функції могли мати змінний список аргументів.

6.6 Використання змінних всередині функції

6.6.1 Глобальні змінні

Щоб використовувати всередині функції змінні, задані поза нею, ці змінні потрібно оголосити як глобальні. Для цього в тілі функції слід перерахувати їх імена після ключового слова global:

```
global $ var1, $ var2;
<?
$ A = 1;
function Test_g () {
global $ a;
$ A = $ a * 2;
echo 'в результаті роботи функції $ a =', $ a;
}
echo 'поза функції $ a =', $ a, ',';
Test_g ();
echo "<br>";
echo 'поза функції $ a =', $ a, ',';
Test_g ();
```

?>

Приклад. Глобальні змінні

У результаті роботи цього скрипта одержимо:

поза функції $ a = 1, в результаті роботи

функції $ a = 2

поза функції $ a = 2, в результаті роботи

функції $ a = 4

Коли змінна оголошується як глобальна, фактично створюється посилання на глобальну змінну. Тому такий запис еквівалентний наступному (масив $ GLOBALS містить всі змінні, глобальні щодо поточної області видимості):

$ Var1 = & $ GLOBALS ["var1"];

$ Var2 = & $ GLOBALS ["var2"];

Це означає, наприклад, що видалення змінної $ var1 не видаляє глобальної змінної $ GLOBALS ["var1"].

4.6.2 Статичні змінні

Щоб використовувати змінні тільки всередині функції, при цьому зберігаючи їх значення і після виходу з функції, потрібно оголосити ці змінні як статичні. Статичні змінні видно тільки всередині функції і не втрачають свого значення, якщо виконання програми виходить за межі функції. Оголошення таких змінних проводиться за допомогою ключового слова static:

static $ var1, $ var2;

Статичній змінній може бути присвоєно будь-яке значення, але не посилання.

```
<?
function Test_s () {
static $ a = 1;
/ / Не можна присвоювати вираз або посилання
$ A = $ a * 2;
echo $ a;
}
Test_s (); / / виведе 2
echo $ a; / / нічого не виведе, так як
```

```
/ / $ A доступна тільки
/ / Всередині функції
Test_s (); / / всередині функції $ a = 2, тому
/ / Результатом роботи функції
/ / Буде число 4
?>
```

Приклад. Використання статичної змінної

6.6.3 Значення, що повертаються

Всі функції, наведені вище в якості прикладів, виконували будь-які дії. Окрім подібних дій, будь-яка функція може повертати як результат своєї роботи якесь значення. Це робиться з допомогою return. Значення, що повертається може бути будь-якого типу, включаючи списки і об'єкти. Коли інтерпретатор зустрічає команду return у тілі функції, він негайно припиняє її виконання і переходить на той рядок, з якої була викликана функція.

Наприклад, складемо функцію, яка повертає вік людини. Якщо людина не померла, то вік вважається відносно поточного року.

```
<? Php
/ * Якщо другий параметр обчислюється
як true, то він розглядається як
дата смерті, * /

function Age ($ birth, $ is_dead) {
if ($ is_dead) return $ is_dead-$ birth;
else return date ("Y") - $ birth;
}
echo Age (1971, false); / / для 2009 року виведе 38
echo Age (1971, 2001); / / виведе 30
?>
```

У цьому прикладі можна було і не використовувати функцію return, а просто замінити її функцією виведення echo. Проте якщо ми все-таки робимо так, що функція повертає якесь

значення (у даному випадку вік людини), то в програмі ми можемо присвоїти будь-якій змінній значення цієї функції:

$ An_age = Age (1981, 2004);

У результаті роботи функції може бути повернуто лише одне значення. Кілька значень можна отримати, якщо повертати список значень (одновимірний масив). Припустимо, ми хочемо отримати повний вік людини з точністю до дня.

```
<? Php
function Full_age ($ b_day, $ b_month, $ b_year)
{
$ Y = date ("Y");
$ M = intval (date ("m"));
$ D = intval (date ("d"));
$ B_month = intval ($ b_month);
$ B_day = intval ($ b_day);
$ B_year = intval ($ b_year);

$ Day = ($ b_day> $ d? 30 - $ b_day + $ d: $ d - $ b_day);
$ TmpMonth = ($ b_day> $ d? -1: 0);
$ Month = ($ b_month> $ m + $ tmpMonth? 12 - $ b_month +
$ TmpMonth + $ m: $ m + $ tmpMonth - $ b_month);
$ TmpYear = ($ b_month> $ m + $ tmpMonth? -1: 0);
if ($ b_year> $ y + $ tmpYear)
{
$ Year = 0; $ month = 0; $ day = 0;
}
else
{
$ Year = $ y + $ tmpYear - $ b_year;
}
return array ($ day, $ month, $ year);
}
$ Age = Full_age ("29", "06", "1986");
echo "Вам $ age [2] років, $ age [1] місяців і $ age [0] днів";
?>
```

Коли функція повертає кілька значень для їх обробки в програмі, зручно використовувати мовну конструкцію list (), яка дозволяє однією дією присвоїти значення одразу кількома змінним. Наприклад, у попередньому прикладі, залишивши без зміни функцію, обробити повертати їй значення можна було так:

```
<?
/ / Завдання функції Full_age ()
list ($ day, $ month, $ year) = Full_age ("07",
"08", "1974");
echo "Вам $ year років, $ month місяців і
$ Day днів ";
?>
```

Взагалі конструкцію list () можна використовувати для присвоєння змінним значень елементів будь-якого масиву.

```
<?
$ Arr = array ("first", "second");
list ($ a, $ b) = $ arr;
/ / Змінної $ a присвоюється перший
/ / Значення масиву, $ b - друге
echo $ a, "", $ b;
/ / Виведе рядок «first second»
?>
```

Приклад. Використання list ()

6.6.4 Повернення посилання

У результаті своєї роботи функція також може повертати посилання на будь-яку змінну. Це може стати в нагоді, якщо потрібно використовувати функцію для того, щоб визначити, якій змінній повинно бути присвоєно посилання. Щоб отримати з функції посилання, потрібно при оголошенні перед її ім'ям написати знак амперсанд (&) і щоразу при виклику функції перед її ім'ям теж писати амперсанд (&). Зазвичай функція повертає посилання на будь-яку глобальну змінну (або її частина - посилання на елемент глобального масиву), посилання на статичну змінну (або її частина) або посилання на один з аргументів, якщо він був також переданий по посиланню.

```
<?
$ A = 3; $ b = 2;
function & ref ($ par) {
global $ a, $ b;
if ($ par% 2 == 0) return $ b;
else return $ a;
}
$ Var = & ref (4);
echo $ var, "i", $ b, "<br>";
/ / Виведе 2 і 2
$ B = 10;
echo $ var, "i", $ b, "<br>";
/ / Виведе 10 і 10
?>
```

Приклад. Повернення посилання

При використанні синтаксису посилань в змінну $ var нашого прикладу не копіюється значення змінної $ b повернутої функцією $ ref, а створюється посилання на цю змінну. Тобто тепер змінні $ var і $ b ідентичні і будуть змінюватися одночасно.

6.6.5 Змінні функції

PHP підтримує концепцію змінних функцій. Це означає, що якщо ім'я змінної закінчується круглими дужками, то PHP шукає функцію з таким же ім'ям і намагається її виконати.

```
<?
/ * Створимо дві прості функції:
Add_sign - додає підпис до рядка і
Show_text - виводить рядок тексту * /

function Add_sign ($ string,
$ Sign = "З повагою, Петро") {
echo $ string. "". $ sign;
}
function Show_text () {
echo "Відправити повідомлення поштою <br>";
```

```
}
$ Func = "Show_text";
/ / Створюємо змінну зі значенням,
/ / Рівним імені функції Show_text
$ Func ();
/ / Це викличе функцію Show_text
$ Func = "Add_sign";
/ / Створюємо змінну зі значенням,
/ / Рівним імені функції Add_sign
$ Func ("Привіт всім <br>");
/ / Це викличе функцію
/ / Add_sign з параметром "Привіт усім"
?>
```

Приклад. Використання змінних функцій

У цьому прикладі функція Show_text просто виводить рядок тексту. Здавалося б, навіщо для цього створювати окрему функцію, якщо існує спеціальна функція echo (). Справа в тому, що такі функції, як echo (), print (), unset (), include () і т.п. не можна використовувати в якості змінних функцій. Тобто якщо ми напишемо:

```
<?
$ Func = "echo";
$ Func ("TEXT");
?>
```

то інтерпретатор виведе помилку:

```
Fatal error: Call to undefined function:
echo () in
c: \ users \ nina \ tasks \ func \ var_f.php on line 2
```

Тому для того, щоб використовувати будь-яку з перерахованих вище функцій як змінну функцію, потрібно створити власну функцію, що ми і зробили в попередньому прикладі.

6.7 Символічні і жорсткі посилання

Хоча в PHP немає такого поняття, як покажчик, все ж таки існує можливість створювати посилання на інші змінні. Існує

два різновиди посилань: жорсткі і символічні (змінні) (перші часто називають просто посиланнями). Жорсткі посилання з'явилися в PHP версії 4 (в третій версії існували лише символічні посилання).

Посилання в PHP - це засіб доступу до вмісту однієї змінної під різними іменами. Вони не схожі на покажчики мови Сі і не є псевдонімами таблиці символів. У PHP ім'я змінної і її вміст - це різні речі, тому один вміст може мати різні імена. Найближча аналогія - імена файлів Unix і медіа - імена змінних є елементами каталогів, а вміст змінних це самі файли. Посилання в PHP - аналог жорстких посилань (hardlinks) у файлових системах Unix.

6.7.1 Жорсткі посилання в PHP

Жорстке посилання представляє собою просто змінну, яка є синонімом іншої змінної. Багаторівневі посилання (тобто, посилання на посилання на змінну, як це можна робити, наприклад, у Perl) не підтримуються. Так що не варто сприймати жорсткі посилання серйозніше, ніж синоніми.

Щоб створити жорстку посилання, потрібно використовувати оператор & (амперсанд). Наприклад:

```
$ A = 10;
$ B = & $ a; / / тепер $ b - те ж саме, що і $ a
$ B = 0; / / насправді $ a = 0
echo "b = $ b, a = $ a"; / / Виводить: "b = 0, a = 0"
```

Посилатися можна не тільки на змінні, але й на елементи масиву (цим жорсткі посилання вигідно відрізняються від символічних). Наприклад:

```
$ A = array ('a' => 'aaa', 'b' => 'bbb');
$ B = & $ A ['b']; / / тепер $ b - те ж, що й елемент з індексом 'b' масиву
$ B = 0; / / насправді $ A ['b'] = 0;
echo $ A ['b']; / / Виводить 0
```

Втім, елемент масиву, для якого планується створити символічне посилання, може і не існувати. Як у наступному випадку:

$ A = array ('a' => 'aaa', 'b' => 'bbb');

$ B = & $ A ['c']; / / тепер $ b - те ж, що й елемент з індексом 'c' масиву

echo "Елемент з індексом 'c': (". $ A ['c']. ")";

В результаті виконання розглянутого скрипта, хоча посиланням $ b і не було нічого присвоєно, в масиві $ A створиться новий елемент з ключем c і значенням - порожній рядком (ми можемо це визначити за результатом роботи echo). Тобто, жорстка посилання на самому ділі не може посилатися на неіснуючий об'єкт, а якщо робиться така спроба, то об'єкт створюється.

Примітка: Якщо прибрати рядок, в якій створюється жорстке посилання, то буде виведено повідомлення про те, що елемент з ключем c не визначено в масиві $ A.

Жорсткі посилання зручно застосовувати при передачі параметрів для користувача функції і повернення значення з неї.

6.7.2 Символічні посилання (змінні змінні)

Символічна посилання - це всього лише строкова змінна, що зберігає ім'я іншої змінної. Щоб дістатися до значення змінної, на яку посилається символічне посилання, необхідно застосувати додатковий знак $ перед ім'ям посилання. Розглянемо приклад:

$ A = 10;

$ B = 20;

$ C = 30;

$ P = "a"; / / або $ p = "b" або $ p = "c" (присвоюємо $ p ім'я іншої змінної)

echo $ $ p; / / виводить змінну, на яку посилається $ p, тобто $ a

$ $ P = 100; / / присвоює $ a значення 100

Ми бачимо, що для того, щоб використовувати звичайну строкову змінну як посилання, потрібно перед нею поставити ще один символ $. Це говорить інтерпретатору, що треба взяти не значення самої $ p, а значення змінної, ім'я якої зберігається у змінній $ p.

Символічні посилання (змінні змінні) використовуються досить рідко.

6.7.3 Жорсткі посилання і призначені для користувача функції

Передача значень за посиланням

Ви можете передавати змінні в призначену для користувача функцію по посиланню, якщо ви хочете дозволити функції модифікувати свої аргументи. У такому випадку, призначені для користувача функція зможе змінювати аргументи. Синтаксис такий:

```
<? Php
function foo (& $ var)
{
$ Var + +;
}

$ A = 5;
foo ($ a);
/ / $ A тут дорівнює 6
?>
```

Зауважте, що у виклику функції відсутній знак посилання - він є тільки у визначенні функції. Цього достатньо для коректної передачі аргументів за посиланням.

Ще один цікавий приклад:

```
<? Php
function funct (& $ string)
{
$ String .= 'а ця всередині.';
}
$ Str = 'Цей рядок за межами функції,';
funct ($ str);
echo $ str; / / Виведе 'Цей рядок за межами функції, а ця всередині.'
?>
```

За посиланням можна передавати:

- Змінні, наприклад foo ($ a)
- Оператор new, наприклад foo (new foobar ())
- Лінки, які повертаються функцією, наприклад:

```
<? Php
function & bar ()
{
$ A = 5;
return $ a;
}
foo (bar ());
?>
```

Будь-яке інше вираження не повинно передаватися по посиланню, так як результат не визначений. Наприклад, наступна передача за посиланням є неправильною:

```
<? Php
function bar () / / Операція & відсутня
{
$ A = 5;
return $ a;
}
foo (bar ());

foo ($ a = 5); / / Вираз, а не змінна
foo (5); / / Константа, а не змінна
?>
```

Повернення значень за посиланням

Розглянемо ще одну можливість для користувача функцій PHP - повернення посилань.

Повернення за посиланням використовується в тих випадках, коли ви хочете використовувати функцію для вибору змінної, з якою має бути пов'язана дане посилання. При поверненні за посиланням використовуйте такий синтаксис:

```
<? Php
```

```
function & find_var ($ param)
{
/ * ... код ... * /
return $ found_var;
}

$ Foo = & find_var ($ bar);
$ Foo-> x = 2;
?>
```

У цьому прикладі встановлюється властивість об'єкта, повернутого функцією find_var, а не його копії, як було б без використання посилань.

Ще один приклад повернення значень користувача функції за посиланням:

```
<? Php
$ A = 100;
/ * Далі йде функція, яка повертає посилання * /
function & s () {
global $ a;
/ / Повертаємо посилання на змінну $ a
return $ a;
}
/ / Надаємо посилання змінної $ b
$ B = & s ();
$ B = 0;
echo $ a; / / Виводить 0
?>
```

6.7.4 Видалення посилань (скидання посилань)

При видаленні посилання, просто розривається зв'язок імені та вмісту змінної. Це не означає, що вміст змінної буде зруйновано. Наприклад:

```
<? Php
$ A = 1;
$ B = & $ a;
unset ($ a);
```

?>

Цей код не скине $ b, а тільки $ a.

І все ж, жорстке посилання - не абсолютно точний синонім об'єкта, на який вона посилається. Справа в тому, що оператор Unset (), виконаний для жорсткого посилання, не видаляє об'єкт, на який він посилається, а всього лише розриває зв'язок між посиланням і об'єктом.

Отже, жорстке посилання і змінна (об'єкт), на яку він посилається, абсолютно рівноправні, але зміна однієї тягне зміну іншої. Оператор Unset () розриває зв'язок між об'єктом і посиланням, але об'єкт віддаляється тільки тоді, коли на нього ніхто вже не посилається.

Тема 7 Обробка масивів даних

Мова PHP надає безліч функцій для роботи з масивами даних. Як правило, ці функції найбільш часто зустрічаються у задачах, пов'язаних з обробкою масивів.

7.1 Створення масивів

Масив можна створити **двома способами:**

1. За допомогою конструкції array
   $ Array_name = array ("key1" => "value1",
   "Key2" => "value2");
2. Безпосередньо задаючи значення елементів масиву
   $ Array_name ["key1"] = value1;

Наприклад, нам потрібно зберігати список документів, які будуть видалені з бази даних. Природно зберігати його у вигляді масиву, ключем в якому буде ідентифікатор документа (його унікальний номер), а значенням - назва документа. Цей масив можна створити таким чином:

```
<?
$ Del_items = array ("10" => "Наука і життя",
"12" => "Інформатика");
$ Del_items ["13"] = "Програмування на Php";
/ / Додаємо елемент в масив
?>
```

7.2 Операції з масивами

Масив - це тип даних, з даними цього типу повинні бути визначені операції. Які ж операції можна проводити з масивами? Масиви можна складати і порівнювати.

Складають масиви за допомогою стандартного оператора «+». Взагалі кажучи, цю операцію по відношенню до масивів точніше назвати об'єднанням. Якщо у нас є два масиви, $ a та $ b, то результатом їх складання (об'єднання) буде масив $ c, що складається з елементів $ a, до яких праворуч дописані елементи масиву $ b. Причому, якщо зустрічаються збігаються ключі, то в результуючий масив включається елемент з першого масиву,

тобто з $ a. Таким чином, якщо складаються масиви в мові PHP, то від зміни місць доданків сума змінюється.

```
<?
$ A = array ("i" => "Інформатика",
"M" => "Математика");
$ B = array ("i" => "Історія", "м" => "Біологія",
"Ф" => "Фізика");
$ C = $ a + $ b;
$ D = $ b + $ a;
print_r ($ c);
/ * Отримаємо: Array ([i] => Інформатика
[M] => Математика [ф] => Фізика) * /
print_r ($ d);
/ * Отримаємо: Array ([i] => Історія
[M] => Біологія [ф] => Фізика) * /
?>
```

Приклад. Додавання масивів

Порівнювати масиви можна, перевіряючи їх рівність чи нерівність або еквівалентність або нееквівалентність. Рівність масивів - це коли співпадають всі пари ключа / значення елементів масивів. Еквівалентність - коли крім рівності значень і ключів елементів потрібно ще, щоб елементи в обох масивах були записані в одному і тому ж порядку. Рівність значень в PHP позначається символом «==», а еквівалентність - символом «===».

```
<?
$ A = array ("i" => "Інформатика",
"M" => "Математика");
$ B = array ("м" => "Математика",
"I" => "Інформатика");
if ($ a == $ b) echo "Масиви рівні і";
elseecho "Масиви НЕ рівні і";
if ($ a === $ b) echo "еквівалентні";
elseecho "НЕ еквівалентні";
/ / Отримаємо echo "Масиви рівні і НЕ еквівалентні "
```

?>

Приклад. Порівняння масивів

Далі розглянемо ще одну важливу операцію з масивом - підрахунок кількості його елементів. Для її реалізації в PHP є спеціальна функція.

7.2.1 Функція count

Не раз вже ми використовували функцію count (), щоб обчислити кількість елементів масиву. Насправді ця функція обчислює число елементів у змінній взагалі. Якщо застосувати її до будь-якої іншої змінної, вона поверне 1. Виняток становить змінна типу NULL - count (NULL) є 0. Крім того, застосовуючи цю функцію до багатовимірного масиву, щоб отримати число його елементів, потрібно використовувати додатковий параметр COUNT_RECURSIVE.

```
<?
$ Del_items = array ("langs" =>array (
"10" => "Python", "12" => "Lisp"),
"Other" => "Інформатика");
echocount ($ del_items). "
";
/ / Виведе 2
echocount ($ del_items, COUNT_RECURSIVE);
/ / Виведе 4
?>
```

Приклад. Застосування функції count ()

7.2.2 Функція in_array

```
in_array ("шукане значення", "масив",
["Обмеження на тип"]);
```

дозволяє встановити, чи міститься у заданому масиві шукане значення. Якщо третій аргумент заданий як true, то в масиві потрібно знайти елемент, що співпадає з шуканим не тільки за

значенням, але і по типу. Якщо шукане значення - рядок, то порівняння чутливе до регістру.

Наприклад, є масив не вивчених нами мов програмування. Ми хочемо дізнатися, чи міститься в цьому масиві мова PHP. Напишемо наступну програму:

```
<? Php
$ Langs = array ("Lisp", "Python", "Java",
"PHP", "Perl");
if (in_array ("PHP", $ langs, true))
echo "Треба б вивчити PHP <br>";
/ / Виведе повідомлення "Треба б вивчити PHP"
if (in_array ("php", $ langs))
echo "Треба б вивчити php<br>";
/ / Нічого не виведе, оскільки в масиві
/ / Є рядок "PHP", а не "php"
?>
```

В якості шуканого значення цієї функції може виступати і масив.

Наприклад:

```
<? Php
$ Langs = array ("Lisp", "Python", array ("PHP", "Java"), "Perl");
if (in_array (array ("PHP", "Java"), $ langs))
echo "Треба б вивчити PHP і Java<br>";
?>
```

7.2.3 Функція array_search

Це ще одна функція для пошуку значення в масиві. На відміну від in_array в результаті роботи **array_search**повертає значення ключа, якщо елемент знайдено, і брехня - в іншому випадку. А ось синтаксис у цих функцій однаковий:

```
array_search ("шукане значення", "масив",
["Обмеження на тип"]);
```

Порівняння рядків чутливо до регістру, а якщо вказаний опціональний аргумент, то рівняються ще й типи значень.

Приклад. Тепер, навпаки, нехай у нас є масив мов програмування, які ми знаємо. Причому ключем кожного елемента є номер, що вказує, яким за рахунком було вивчено цю мову.

```
<? Php
$ Langs = array ("," Lisp "," Python "," Java ",
"PHP", "Perl");
if (! array_search ("PHP", $ langs))
echo "Треба б вивчити PHP <br>";
else {
$ K = array_search ("PHP", $ langs);
echo "PHP я вивчила $ k-м";
}
?>
```

Прикла. Застосування функції array_search ()

У результаті ми отримаємо рядок:

PHP я вивчила 4-м

Очевидно, що ця функція більш функціональна, ніж in_array, оскільки ми не тільки отримуємо інформацію про те, що шуканий елемент у масиві є, але і дізнаємося, де саме в масиві він знаходиться. А що буде, якщо шуканих елементів у масиві декілька? У такому випадку функція array_search () поверне ключ першого зі знайдених елементів. Щоб отримати ключі всіх елементів, потрібно скористатися функцією array_keys ().

7.2.4 Функція array_keys

Функція **array_keys ()** вибирає всі ключі масиву. Але в неї є додатковий аргумент, за допомогою якого можна отримати список ключів елементів з конкретним значенням. Синтаксис цієї функції такий:

```
array_keys ("масив",
["Значення для пошуку"])
```

Функція **array_keys ()** повертає як рядкові, так і числові ключі масиву, організовуючи всі значення у вигляді нового масиву з числовими індексами.

Приклад. Ми записали масив мов, які вивчили. Список був довгим, і деякі мови були записані декілька разів. У нас виникла підозра, що один з таких мов - Lisp. Давайте це перевіримо:

```
<? Php
$ Langs =
array ("Lisp", "Python", "Java", "PHP",
"Perl", "Lisp");
$ Lisp_keys = array_keys ($ langs, "Lisp");
echo "Lisp входить в масив".
count ($ lisp_keys). "рази: <br>";
foreach ($ lisp_keysas $ val) {
echo "під номером $ val<br>";
}
?>
```

Приклад. Застосування функції array_keys ()

У результаті отримаємо:

```
Lisp входить в масив 2 рази:
під номером 0
під номером 5
```

Функція **array_keys ()**, як і дві попередні, залежать від регістра, тобто елементів LISP в масиві вона не виявить. array_keys () з'явилася лише в PHP4. У PHP3 для реалізації її функціональності потрібно придумувати свою функцію.

Якщо є функція для отримання всіх ключів масиву, то можна припустити, що існує і функція для отримання всіх значень масиву. Дійсно, вона існує. Це функція **array_values (масив)**. Всі значення переданого їй масиву записуються в новий масив, проіндексований цілими числами, тобто всі ключі масиву губляться, залишаються лише значення. Але повернемося до нашого прикладу.

Отже, ми з'ясували, що мова Lisp випадково згаданий в нашому масиві двічі. Оскільки вивчити одну мову двічі не

можна («вивчав, але забув» не береться до уваги), то потрібно якось позбутися від повторюваних мов. Зробити це досить просто за допомогою функції array_unique ().

7.2.5 Функція array_unique

Функція array_unique (масив) повертає новий масив в якому елементи, що повторюються фігурують в одному екземплярі. Таким чином, замість кількох однакових значень і їх ключів ми маємо одне значення. Який у нього буде ключ? Як з декількох ключів однакових елементів вибирається той, який буде збережений в новому масиві? Відбувається наступне. Всі елементи масиву перетворюються у рядки та сортуються. Потім обробник запам'ятовує перший ключ для кожного значення, а решту ключів ігнорує.

Спробуємо позбутися від повторюваних мов у списку вивчених.

```
<? Php
$ Langs =
array ("Lisp", "Java", "Python", "Java",
"PHP", "Perl", "Lisp");
print_r (array_unique ($ langs));
?>
```

Отримаємо наступне:

```
Array ([0] =>Lisp [1] =>Java [2] =>Python [3]
=> PHP [4] =>Perl)
```

Далі розглянемо задачу сортування масиву.

7.3 Сортування масивів

Необхідність сортування даних, в тому числі й даних, що зберігаються у вигляді масивів, дуже часто виникає при вирішенні найрізноманітніших завдань. Якщо мові Сі для того, щоб вирішити цю задачу, потрібно написати десятки рядків коду, то в PHP це робиться однією простою командою.

7.3.1 Функція sort

Функція sort має наступний синтаксис

sort (масив [, прапори])

і сортує масив, тобто впорядковує його значення за зростанням. Ця функція видаляє всі існуючі в масиві ключі, замінюючи їх числовими індексами, відповідно новим порядком елементів. У разі успішного завершення роботи вона повертає true, інакше - false.

Приклад. Нехай у нас є два масиви: ціна товарів - їх назви і, навпаки, назви товарів - їх ціна. Впорядкуємо ці масиви за зростанням:

```
<?
$ Items = array (10 => "хліб", 20 => "молоко",
30 => "бутерброд");
sort ($ items);
/ / Рядки сортуються в алфавітному
/ / Порядок, ключі губляться
print_r ($ items);

$ Rev_items = array ("хліб" => 10,
"Бутерброд" => 30, "молоко" => 20);
sort ($ rev_items);
/ / Числа сортуються за зростанням,
/ / Ключі губляться
print_r ($ rev_items);
?>
```

Приклад. Застосування функції sort ()

Отримаємо:

```
Array ([0] => бутерброд [1] =>
молоко [2] => хліб)
Array ([0] => 10 [1] => 20 [2] => 30)
```

В якості додаткового аргументу можна використовувати одне з наступних констант:

- SORT_REGULAR - порівнювати елементи масиву звичайним чином;
- SORT_NUMERIC - порівнювати елементи масиву як числа;
- SORT_STRING - порівнювати елементи масиву як рядки.

7.3.2 Функції asort, rsort, arsort

Якщо потрібно зберігати індекси елементів масиву після сортування, то потрібно використовувати функцію asort (масив [, прапори]). Якщо необхідно відсортувати масив у зворотному порядку, тобто від найбільшого значення до найменшого, то можна задіяти функцію rsort (масив [, прапори]). А якщо при цьому потрібно ще й зберегти значення ключів, то слід використовувати функцію arsort (масив [, прапори]). Як ви, напевно, помітили синтаксис у цих функцій абсолютно такий же, як у функції sort. Відповідно і значення прапорів можуть бути такими ж, як у sort: SORT_REGULAR, SORT_NUMERIC, SORT_STRING. До речі кажучи, прапорецьSORT_NUMERIC з'явився тільки в PHP4.

```
<? Php
$ Books = array ("Пушкін" => "Руслан і Людмила",
"Толстой" => "Війна і мир",
"Лермонтов" => "Герой нашого часу");
asort ($ books);
/ / Сортуємо масив,
/ / Зберігаючи значення ключів
print_r ($ books);
echo "<br>";
rsort ($ books);
/ / Сортуємо масив у зворотному порядку,
/ / Ключі будуть замінені
print_r ($ books);
?>
```

Приклад. Застосування функцій asort, rsort, arsort

У результаті роботи цього скрипта одержимо:

```
Array ([Толстой] => Війна і мир
[Лермонтов] => Герой нашого часу
[Пушкін] => Руслан і Людмила)
Array ([0] => Руслан і Людмила
[1] => Герой нашого часу
[2] => Війна і мир)
```

Приклад. Припустимо, ми створюємо каталог описів документів. У кожного документа є автор, назва, дата публікації і короткий зміст. Ми вже не раз відображали описи, складені з цих характеристик. Кожного разу порядок відображення цих елементів залежав від створеної нами програми. Тепер же ми хочемо мати можливість змінювати порядок відображення елементів за бажанням користувача. Складемо для цього наступну форму:

```
<formaction=task.php>
<tableborder=1>
<tr><td> Назва </ td><td><input type = text
name = titlesize = 5></ td></ tr>
<tr><td> Короткий зміст </ td><td><input
type = text name = descriptionsize = 5>
</ Td></ tr>
<tr><td> Автор </ td><td><input type = text
name = authorsize = 5></ td></ tr>
<tr><td> Дата публікації </ td><td><input
type = text name = publishedsize = 5></ td></ tr>
</ Table>
<input type=submit value="Відправте">
</ Form>
```

Приклад. Форма

Будемо упорядковувати дані, передані цією формою, за зменшенням їх значень, зберігаючи при цьому значення ключів. Для цього зручно скористатися функцією arsort (). Оскільки нам важливий тільки новий порядок елементів, збережемо у новому масиві ключі вихідного масиву в потрібному порядку. Ми зберігаємо ключі вихідного масиву, оскільки вони є іменами елементів, з яких конструюється опис документа, а пам'ятати їх важливо. Отже, отримуємо такий скрипт:

```
<? Php
print_r ($ _GET); echo "<br>";
arsort ($ _GET);
/ / Сортуємо масив у зворотному порядку,
```

/ / Зберігаючи ключі
print_r ($ _GET); echo "
";
$ Ordered_names = array_keys ($ _GET);
/ / Складаємо новий масив
foreach ($ ordered_namesas $ key => $ val)
echo "$ key: $ val
";
/ / Виводимо елементи нового масиву
?>

Приклад . Програма обробки форми

7.3.3 Сортування масиву по ключах

Очевидно, що може виникнути необхідність у сортуванні масиву за значеннями ключів. Наприклад, якщо у нас є масив даних про книжки, як у наведеному вище прикладі, то цілком ймовірно, що ми захочемо відсортувати книги по іменах авторів. Для цього в PHP також не потрібно писати багато рядків коду - можна просто скористатися функцією ksort () для сортування за зростанням (прямий порядок сортування) або krsort () - для сортування за спаданням (зворотний порядок сортування). Синтаксис цих функцій знову ж аналогічний синтаксису функції sort ().

<? Php
$ Books = array ("Пушкін" => "Руслан і Людмила",
"Толстой" => "Війна і мир",
"Лермонтов" => "Герой нашого часу");
ksort ($ books);
/ / Сортуємо масив,
/ / Зберігаючи значення ключів
print_r ($ books);
?>

Приклад. Сортування масиву по ключах

Отримаємо:
Array ([Лермонтов] => Герой нашого часу
[Пушкін] => Руслан і Людмила
[Толстой] => Війна і мир)

7.3.4 Сортування за допомогою функції, заданої користувачем

Крім двох простих способів сортування значень масиву (за спаданням або за зростанням) PHP пропонує користувачеві можливість самому задавати критерії для сортування даних. Критерій задається за допомогою функції, ім'я якої вказується в якості аргументу для спеціальних функцій сортування usort () або uksort (). За назвами цих функцій можна здогадатися, що usort () сортує значення елементів масиву, а uksort () - значення ключів масиву за допомогою певної функції користувача. Обидві функції повертають true, якщо сортування пройшла успішно, і false - в протилежному випадку. Їх синтаксис виглядає наступним чином:

usort (масив, сортується функція)

uksort (масив, сортується функція)

Звичайно ж, не можна сортувати масив за допомогою будь-якої функції користувача. Ця функція повинна задовольнятись певним критеріям, що дозволить порівнювати елементи масиву. Як повинна бути влаштована функція сортування? По-перше, вона повинна мати два аргументи. У них інтерпретатор буде передавати пари значень елементів для функції usort () або ключів масиву для функції uksort (). По-друге, функція, що сортується повинна повертати:

- ціле число, менше нуля, якщо перший аргумент менше другого;
- число, рівне нулю, якщо два аргументи рівні;
- число більше за нуль, якщо перший аргумент більше другого.

Як і для інших функцій сортування, для функції usort () існує аналог, що не змінює значення ключів, - функція uasort ().

Приклад. Припустимо, у нас є масив, який містить такі відомості про літературні твори, як назва, автор і рік створення. Ми хочемо впорядкувати книги за датою створення.

<? Php

/ / Масив виглядає таким чином:

```
$ Books = array ("Герой нашого часу" =>
array ("Лермонтов", 1840),
"Руслан і Людмила" =>array ("Пушкін", 1820),
"Війна і мир" =>array ("Толстой", 1863),
"Ідіот" =>array ("Достоєвський", 1868));
/ * Можна, звичайно переписати цей масив
по-іншому, зробивши рік видання, наприклад,
індексом, але набагато зручніше написати свою
функцію для сортування * /

uasort ($ books, "cmp");
/ / Сортуємо масив за допомогою функції cmp

foreach ($ booksas $ key => $ book) {
echo "$ book [0]: \" $ key \ "<br>";
}
functioncmp ($ a, $ b) {
/ / Функція, що визначає спосіб сортування
if ($ a [1] <$ b [1]) return -1;
elseif ($ a [1] == $ b [1]) return 0;
elsereturn 1;
}
?>
```

Приклад. Сортування за допомогою призначених для користувача функцій

У результаті отримаємо:

Пушкін: "Руслан і Людмила"

Лермонтов: "Герой нашого часу"

Толстой: "Війна і мир"

Достоєвський: "Ідіот"

Ми застосували нашу власну функцію сортування до всіх елементів масиву. Далі розглянемо, як застосувати до елементів масиву будь-яку іншу функцію користувача.

7.4 Застосування функції до всіх елементів масиву

Функція **array_walk** (масив, функція [, дані]) застосовує створену користувачем функцію до всіх елементів масиву масив і повертає true у разі успішного виконання операції і false - в протилежному випадку.

Функція користувача, як правило, має два аргументи, у які по черзі передаються значення і ключ кожного елемента масиву. Але якщо при виконанні функції array_walk () вказано третій аргумент, то він буде розглянутий, як значення третього аргументу для функції користувача, зміст якого визначає сам користувач. Якщо функція користувача потребує більше аргументів, ніж у неї передано, то при кожному виклику **array_walk ()** буде видаватися попередження.

Якщо необхідно працювати з реальними значеннями масиву, а не з їх копіями, слід передавати аргумент на функцію по посиланню. Однак потрібно мати на увазі, що не можна додавати або видаляти елементи масиву і проводити дії, що змінюють сам масив, оскільки в цьому випадку результат роботи array_walk () вважається невизначеним.

```
<? Php
$ Books1 = array (
"А. С. Пушкін" => "Руслан і Людмила",
"Л. Н. Толстой" => "Війна і мир",
"М. Ю. Лермонтов" => "Герой нашого часу");
/ / Створюємо функцію, яку хочемо
/ / Застосувати до елементів масиву

functiontry_walk ($ val, $ key, $ data) {
echo "$ data \" $ val \ "написав $ key<br>";
}
/ / Застосовуємо до всіх елементів масиву
/ / $ Books1 функцію try_walk
array_walk ($ books1, "try_walk", "Роман");
?>
```

Приклад. Застосування функції до всіх елементів масиву

У результаті роботи скрипта одержимо:

Роман "Руслан і Людмила" написав А.С. Пушкін
Роман "Війна і мир" написав Л.М. Толстой
Роман "Герой нашого часу" написав М.Ю. Лермонтов

Зауважимо, що ми не змінили значень у елементах масиву. Щоб їх змінити, треба було передавати значення в змінну $ val функції try_walk по посиланню.

```
<? Php
$ Books1 = array (
"А. С. Пушкін" => "Руслан і Людмила",
"Л. Н. Толстой" => "Війна і мир",
"М. Ю. Лермонтов" => "Герой нашого часу");
/ / Створюємо функцію, яку хочемо
/ / Застосувати до елементів масиву

functiontry_walk (& $ val, $ key) {
$ Key = "<p> Автор:". $ Key. "<br>";
$ Val = "Назва: \" ". $ Val." \ "";
echo $ key. $ val;
}
/ / Застосовуємо до всіх елементів масиву
/ / $ Book1 функцію try_walk

array_walk ($ books1, "try_walk");
print_r ($ books1);
?>
```

Приклад. Застосування функції до всіх елементів масиву. Варіант 2

У результаті роботи скрипта одержимо:

Автор: А.С. Пушкін
Назва: "Руслан і Людмила"
Автор: Л.М. Толстой
Назва: "Війна і мир"
Автор: М.Ю. Лермонтов
Назва: "Герой нашого часу"
Array ([А. С. Пушкін] =>

Назва: "Руслан і Людмила"
[Л.М. Толстой] =>
Назва: "Війна і мир"
[М.Ю. Лермонтов] =>
Назва: "Герой нашого часу")

7.5 Виділення підмасивів

7.5.1 Функція array_slice

Оскільки масив - це набір елементів, цілком ймовірно, буде потрібно виділити з нього який-небудь піднабір. У PHP для цих цілей є функція array_slice. Її синтаксис такий:

array_slice (масив,
номер_елемента [, довжина])

Ця функція виділяє підмасив довжини довжина в масиві масив, починаючи з елемента, номер якого заданий параметром номер_елемента. Позитивний номер_елемента вказує на порядковий номер елемента щодо початку масиву, негативний - на номер елемента з кінця масиву.

```
<? Php
$ Arr = array (1,2,3,4,5);
$ Sub_arr = array_slice ($ arr, 2);
print_r ($ sub_arr);
/ * виведе Array ([0] => 3 [1] => 4 [2] => 5), тобто підмасив, що складається з елементів 3, 4, 5 * /
$ Sub_arr = array_slice ($ arr, -2);
print_r ($ sub_arr);
/ / Виведе Array ([0] => 4 [1] => 5),
/ / Тобто підмасив, з елементів 4, 5
?>
```

Приклад. Використання функції array_slice ()

Якщо задати параметр довжина при використанні array_slice, то буде виділений підмасив, що має рівно стільки елементів, скільки задано цим параметром. Довжину можна вказувати і

негативну. У цьому випадку інтерпретатор видалить з кінця масиву число елементів, рівне модулю параметра довжина.

```
<? Php
$ Arr = array (1,2,3,4,5);
$ Sub_arr = array_slice ($ arr, 2, 2);
/ / Містить масив з елементів 3, 4
$ Sub = array_slice ($ arr, -3, 2);
/ / Теж містить масив з елементів 3, 4
$ Sub1 = array_slice ($ arr, 0, -1);
/ / Містить масив з
/ / Елементів 1, 2, 3, 4
$ Sub2 = array_slice ($ arr, -4, -2);
/ / Містить масив з елементів 2, 3
?>
```

Приклад. Використання функції array_slice (). Варіант 2

7.5.2 Функція array_chunk

Є ще одна функція, схожа на **array_slice ()** - це **array_chunk ()**. Вона розбиває масив на кілька підмасивів заданої довжини. Синтаксис її такий:

```
array_chunk (масив, розмір
[ Збережуть_ключі])
```

У результаті роботи array_chunk () повертає багатовимірний масив, елементи якого являють собою отримані підмасиви. Якщо задати параметр зберігати ключі як true, то при розбитті будуть збережені ключі вихідного масиву. В іншому випадку ключі елементів замінюються числовими індексами, які починаються з нуля.

Приклад. У нас є список запрошених, оформлений у вигляді масиву їх прізвищ. У нас є столики на три персони. Тому потрібно розподілити всіх запрошених по троє.

```
<? Php
$ Persons = array ("Іванов", "Петров",
"Сидорова", "Зайцева", "Волкова");
$ Triples = array_chunk ($ persons, 3);
```

```
/ / Ділимо масив на підмасиви
/ / По три елементи
foreach ($ triplesas $ k => $ table) {
/ / Виводимо отримані трійки
echo "За столиком номер $ k сидять: <ul>";
foreach ($ tableas $ pers)
echo "<li> $ pers";
echo "</ ul>";
}
?>
```

Приклад. Використання функції array_chunk ()

У результаті отримаємо:

за столиком номер 0 сидять:

- Іванов
- Петров
- Сидорова

за столиком номер 1 сидять:

- Зайцева
- Волкова

7.6 Сума елементів масиву

У цьому розділі ми познайомимося з функцією, що обчислює суму всіх елементів масиву. Сама задача обчислення суми значень масиву гранично проста. Але навіщо писати зайвий раз один і той самий код, якщо можна скористатися спеціально створеною і завжди доступною функцією. Функція ця називається, як можна здогадатися, array_sum (). І як параметр їй передається тільки ім'я масиву, суму значень елементів якого потрібно обчислити.

Як приклад використання цієї функції наведемо рішення більш складного завдання, ніж просто обчислення суми елементів. Цей приклад також ілюструє застосування функції array_slice (), яку ми обговорювали трохи раніше.

Приклад. Програма пошуку числа, такого що сума елементів праворуч від нього дорівнює сумі елементів зліва від нього

```
<? Php
/ / Масив задається функцією array
$ Arr = array (2,1,3,4,5,6,4);
/ / Перебираємо кожний елемент масиву $ arr.
/ / Для циклу поточний ключ масиву
/ / Міститься у змінній $ k,
/ / Поточне значення - у змінній $ val
foreach ($ arras $ k => $ val) {
$ P = $ k + 1;
/ / Синтаксис arrayarray_slice (
/ / Arrayarray, intoffset [, intlength])
/ / Array_slice виділяє підмасив
/ / Довжини length в масиві array,
/ / Починаючи з елемента offset.
$ Out_next = array_slice ($ arr, $ p);
/ / Отримуємо масив елементів,
/ / Що йдуть після поточного
$ Out_prev = array_slice ($ arr, 0, $ k);
/ / Отримуємо масив елементів,
/ / Що йдуть перед поточним
/ / Функція mixedarray_sum (arrayarray)
/ / Підраховує суму елементів масиву array
$ Next_sum = array_sum ($ out_next);
$ Prev_sum = array_sum ($ out_prev);
/ / Якщо сума елементів до поточного дорівнює
/ / Сумі елементів після, то виводимо
/ / Значення поточного елемента
if ($ next_sum == $ prev_sum)
echo "value: $ val";
/ / Можна подивитися, що представляють собою
/ / Розглянуті масиви на кожному кроці
/ / Print_r ($ out_next); echo "<br>";
/ / Print_r ($ out_prev);
/ / Echo "$ next_sum, $ prev_sum<br>";
echo "<hr>";
}                                                    ?>
```

Тема 8 Робота з рядками

8.1 Визначення рядка

В одній з перших лекцій ми приводили три способи завдання рядків: за допомогою одинарних лапок, подвійних лапок і з допомогою heredoc-синтаксису. Відзначали ми й основні відмінності між цими способами. В основному вони стосуються обробки змінних і керуючих послідовностей всередині рядка.

Приклад. Способи завдання рядків

```
<? Php
echo 'У такому рядку НЕ обробляються
змінні і більшість
послідовностей ';
echo "Тут змінні і послідовності
обробляються ";
echo<<<EOT
Тут теж обробляються як змінні,
так і керуючі послідовності.
І крім того, можна вводити символи лапок
без їх екранування зворотним слешем.
EOT;
?>
```

Вже не раз, починаючи з самої першої лекції, ми використовували функцію echo. Насправді, echo - не функція, а мовна конструкція, тому використовувати при її виклику круглі дужки не обов'язково. Echo дозволяє виводити на екран рядки, передані їй в якості параметрів. Параметрів у echo може бути скільки завгодно. Їх поділяють комами або поєднують за допомогою оператора конкатенації і ніколи не розміщують в круглі дужки.

```
<?
echo "Прийшов", "побачив", "переміг";
/ / Виведе рядок "Прийшов побачив переміг"
/ / Багато хто воліє передавати кілька
/ / Параметрів у echo за допомогою конкатенації
```

```
echo "Прийшов". "Побачив". "Переміг";
/ / Теж виведе рядок
/ / "Прийшов побачив переміг"
echo ("Прийшов", "побачив", "переміг");
/ / Видасть помилку: unexpected ','
?>
```

Приклад. Використання функції echo

Існує скорочений синтаксис для команди echo:

```
<? = Рядок_для_виводу?>
```

Тут параметр рядок_для_виводу містить рядок, заданий будь-яким з відомих способів, який повинне бути виведений на екран.

Наприклад, такий скрипт виведе на екран червоним кольором "Мене звуть Вася":

```
<? $ Name = "Вася"?>
<fontcolor=red> Мене звуть <? = $ name?></ font>
```

Крім мовної конструкції echo існує ряд функцій для виводу рядків. Це в першу чергу функція print і її різновиду printf, sprintf і т.п.

Функція **print** дозволяє виводити на екран тільки один рядок і, як і echo, не може бути викликана за допомогою змінних функцій, оскільки є мовною конструкцією.

Функція **print_r** не відноситься до строкових функцій, як можна було б подумати. Вона відображає інформацію про змінну в формі, зрозумілою користувачеві.

Функції **sprintf** і **printf** обробляють переданий їм рядок у відповідності із заданим форматом. Але про них ми говорити не будемо. А поговоримо про те, як можна здійснювати пошук в тексті, представленому у вигляді рядка.

8.2 Пошук елемента в рядку

Для того щоб визначити, чи входить даний підрядок до складу рядка, використовується функція strpos (). Синтаксис strpos () такий:

strpos (вихідний рядок, рядок для пошуку
[З якого символу шукати])

Вона повертає позицію появи шуканого рядку у вихідному рядку або повертає логічне false, якщо входження не знайдено. Додатковий аргумент дозволяє задавати символ, починаючи з якого буде проводитися пошук. Крім логічного false ця функція може повертати і інші значення, які приводяться до false (наприклад, 0 або ""). Тому для того, щоб перевірити, чи знайдений шуканий рядок, рекомендують використовувати оператор еквівалентності «===».

```
<?
$ Str = "Ідея наносити дані на перфокарти
і потім зчитувати та обробляти їх
автоматично належала Джону Біллінгс,
а її технічне рішення здійснив Герман
Холлеріт. Перфокарта Холлеріта виявилася
настільки вдалою, що без жодних змін
проіснувала до наших днів. ";
$ Pos = strpos ($ str, "Холлеріт");
if ($ pos! == false) echo "Шуканий рядок
зустрінута в позиції номер $ pos ";
elseecho "Шуканий рядок не знайдено";
/ * Зауважимо, що ми перевіряємо значення
$ Pos на еквівалентність з false.
Інакше рядок, що знаходиться в першій позиції,
не був би знайдений, тому що 0
інтерпретується як false. * /
?>
```

Приклад. Використання функції strpos ()

Якщо значення параметра рядок_для_пошука не є рядком, то воно перетвориться до цілого типу і розглядається як ASCII-код

символу. Щоб отримати ASCII-код будь-якого символу в PHP, можна скористатися функцією ord ("символ")

Наприклад, якщо ми напишемо $ pos = strpos ($ str, 228); то інтерпретатор буде вважати, що ми шукаємо символ «д». Якщо додати цей рядок у наведений вище приклад і вивести результат, то отримаємо повідомлення, що шуканий рядок знайдений в позиції 1.

Функція, обернена за змістом **ord**, - це **chr** (код символу). Вона по ASCII-коду виводить символ, що відповідає цьому коду.

За допомогою функції **strpos** можна знайти номер тільки першої появи рядка у заданому рядку. Природно, є функції, які дозволяють обчислити номер останньої появи рядка у заданому рядку. Це функція strrpos (). Її синтаксис такий:

strrpos (вихідний рядок, символ для пошуку)

На відміну від **strpos ()** ця функція дозволяє знайти позицію останньої появи в рядку зазначеного символу.

Бувають ситуації, коли знати позицію, де знаходиться шуканий рядок, необов'язково, а потрібно просто отримати всі символи, які розташовані після входження цього рядка. Можна, звичайно, скористатися і наведеними вище функціями strpos () і strrpos (), але можна зробити і простіше - виділити підрядок з допомогою призначених саме для цього функцій.

8.3 Виділення підрядка

8.3.1 Функція strstr

Говорячи про виділення підрядка з шуканого рядки в мові PHP, в першу чергу варто відзначити функцію **strstr ()**:

strstr (вихідний рядок, рядок для пошуку)

Вона знаходить першу появу шуканого рядка і повертає підрядок, починаючи з цього шуканого рядка до кінця початкового рядка.

Якщо рядок для пошуку не знайдено, то функція поверне false. Якщо рядок для пошуку не належить строковому типу даних, то вона переводиться в ціле число і розглядається як код символу. Крім того, ця функція чутлива до регістру, тобто якщо

ми будемо паралельно шукати входження слів «Ідея» та «ідея», то результати будуть різними. Замість strstr () можна використовувати абсолютно ідентичну їй функцію strchr ().

Приклад. Виділимо з рядка, що містить назву та автора дослідження, підрядок, що починається зі слова «Назва»:

```
<?
$ Str = "Автор: Іванов Іван (<a
href = mailto: van@ukr.net> написати лист </ a>),
Назва: 'Дослідження мов
програмування '";
echo "<b> Вихідний рядок: </ b>", $ str;
if (! strstr ($ str, "Назва"))
echo "Рядок не знайдено <br>";
elseecho "<p><b> Отриманий підрядок: </ b>",
strstr ($ str, "Назва");
?>
```

Приклад. Використання функції strstr ()

```
У результаті отримаємо:
Вихідний рядок: Автор: Іванов Іван
(Написати лист),
Назва: 'Дослідження мов
програмування '
Отриманий підрядок: Назва:
'Дослідження мов програмування'
```

Для реалізації регістронезалежного пошуку підрядка існує відповідний аналог цієї функції - функція **stristr**(вихідний рядок, шуканий рядок). Діє і використовується вона точно так само, як і **strstr ()**, за винятком того, що регістр, в якому записані символи шуканого рядка, не грають ролі при пошуку.

Очевидно, що функція **strstr ()** не надто часто використовується - на практиці рідко буває потрібно отримати підрядок, що починається з певного слова або рядка. Але в деяких випадках і вона може стати в нагоді. Крім того, в PHP є і більш зручні функції для пошуку входжень. Найбільш потужні з

них, звичайно, пов'язані з регулярними виразами. Їх ми розглянемо в одній з наступних лекцій.

8.3.2 Функція substr

Іноді ми не знаємо, з яких символів починається шуканий рядок, але знаємо, наприклад, що починається він з п'ятого символу і закінчується за два символи до кінця початкового рядка. Як виділити підрядок за таким описом? Дуже просто, за допомогою функції substr (). Її синтаксис можна записати наступним чином:

substr (вихідний рядок, позиція початкового символу [, довжина])

Ця функція повертає частину рядка завдовжки, заданої параметром довжина, починаючи з символу, зазначеного параметром позиція початкового символу. Позиція, з якої починається виділяється підрядок, може бути як позитивним цілим числом, так і негативним. В останньому випадку відлік елементів проводиться з кінця стрічки. Якщо параметр довжина опущений, то substr () повертає підрядок від зазначеного символу і до кінця початкового рядка. Довжина підрядка, що виділяється теж може бути задана негативним числом. Це означає, що вказане число символів відкидається з кінця рядка.

Приклад. Припустимо, у нас є фраза, виділена жирним шрифтом за допомогою тега<b> мови HTML. Ми хочемо отримати цю ж фразу, але в звичайному стилі. Напишемо таку програму:

```
<? Php
$ Word = "<b>Hello, world! </ B>";
echo $ word, "<br>";
$ Pure_str = substr ($ word, 3, -4);
/ * Виділяємо підрядок,
починаючи з 3-го символу,
не включаючи 4 символи з кінця рядка * /
echo $ pure_str;
?>
```

Приклад. Використання функції substr ()

У результаті роботи цього скрипта одержимо:

Hello, world!

Hello, world!

Насправді вирішити таке завдання можна набагато простіше, за допомогою функції strip_tags:

strip_tags (рядок [, допустимі теги])

Ця функція повертає рядок, з якої вилучені всі html і php-теги. За допомогою додаткового аргументу можна задати теги, які не будуть видалені з рядка. Список з кількох тегів вводиться без будь-яких знаків роздільників. Функція видає попередження, якщо зустрічає неправильні або неповні теги.

```
<? Php
$ String = "<b>Boldtext</ b>
<i>Italictext</ i> ";
$ Str = strip_tags ($ string);
/ / Видаляємо всі теги з рядка
$ Str1 = strip_tags ($ string, '<b>');
/ / Видаляємо всі теги крім тега<b>
$ Str2 = strip_tags ($ string, '<i>');
/ / Видаляємо всі теги крім тегів <i>
echo $ str, "<br>", $ str1, "<br>", $ str2;
?>
```

Приклад. Використання функції strip_tags ()

У результаті отримаємо:

BoldtextItalictext

BoldtextItalictext

BoldtextItalictext

Наведемо ще один приклад використання функції substr (). Припустимо, у нас є якесь повідомлення з привітанням і підписом автора. Ми хочемо видалити спочатку вітання, а потім і підпис, залишивши тільки змістовну частину повідомлення.

```
<? Php
$ Text = "Привіт! Сьогодні ми вивчаємо роботу
з рядками. Автор. ";
```

```
$ No_hello = substr ($ text, 8);
/ / Прибираємо привітання
$ Content = substr ($ text, 8, 45);
/ / Те ж саме, що substr ($ text, 8, -6).
/ / Прибираємо підпис.
echo $ text, "<br>", $ no_hello,
"<br>", $ Content;
?>
```

У результаті отримаємо:

Привіт! Сьогодні ми вивчаємо роботу
з рядками. Автор.
Сьогодні ми вивчаємо роботу з рядками. Автор.
Сьогодні ми вивчаємо роботу з рядками.

Якщо нам потрібно отримати один конкретний символ з рядка, знаючи його порядковий номер, то не слід задіяти функцій типу substr. Можна скористатися більш простим синтаксисом - записуючи номер символу в фігурних дужках після імені строкової змінної. У контексті попереднього прикладу букву «р», розташовану другою за рахунком, можна отримати так:

echo $ text {1}; / / виведе символ "р"

Зауважимо, що номером цього символу є число один, а не два, так як нумерація символів рядка починається з нуля.

Раз вже ми почали говорити про символи в рядку і їх нумерації, то мимоволі виникає питання, скільки всього символів в рядку і як це визначити. Кількість символів у рядку - це довжина рядка. Обчислити довжину рядка можна за допомогою функції strlen (рядок). Наприклад, довжина рядка «Розробка інформаційної моделі» обчислюється за допомогою команди: strlen ("Розробка інформаційної моделі"); і дорівнює 32 символам.

Отже, як виділяти і знаходити підрядки, ми розглянули. Тепер навчимося замінювати рядок, що входить до складу початкового рядка, на інший рядок за нашим вибором.

8.4 Заміна входження підрядка

8.4.1 Функція str_replace

Для заміни входження підрядка можна використовувати функцію str_replace (). Це проста і зручна функція, що дозволяє вирішувати безліч завдань, які потребують особливих тонкощів при виборі замінної підрядка. Для того щоб робити заміни з більш складними умовами, використовують механізм регулярних виразів і відповідні функції ereg_replace () і preg_replace (). Синтаксис функції str_replace () такий:

str_replace (шукане значення, значення для заміни, об'єкт)

Функція **str_replace ()** шукає в даному об'єкті значення і заміняє його значенням, призначеним для заміни.

Якщо об'єкт, в якому проводиться пошук і заміна, є масивом, то ці дії виконуються для кожного елемента масиву і в результаті повертається новий масив.

```
<? Php
$ Greeting = array ("Привіт", "Привіт всім!",
"Привіт, дорога!"); / / Об'єкт
$ New_greet = str_replace ("Привіт",
"Доброго ранку", $ greeting);
/ / Робимо заміну
print_r ($ new_greet);
/ * Отримаємо: Array ([0] => Добрий ранок
[1] => Добрий ранок усім!
[2] => Добрий ранок, дорога!) * /
?>
```

Приклад. Використання функції str_replace ()

Якщо шукане значення і значення для заміни - масиви, то береться по одному значенню з кожного масиву і проводиться їх пошук і заміна в об'єкті. Якщо значень для заміни менше, ніж значень для пошуку, то в якості нових значень використовується порожній рядок.

```
<? Php
```

```
$ Greeting = array ("Привіт", "Привіт всім!",
"Привіт, дорога!", "Здрастуйте",
"Здрастуйте, товариші", "Ні");
/ / Об'єкт
$ Search = array ("Привіт",
"Добрий день", "Ні");
/ / Значення, що будемо замінювати
$ Replace = array ("Доброго ранку",
"День добрий");
/ / Значення, якими будемо замінювати
$ New_greet = str_replace ($ search, $ replace,
$ Greeting);
/ / Робимо заміну
print_r ($ new_greet);
/ / Виводимо отриманий масив
?>
```

Приклад. Використання функції str_replace (). Варіант 2

У результаті отримаємо такий масив:

```
Array (
[0] => Добрий ранок
[1] => Добрий ранок усім!
[2] => Добрий ранок, кохана!
[3] => День добрий
[4] => День добрий, товариші
[5] =>
)
```

Якщо значення для пошуку - масив, а значення для заміни - рядок, то цей рядок буде використаний для заміни всіх знайдених значень.

```
<? Php
$ Greeting = array ("Привіт", "Привіт всім!",
"Привіт, дорога!", "Здрастуйте",
"Здрастуйте, товариші");
/ / Об'єкт
$ Search = array ("Привіт", "Здрастуйте");
```

/ / Значення, що будемо замінювати
$ Replace = "День добрий";
/ / Значення, яким будемо замінювати
$ New_greet = str_replace ($ search,
$ Replace, $ greeting); / / робимо заміну print_r ($ new_greet);
/ / Виводимо отриманий масив
?>

Приклад. Використання функції str_replace (). Варіант 3

Отримаємо:

Array (
[0] => День добрий
[1] => День добрий усім!
[2] => День добрий, дорога!
[3] => День добрий
[4] => День добрий, товариші
)

Функція **str_replace ()** чутлива до регістру, але існує її регістронезалежний аналог - функція **str_ireplace ()**. Однак ця функція підтримується не у всіх версіях PHP.

Ще один приклад використання функції str_replace () - обробка шаблонів.

Звернемося в черговий раз до опису будь-якого документа, наприклад статті. Багато разів ми вже створювали форму для введення подібного опису і навіть відображали дані, введені користувачем у такого роду форму. Але як відображати ці дані, ми описували безпосередньо в коді нашої програми. Тепер ми хочемо, щоб спосіб відображення даних ставив сам користувач. Для цього додамо в нашу форму ще один елемент для введення шаблону.

<h2> Введіть опис статті </ h2>
<formaction=sbl.php>
<tableborder=0>
<tr><td> Назва </ td><td><input
type = text name = title></ td></ tr>
<tr><td> Короткий зміст </ td><td><input
type = textareaname = description></ td></ tr>

```
<tr><td> Автор </ td><td><input
type = text name = author></ td></ tr>
<tr><td> Дата публікації </ td><td><input
type = text name = published></ td></ tr>
<tr><td> Шаблон документа </ td><td><textarea
name = shablon></ textarea></ td></ tr>
</ Table>
<input type=submit value="Відправити">
</ Form>
```

Однак просто поля для введення шаблону недостатньо. Одна людина введе в нього одне, інший - інше. Потрібно домовитися про те, як створювати шаблони, що можна в них використовувати, тобто потрібно придумати мову шаблонів. Наприклад, ми домовляємося, що під час створення шаблону можна задіяти будь-html-теги, а набір спецсимволів виду визначає значення елемента з іменем ім'я_елемента. Далі, як обробляти такого роду шаблони? Можна використовувати функцію str_replace ():

```
<? Php
$ Tmpl = $ _GET ["shablon"];
/ * Шаблон, ввели.
Наприклад, це може бути такий рядок:
"<h1><! Title></ h1><p>
size =- 1><! description></ font></ p><p
align = right><! author><br><! published></ p> "* /
functionShow () {
/ / Функція, яка робить заміну
/ / Елементу шаблону на його значення
global $ tmpl;
foreach ($ _GET as $ k => $ v) {
$ Tmpl = str_replace (" ", $ v, $ tmpl);
}
echo $ tmpl;
}
Show ();
```

?>

Як ці файли виглядають для звичайного користувача? Якщо ми введемо в форму такі дані як показано на рисунку 8.1, то в результаті отримаємо:

Перша автоматизований перепис населення

Ідея автоматизувати процедуру перепису населення належала винахіднику ....

А. М. Федотов

2.02.13

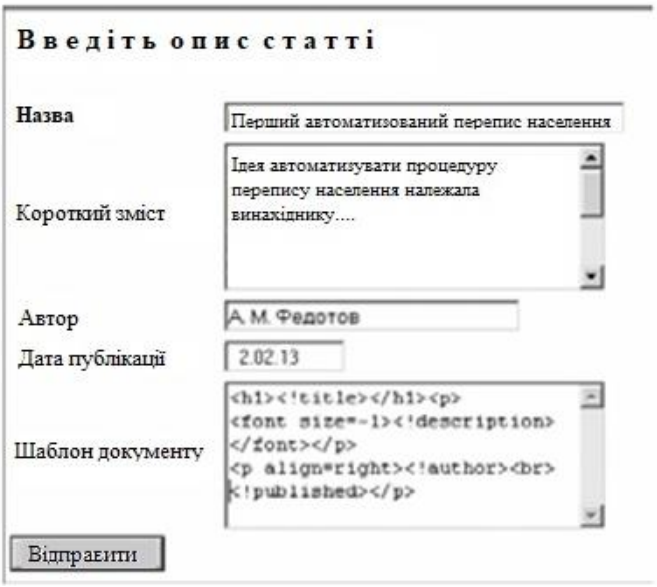


Рис. 8.1. Форма для введення опису документа «стаття» і шаблону для його відображення

8.4.2 Функція substr_replace

Ця функція поєднує в собі властивості двох уже розглянутих нами функцій - функції str_replace () і substr (). Її синтаксис такий:

substr_replace (вихідний рядок,

рядок для заміни,

позиція початкового символу [, довжина])

Ця функція замінює частину рядка рядком, призначеного для заміни. Замінюється та частина рядка (тобто підрядок), яка починається з позиції, вказаної параметром позиція початкового символу. За допомогою додаткового аргументу довжина можна обмежити число замінних символів. Тобто, фактично, ми не вказуємо конкретно рядок, який потрібно замінити, ми тільки описуємо, де вона знаходиться і, можливо, яку довжину має. У цьому відмінність функції substr_replace () від str_replace ().

Як і у випадку з функцією substr () аргументи позиції початкового символу і довжина можуть бути негативними. Якщо позиція початкового символу негативна, то заміна відбувається, починаючи з цієї позиції щодо кінця рядка. Негативна довжина задає, скільки символів від кінця рядка не повинно бути замінено. Якщо довжина не вказується, то заміна відбувається до кінця рядка.

```
<? Php
$ Text = "Мене звуть Вася.";
echo "Вихідний рядок: $ text<hr> \ n";
/ * Наступні два рядки замінять весь
вихідний рядок рядком 'А мене - Петя' * /
echosubstr_replace ($ text, 'А мене - Петя',
0). "<br> \ N";
echosubstr_replace ($ text, 'А мене - Петя',
0, strlen ($ text)). "<br> \ N";
/ / Наступний рядок додасть слово 'Привіт! '
/ / У початок початкового рядка
echosubstr_replace ($ text, 'Привіт!',
0, 0). "<br> \ N";
/ / Наступні два рядки замінять ім'я Вася
/ / На ім'я Іван у вихідному рядку
echosubstr_replace ($ text, 'Іван', 11,
-1). "<br> \ N";
echosubstr_replace ($ text, 'Іван', -5,
-1). "<br> \ N";
?>
```

Приклад. Використання функції substr_replace ()
У результаті роботи цього скрипта одержимо:
Вихідний рядок: Мене звуть Вася.

А мене - Петя
А мене - Петя
Привіт! Мене звуть Вася.
Мене звуть Іван.
Мене звуть Іван.

8.5 Поділ і з'єднання стрічки

Дуже корисні функції - функція розподілу рядка на частини і зворотна їй функція об'єднання рядків в один рядок. Чому дуже корисні? Наприклад, якщо ви динамічно генеруєте форму за бажанням користувача, можна запропонувати йому вводити елементи для створення списку вибору, розділяючи їх яким-небудь символом. І для того щоб обробити отриманий список значень, якраз і стане в нагоді вміння розбивати рядок на шматочки. Для реалізації такого розбиття в PHP можна використовувати кілька функцій:

explode (роздільник, вихідний рядок
[Максимальна кількість елементів])
split (шаблон, вихідний рядок
[Максимальна кількість елементів])
preg_split (шаблон, вихідний рядок
[Максимальна кількість елементів
[Прапори]])

Останні дві функції працюють з регулярними виразами, тому в даній лекції ми їх розглядати не будемо. Розглянемо більш просту функцію - explode ().

Функція explode () ділить заданий рядок на підрядки, кожний з яких відділений від сусіднього з допомогою зазначеного роздільника, і повертає масив отриманих рядків. Якщо задано додатковий параметр Максимальна кількість елементів, то число елементів у масиві буде не більше цього параметра, в останній

елемент записується весь залишок рядка. Якщо в якості роздільника вказана порожній рядок «""», то функція explode () поверне false. Якщо символу роздільника у вихідній рядку немає, то повертається вихідний рядок без змін.

Приклад. ми хочемо створити елемент форми - випадаючий список і значення для цього списку повинен ввести користувач, не знайомий з мовою html. Створимо таку форму:

<formaction=exp.php>
Введіть варіанти для вибору автора статті
через двокрапку (":"):
<input type=textname=authorsize=40>

<input type=submit value=Створити елемент>
</ Form>

Приклад. Використання функції explode ()

Скрипт, який буде її обробляти (exp.php), може бути таким:

<? Php
$ Str = $ _GET ["author"];
$ Names = explode (":",$ str); /* Розбиваємо рядок введений, Користувачем за допомогою ":" */
$ S = "<selectname=author>"; // Створюємо список, що випадає
foreach ($ namesas $ k => $ name) {
$ S .= "

У результаті, якщо ми введемо такий рядок у форму:

Рис. 8.2. Введення значень для створення списку, що випадає то отримаємо список, що випадає.

Крім поділу рядка на частини іноді, навпаки, виникає необхідність об'єднання кількох рядків в одне ціле. Функція, пропонована для цього мовою PHP, називається implode ():

implode (string $ glue, array $ pieces)

Ця функція об'єднує елементи масиву з допомогою переданого їй об'єднуючого елементу (наприклад, коми). На відміну від функції explode (), порядок аргументів у функції implode () не має значення.

Приклад. Припустимо, ми зберігаємо ім'я, прізвище і по батькові людини окремо, а виводити їх на сторінці потрібно разом. Щоб з'єднати їх в один рядок, можна використовувати функцію implode ():

```
<? Php
$ Data = array ("Іванов", "Іван", "Іванович");
$ Str = implode ("", $ data);
echo $ str;
?>
```

Приклад. Використання функції implode ()

У результаті роботи цього скрипта отримаємо рядок:

Іванов Іван Іванович

У функції **implode ()** існує псевдонім - функція join (), тобто ці дві функції відрізняються лише іменами.

8.6 Рядки, що містять html-код

Досить часто ми працюємо з рядками, що містять html-теги. Якщо відобразити такий рядок в браузер за допомогою звичайних функцій відображення даних echo () або print (), то ми не побачимо самих html-тегів, а отримаємо відформатований у відповідності з цими тегами рядок. Браузер обробляє всі html-теги у відповідності зі стандартом мови HTML. Іноді нам потрібно бачити безпосередньо рядок, без обробки його браузером. Щоб цього досягти, потрібно перед тим, як виводити, застосувати до нього функцію htmlspecialchars ().

Функція **htmlspecialchars** (рядок [, стиль лапок [, кодування]]) переводить спеціальні символи, такі як «<», «>», «&», «" »,« '»у такі сутності мови HTML, як« <» , «>», «&», «" »,« '»відповідно.

Додатковий аргумент стиль лапок визначає, як повинні інтерпретуватися подвійні та одинарні лапки. Він може мати одне з трьох значень: ENT_COMPAT, ENT_QUOTES, ENT_NOQUOTES. Константа ENT_COMPAT означає, що подвійні лапки повинні бути переведені в спецсимволи, а одинарні повинні залишитися без змін. ENT_QUOTES говорить, що повинні конвертуватися і подвійні та одинарні лапки, а ENT_NOQUOTES залишає і ті й інші лапки без змін.

У пункті кодування можуть бути задані такі кодування, як UTF-8, ISO-8859-1 та інші.

```
<? Php
$ New = htmlspecialchars ("<a
href = 'mailto: au@ukr.net'>
Написати листа </ a> ", ENT_QUOTES);
echo $ new;
/ * Наш рядок перекодовується в такий:
<a href='mailto:au@ukr.net'>
Написати листа </ a> * /
```

Приклад. Використання функції htmlspecialchars ()

```
У браузері ми побачимо:
<a href='mailto:au@ukr.net'>
Написати листа </ a>
```

Функція htmlspecialchars () перекодує тільки найбільш часто використовувані спецсимволи. Якщо необхідно конвертувати всі символи по суті HTML, слід задіяти функцію htmlentities (). Російські літери при використанні цієї функції теж кодуються спеціальними послідовностями. Наприклад, літера «А» замінюється комбінацією «А». Її синтаксис і принцип дії аналогічний синтаксису і принципом дії htmlspecialchars ().

Тема 9 Робота з файловою системою

9.1 Створення файлу

Функція fopen

Взагалі кажучи, в PHP не існує функції, призначеної саме для створення файлів. Більшість функцій працюють з вже існуючими файлами в файловій системі сервера. Є кілька функцій, які дозволяють створювати тимчасові файли, або, що те ж саме, файли з унікальним для поточної директорії імені. А от для того, щоб створити самий звичайний файл, потрібно скористатися функцією, яка відкриває локальний або віддалений файл. Називається ця функція fopen (). Що значить «відкриває файл»? Це означає, що fopen пов'язує цей файл з потоком управління програми. Причому зв'язування буває різним в залежності від того, що ми хочемо робити з цим файлом: читати його, записувати в нього дані або робити і те й інше. Синтаксис цієї функції такий:

resourcefopen (ім'я_файлу, тип_доступу [Use_include_path])

У результаті роботи ця функція повертає покажчик (типу ресурс) на відкритий нею файл. В якості параметрів цієї функції передаються: ім'я файлу, який потрібно відкрити, тип доступу до файлу (визначається тим, що ми збираємося робити з ним) і, можливо, параметр, що визначає, чи шукати вказаний файл у include_path. Є ще один параметр, але про нього ми говорити не будемо, щоб не ускладнювати виклад. Обговоримо докладніше кожен з цих трьох параметрів.

Параметр ім'я_файлу повинен бути рядком, що містить правильне локальне ім'я файлу або URL-адресу файлу в мережі. Якщо ім'я файлу починається з вказівки протоколу доступу (наприклад, http:// ... або ftp:// ...), то інтерпретатор вважає це ім'я адресою URL і шукає обробник зазначеного в URL протоколу. Якщо обробник знайдений, то PHP перевіряє, чи дозволено працювати з об'єктами URL як зі звичайними файлами (директива allow_url_fopen). Якщо allow_url_fopen = off, то функція fopen викликає помилку і генерується попередження.

Якщо ім'я файлу не починається з протоколу, то вважається, що зазначено ім'я локального файлу. Щоб відкрити локальний файл, потрібно, щоб PHP мав відповідні права доступу до цього файлу.

Параметр use_include_path, встановлений в значення 1 або TRUE, змушує інтерпретатор шукати зазначений у fopen () файл у include_path. Нагадаємо, що include_path - це директива з файлу налаштувань PHP, що задає список директорій, в яких можуть знаходитися файли для включення. Крім функції fopen () вона використовується функціями include () і require ().Параметр use_include_path, встановлений в значення 1 або TRUE, змушує інтерпретатор шукати зазначений у fopen () файл у include_path. Нагадаємо, що include_path - це директива з файлу налаштувань PHP, що задає список директорій, в яких можуть знаходитися файли для включення. Крім функції fopen () вона використовується функціями include () і require ().

Параметр тип_доступу може приймати одне з наступних значень (див. таб. 9.1).

Отже, щоб створити файл, потрібно, як би безглуздо це не звучало, відкрити неіснуючий файл на запис.

```
<? Php
$ H = fopen ("my_file.html", "w");
/ * Відкриває на запис файл my_file.html,
якщо він існує, або створює порожній
файл з таким ім'ям, якщо його ще немає * /
$ H = fopen ("dir / another_file.txt", "w +");
/ * Відкриває на запис і читання або створює
файл another_file.txt в директорії dir * /
$ H = fopen (
"Http://www.server.ru/dir/file.php", "r");
/ * Відкриває на читання файл, що знаходиться за
вказаною адресою * /
?>
```

Приклад. Використання функції fopen ()

Створюючи файл, потрібно враховувати, під якою операційною системою ви працюєте, і під яку ОС імовірно цей файл буде читатися. Справа в тому, що різні операційні системи по-різному позначають кінець рядка. У Unix-подібних ОС кінець рядка позначається \ n, в системах типу Windows - \ r \ n. Windows пропонує спеціальний прапорецьt для перекладу символів кінця рядка систем типу Unix в свої символи кінця рядка. На противагу цьому існує прапорецьb, використовуваний найчастіше для бінарних файлів, завдяки якому такої трансляції не відбувається. Використовувати ці прапори можна, просто дописавши їх після останнього символу обраного типу доступу до файлу. Наприклад, відкриваючи файл на читання, замість r слід використовувати rt, щоб перекодувати всі символи кінця рядка в \ r \ n. Якщо не використовувати прапорецьb при відкритті бінарних файлів, то можуть з'являтися помилки, пов'язані зі зміною вмісту файлу. З міркувань переносимості програми на різні платформи рекомендується завжди використовувати прапорецьb при відкритті файлів за допомогою fopen ().

Таблиця 8.1. Значення прийняті параметром тип доступу

	Тип доступу	**Опис**
r	Відкриває файл тільки для читання; встановлює вказівник позиції у файлі на початок файлу.	
r +	Відкриває файл для читання і запису; встановлює вказівник файлу на його початок.	
w	Відкриває файл тільки для запису; встановлює вказівник файлу на його початок і обмежували файл до нульової довжини.	Якщо файл не існує, то намагається створити його.
w +	Відкриває файл для читання і запису; встановлює вказівник файлу на його початок і обмежували файл до нульової довжини.	Якщо файл не існує, то намагається створити його.
a	Відкриває файл тільки для запису;	Якщо файл не

	встановлює вказівник файлу в його кінець.	існує, то намагається створити його.
a +	Відкриває файл для читання і запису; встановлює вказівник файлу в його кінець.	Якщо файл не існує, то намагається створити його.
x	Створює і відкриває файл тільки для запису; поміщає вказівник файлу на його початок.	Якщо файл вже існує, то fopen () повертає false і генерується попередження. Якщо файл не існує, то робиться спроба створити його.
x +	Створює і відкриває файл для читання і запису; поміщає вказівник файлу на його початок.	Якщо файл вже існує, то fopen () повертає false і генерується попередження. Якщо файл не існує, то робиться спроба створити його.

Що відбувається, якщо відкрити або створити файл за допомогою fopen не вдається? У цьому випадку PHP генерує попередження, а функція fopen повертає як результат своєї роботи значення false. Такого роду попередження можна «придушити» (заборонити) за допомогою символу @.

Наприклад, така команда не виведе попередження, навіть якщо відкрити файл не вдалося:

$ H = @ fopen ("dir / another_file.txt", "w +");

Таким чином, функція fopen () дозволяє створити тільки лише порожній файл і зробити його доступним для запису. Як же

записати дані в цей файл? Як прочитати дані з вже існуючого файлу?

Перш ніж відповісти на ці запитання, розглянемо, як закрити встановлене з допомогою fopen () з'єднання.

9.2 Закриття з'єднання з файлом

Після виконання необхідних дій з файлом, будь то читання або запис даних або що-небудь інше, з'єднання, встановлене з цим файлом функцією fopen (), потрібно закрити. Для цього використовують функцію fclose (). Синтаксис у неї наступний:

fclose (покажчик на файл)

Ця функція повертає TRUE, якщо з'єднання успішно закрито, і FALSE - у противному випадку. Аргумент цієї функції повинен вказувати на файл, успішно відкритий, наприклад, за допомогою функції fopen ().

```
<? Php
$ H = fopen ("my_file.html", "w");
fclose ($ h);
?>
```

Приклад. Використання функції fclose ()

Звичайно, якщо не закривати з'єднання з файлом, ніяких помилок виконання скрипта не відбудеться. Але в цілому для сервера це може мати серйозні наслідки. Наприклад, хакер може скористатися відкритим з'єднанням і записати в файл вірус, не кажучи вже про зайві витрати ресурсів сервера. Тож радимо завжди закривати з'єднання з файлом після виконання необхідних дій.

9.3 Запис даних у файл

Функція fwrite

Для того щоб записати дані у файл, доступ до якого відкритий функцією fopen (), можна використовувати функцію fwrite (). Синтаксис у неї наступний:

intfwrite (покажчик на файл, рядок [, довжина])

Ця функція записує рядок у файл, на який вказує покажчик на файл. Якщо вказаний додатковий аргумент довжина, то запис закінчується після того, як записано кількість символів, рівне значенню цього аргументу, або коли буде досягнуто кінець рядка.

У результаті своєї роботи функція fwrite () повертає число записаних байтів або false, у разі помилки.

Приклад. Нехай у нашій робочій директорії немає файлу my_file.html. Створимо його і запишемо в нього рядок тексту:

```
<? Php
$ H = fopen ("my_file.html", "w");
$ Text = "Цей текст запишемо у файл.";
if (fwrite ($ h, $ text))
echo "Запис пройшов успішно";
else
echo "Сталася помилка під час запису даних";
fclose ($ h);
?>
```

Приклад. Використання функції fwrite ()

У результаті роботи цього скрипта в браузері ми побачимо повідомлення про те, що запис пройшов успішно, а у файлі my_file.html з'явиться рядок "Цей текст запишемо у файл.". Якщо б цей файл існував до того, як ми виконали цей скрипт, що всі дані в ньому були б видалені.

Якщо ж ми напишемо такий скрипт:

```
<? Php
$ H = fopen ("my_file.html", "a");
$ Add_text = "Додамо текст у файл.";
if (fwrite ($ h, $ add_text, 7))
echo "Додавання тексту пройшло
успішно <br> ";
elseecho "Сталася помилка при
додаванні даних <br> ";
fclose ($ h);
?>
```

то до рядка, що вже існує у файлі my_file.html, додасться ще сім символів з рядка, що міститься у змінній $ add_text, тобто слово «Додамо»

Функція fwrite () має псевдонім fputs (), використовуваний таким же чином, що і сама функція.

Далі ми розглянемо, які методи читання даних з файлу пропонує мова PHP.

9.4 Читання даних з файлу

Якщо ми хочемо прочитати дані з існуючих файлів, однієї функції fopen (), як і у випадку з записом даних, недостатньо. Вона лише повертає покажчик на відкритий файл, але не зчитує ні одного рядка з цього файлу. Тому для того, щоб прочитати дані з файлу, потрібно скористатися однією із спеціальних функцій: file, readfile, file_get_contents, fread, fgets і т.п.

9.4.1 Функція fread

Ця функція здійснює читання даних з файлу. Її можна використовувати і для читання даних з бінарних файлів, не побоюючись їх пошкодження. Синтаксис fread () такий:

stringfread (покажчик на файл, довжина)

При виклику цієї функції відбувається читання даних довжини (у байтах), визначеної параметром довжина, з файлу, на який вказує покажчик на файл. Параметр покажчик на файл повинен бути реально існуючої змінної типу ресурс, що містить у собі зв'язок з файлом, відкритий, наприклад, за допомогою функції fopen (). Читання даних відбувається до тих пір, поки не зустрінеться кінець файлу або поки не буде прочитано зазначене параметром довжина число байтів.

У результаті роботи функція fread () повертає рядок із ліченої з файлу інформації.

Як ви помітили, у цій функції параметр довжина - обов'язковий. Отже, якщо ми хочемо рахувати весь файл в рядок, потрібно знати його довжину. PHP може самостійно обчислити довжину зазначеного файлу. Для цього потрібно скористатися функцією filesize (ім'я файлу). У випадку помилки

ця функція поверне false. На жаль, її можна використовувати тільки для отримання розміру локальних файлів.

Приклад. Прочитаємо вміст файлу my_file.html

```
<? Php
$ H = fopen ("my_file.html", "r +");
/ / Відкриваємо файл на запис і читання
$ Content = fread ($ h,
filesize ("my_file.html"));
/ / Зчитуємо вміст файлу в рядок
fclose ($ h); / / закриваємо з'єднання з файлом
echo $ content;
/ / Виводимо вміст файлу
/ / На екран браузера
?>
```

Приклад Використання функції fread ()

Для того щоб рахувати вміст бінарного файлу, наприклад, зображення, в таких системах, як Windows, рекомендується відкрити файл з допомогою прапора rb або йому подібних, що містять символ b в кінці.

Функція **filesize ()** кешує результати своєї роботи. Якщо змінити вміст файлу my_file.html і знову запустити наведений вище скрипт, то результат його роботи не зміниться. Більш того, якщо запустити скрипт, що зчитує дані з цього файлу з допомогою іншої функції (наприклад, fgetss), то результат може виявитися таким, як якби файл не змінився. Щоб цього уникнути, потрібно очистити статичний кеш, додавши в код програми команду clearstatcache ();

9.4.2 Функція fgets

За допомогою функції fgets () можна зчитати з файлу рядок тексту. Синтаксис цієї функції практично такий же, як і у fread (), за винятком того, що довжину зчитується рядка вказувати необов'язково:

stringfgets (покажчик на файл [, довжина])

У результаті роботи функція fgets () повертає рядок довжиною (довжина-1) байт з файлу, на який вказує покажчик

на файл. Читання закінчується, якщо прочитано (довжина-1) символів або зустрівся символ перекладу рядка або кінець файлу. Нагадаємо, що в PHP один символ - це один байт. Якщо довжина, що зчитується з рядка не зазначена, то зчитується 1 Кбайт (1024 байт) тексту або, що те ж саме, 1024 символу. Якщо параметр довжина не заданий, зчитується рядок цілком. У випадку помилки функція fgets () повертає false.

```
<? Php
$ H = fopen ("my_file.html", "r +");
$ Content = fgets ($ h, 2);
/ / Вважає перший символ з
/ / Першого рядка файлу my_file.html
fclose ($ h);
echo $ content;
?>
```

Приклад. Використання функції fgets ()

Обидві функції, fread () і fgets (), припиняють зчитування даних з файлу, якщо зустрічають кінець файлу. У PHP є спеціальна функція, що перевіряє, чи дивиться покажчик позиції файлу на кінець файлу. Це булева функція feof (), як параметр якої передається покажчик на з'єднання з файлом.

Наприклад, ось так можна вважати всі рядки файлу my_file.html:

```
<? Php
$ H = fopen ("my_file.html", "r");
while (! feof ($ h)) {
$ Content = fgets ($ h);
echo $ content, "<br>";
}
fclose ($ h);
?>
```

Функція fgetss

Існує різновид функції fgets () - функція fgetss (). Вона теж дозволяє зчитувати рядок із зазначеного файлу, але при цьому

видаляє з нього всі, що зустрілися html-теги, за винятком, можливо, деяких. Синтаксис fgetss () такий:

stringfgetss (покажчик на файл,

довжина [, допустимі теги])

Зверніть увагу, що тут аргумент довжина обов'язковий.

Приклад. Нехай у нас є файл my_file.html наступного змісту:

<h1> Без праці не виймеш і рибку зі ставка. </ h1>

<b> Тихіше їдеш - далі будеш </ b> У семи няньок <i> дитя без ока </ i>.

Виведемо на екран всі рядки файлу my_file.html, видаливши з них всі теги, крім <b> і <i>:

```
<? Php
$ H = fopen ("my_file.html", "r");
while (! feof ($ h)) {
$ Content = fgetss ($ h, 1024, '<b><i>' );
echo $ content, "<br>";
}
fclose ($ h);
?>
```

Приклад. Використання функції fgetss ()

У результаті роботи цього скрипта одержимо:

Без праці не виймеш і рибку зі ставка.

Тихіше їдеш - далі будеш У семи няньок дитя без ока.

9.4.3 Функція fgetc

Природно, якщо можна зчитувати інформацію з файлу порядково, то можна зчитувати її і посимвольно. Для цього призначена функція fgetc (). Легко здогадатися, що синтаксис у неї наступний:

stringfgetc (покажчик на файл)

Ця функція повертає символ з файлу, на який посилається вказівник на файл, і значення, що обчислюється як FALSE, якщо зустрінутий кінець рядка.

Ось так, наприклад, можна вважати файл по одному символу:

```
<? Php
$ H = fopen ("my_file.html", "r");
```

```
while (! feof ($ h)) {
$ Content = fgetc ($ h);
echo $ content, "<br>";
}
fclose ($ h);
?>
```

Насправді для того щоб прочитати вміст файлу, відкривати з'єднання з ним за допомогою функції fopen () зовсім не обов'язково. У PHP є функції, які дозволяють робити це, використовуючи лише ім'я файлу. Це функції readfile (), file () і file_get_contents (). Розглянемо кожну з них детальніше.

9.4.4 Функція readfile

Синтаксис:

intreadfile (ім'я_файлу [Use_include_path])

Функція readfile () зчитує файл, ім'я якого передано їй як параметр ім'я файлу, і виводить його вміст на екран. Якщо додатковий аргумент use_include_path має значення TRUE, то пошук файлу з заданим ім'ям виробляється і по директоріях, що входять в include_path.

У програму ця функція повертає число лічених байтів (символів) файлу, а в разі помилки - FALSE. Повідомлення про помилку в цій функції можна придушити оператором @.

Приклад. Наступний скрипт виведе на екран вміст файлу my_file1.html і розмір цього файлу, якщо він існує. В іншому випадку виведеться наше повідомлення про помилку - рядок "Errorinreadfile".

```
<? Php
$ N = @ readfile ("my_file1.html");
/ * Виводить на екран вміст файлу і
записує його розмір у змінну $ n * /
if (! $ n) echo "Errorinreadfile";
/ * Якщо функція readfile () виконалася
з помилкою, то $ n = false і виводимо
```

повідомлення про помилку * /
elseecho $ n;
/ / Якщо помилки не було, то виводимо число
/ / Лічених символів
?>

Приклад. Використання функції readfile ()

За допомогою функції readfile () можна читати вміст віддалених файлів, вказуючи їх URL-адресу в якості імені файлу, якщо ця опція не відключена в налаштуваннях сервера.

Відразу ж виводити вміст файла на екран не завжди зручно. Часом потрібно записати інформацію з файлу в змінну, щоб надалі зробити з нею будь-які дії. Для цього можна використовувати функцію file () або file_get_contents ().

9.4.6 Функція file

Функція file () призначена для зчитування інформації з файлу в змінну типу масив. Синтаксис у неї такий же, як і у функції readfile (), за винятком того, що в результаті роботи вона повертає масив:

arrayfile (ім'я_файлу [Use_include_path])

Що за масив повертає ця функція? Кожен елемент даного масиву є рядком у файлі, інформацію з якого ми зчитуємо (його ім'я задано аргументом ім'я_файлу). Символ нового рядка теж включається в кожен з елементів масиву. У випадку помилки функція file (), як і всі вже розглянуті, повертає false. Додатковий аргумент use_include_path знову ж таки визначає, шукати чи ні даний файл в директоріях include_path. Відкривати вилучені файли за допомогою цієї функції теж можна, якщо не заборонено сервером.

Наприклад, у нас є файл my_file.html наступного змісту:

<h1> Без праці не виймеш
і рибку зі ставка. </ h1>
<b> Тихіше їдеш - далі будеш </ b>

Прочитаємо його вміст за допомогою функції file ():

<? Php
$ Arr = file ("my_file.html");

```
foreach ($ arras $ i => $ a) echo $ i, ":",
htmlspecialchars ($ a), "<br>";
?>
```

У результаті на екран буде виведено наступне повідомлення:

```
0: <h1> Без праці не виймеш
і рибку зі ставка. </ h1>
1: <b> Тихіше їдеш - далі будеш </ b>
```

Функція file_get_contents

Для порядку все одно наведемо її синтаксис:

```
stringfile_get_contents ( ім'я_файлу [, use_include_path])
```

Ця функція абсолютно ідентична функції file (), тільки повертає вона вміст файлу у вигляді рядка. Крім того, вона безпечна для обробки бінарних даних і може зчитувати інформацію з віддалених файлів, якщо це не заборонено налаштуваннями сервера.

9.5 Перевірка існування файлу

Отже, створювати файл ми навчилися, записувати дані в нього - навчилися, зчитувати дані з файлу - теж навчилися. Але от питання: а що якщо файл, з яким ми намагаємося виконати всі ці операції, не існує? Або він недоступний для читання або запису? Очевидно, що в такому разі жодна з вивчених нами функцій працювати не буде і PHP видасть повідомлення про помилку. Щоб відстежувати такого роду помилки, можна використовувати функції file_exists (), is_writable (), is_readable ().

9.5.1 Функція file_exists

Синтаксис:

boolfile_exists (ім'я файлу або директорії)

Функція file_exists () перевіряє, чи існує файл або директорія, ім'я якої передано їй як аргумент. Якщо директорія або файл у файловій системі сервера існує, то функція повертає TRUE, в іншому випадку - FALSE. Результат роботи цієї функції кешується. Відповідно очистити кеш можна, як уже зазначалося,

за допомогою функції clearstatcache (). Для нелокальних файлів використовувати функцію file_exists () не можна.

```
<? Php
$ Filename = 'c: / users / files / my_file.html';
if (file_exists ($ filename)) {
print "Файл <b> $ filename</ b> існує";
} Else {
print "Файл <b> $ filename</ b>
НЕ існує ";
}
?>
```

Приклад. Використання функції file_exists ()

9.5.2 Функція is_writable

Якщо крім перевірки існування файлу потрібно дізнатися ще, чи дозволено записувати інформацію в цей файл, слід використовувати функцію is_writable () або її псевдонім - функцію is_writeable ().

Синтаксис:

boolis_writable (ім'я файлу або директорії)

Ця функція повертає TRUE, якщо файл (або директорія) існує і доступний для запису. Доступ до файлу здійснюється під тим обліковим записом користувача, під яким працює сервер (найчастіше це користувач nobody або www). Результати роботи функції is_writablекешуються.

Функція is_readable Якщо крім перевірки існування файлу потрібно дізнатися ще, чи дозволено читати інформацію з нього, потрібно використовувати функцію is_readable (). Синтаксис: boolis_readable (ім'я файлу) Ця функція працює подібно до функції is_writable ().

```
<? Php
$ Filename = 'c: / users / files / my_file.html';
if (is_readable ($ filename)) {
print "Файл <b> $ filename</ b> існує
і доступний для читання ";
} Else {
```

print "Файл <b> $ filename</ b>
НЕ існує або
НЕ доступний для читання ";
}
?>

Приклад. Використання функції is_readable ()

9.6 Видалення файлу

Останнє, що ми хочемо вивчити з дій над файлами, - це видалення файлів. Для того щоб видалити файл за допомогою мови PHP, потрібно скористатися функцією unlink (). Синтаксис цієї функції можна описати таким чином:

boolunlink (ім'я_файлу)

Ця функція видаляє файл, який має ім'я ім'я_файлу, повертає TRUE у разі успіху цієї операції і FALSE - у разі помилки. Щоб видалити файл, потрібно теж мати відповідні права доступу до нього (наприклад, доступу тільки на читання для видалення файлу недостатньо).

<? Php
$ Filename = 'c: / users / files / my_file.html';
unlink ($ filename);
/ / Видаляємо файл з ім'ям
/ / C: / users / files / my_file.html
?>

Приклад. Використання функції unlink ()

9.7 Завантаження файлу на сервер

Тепер вирішимо більш складне і часто, виникаюче на практиці завдання завантаження файлу на сервер. Перше, що потрібно зробити, щоб завантажити файл на сервер, це створити html-форму. Для того щоб за допомогою цієї форми можна було завантажувати файли, вона повинна містити атрибут enctype в тезі form зі значенням multipart / form-data, а також елемент input типу file.

Приклад.

<Formenctype = "multipart / form-data"

action = "parse.php" method = "post">
<Input type = "hidden" name = "MAX_FILE_SIZE"
value = "30000" />
Завантажити файл: <input type = "file"
name = "myfile" />
<Input type = "submit"
value = "Надіслати файл" />
</ Form>

Приклад. Форма для завантаження файлу на сервер

Зауважимо, що ми додали у формі приховане поле, яке містить у собі максимальний допустимий розмір файлу, в байтах. При спробі завантажити файл, розмір якого більше зазначеного в цьому полі значення, буде зафіксована помилка. У браузері створена нами форма буде виглядати як рядок для введення тексту з додатковою кнопкою для вибору файлу з локального диска (рис 9.1).

Рис. 9.1. Приклад форми для завантаження файлу на сервер

Тепер потрібно написати скрипт, який буде обробляти отриманий файл.

Вся інформація про завантаженому на сервер файл міститься в глобальному масиві $ _FILES. Якщо включена директива register_globals, то значення переданих змінних доступні просто за їхніми іменами.

Якщо ми завантажили з комп'ютера-клієнта файл з ім'ям critics.htm розміром 15136 байт, то скрипт з єдиною командою print_r ($ _FILES); виведе на екран наступне:

Array ([myfile] =>
Array ([name] => critics.htm
[Type] =>text / html
[Tmp_name] => C: \ WINDOWS \ TEMP \ php49F.tmp
[Error] => 0

[Size] => 15136
))

Взагалі кажучи, масив $ _FILES завжди має такі елементи:

- $ _FILES ['myfile'] ['name'] - ім'я, яке мав файл на машині клієнта.
- $ _FILES ['myfile'] ['type'] - mime-тип відправленого файлу, якщо браузер надав цю інформацію. У нашому прикладі це text / html.
- $ _FILES ['myfile'] ['size'] - розмір завантаженого файлу в байтах.
- $ _FILES ['myfile'] ['tmp_name'] - тимчасове ім'я файлу, під яким він був збережений на сервері.
- $ _FILES ['myfile'] ['error'] - код помилки, що з'являється при завантаженні.

Тут 'myfile' - це ім'я елемента форми, за допомогою якого проведено завантаження файлу на сервер. Тобто воно може бути іншим, якщо елемент форми назвати інакше. Але от інші ключі (name, type і т. д.) залишаються незмінними для будь-якої форми.

Якщо register_globals = On, то доступні також додаткові змінні, такі як $ myfile_name, яка еквівалентна $ _FILES ['myfile'] ['name'], і т.п.

Помилок при завантаженні в PHP виділяють п'ять типів і відповідно $ _FILES ['myfile'] ['error'] може мати п'ять значень:

0 - помилки не сталося, файл завантажений успішно

1 - завантаження перевищує розмір, встановлений директивою upload_max_filesize у файлі php.ini

2 - завантаження перевищує розмір, встановлений елементом MAX_FILE_SIZE форми html

3 - файл був завантажений частково

4 - файл завантажений не був

За замовчуванням завантажені файли зберігаються в тимчасовій директорії сервера, якщо інша директорія не вказана за допомогою опції upload_tmp_dir у файлі налаштувань php.ini. Перемістити завантажений файл в потрібну директорію можна за допомогою функції move_uploaded_file ().

Функція move_uploaded_file () має наступний синтаксис:

boolmove_uploaded_file (тимчасове_імя_файла, місце_призначення)

Ця функція перевіряє, чи дійсно файл, позначений рядком тимчасове_імя_файла, був завантажений через механізм завантаження HTTP методом POST. Якщо це так, то файл буде перенесено в файл, заданий параметром місце_призначення (цей параметр містить як шлях до нової директорії для зберігання, так і нове ім'я файлу).

Якщо тимчасове_імя_файла задає неправильний завантажений файл, то ніяких дій вироблено не буде, і move_uploaded_file () поверне FALSE. Те ж саме станеться, якщо файл з якихось причин не може бути переміщений. У цьому випадку інтерпретатор виведе відповідне попередження. Якщо файл, заданий параметром місце_призначення, існує, то функція move_uploaded_file () перезапише його.

```
<?
$ Uploaddir = 'c: / uploads /';
/ / Будемо зберігати файли
/ / Файли в цю директорію
$ Destination = $ uploaddir.
$ _FILES ['Myfile'] ['name'];
/ / Ім'я файлу залишимо незмінним
print "<pre>";
if (move_uploaded_file (
$ _FILES ['Myfile'] ['tmp_name'],
$ Destination)) {
/ * Переміщаємо файл з тимчасової папки у вибрану директорію для зберігання * /
print "Файл успішно завантажений <br>";
} Else {
echo "Сталася помилка при завантаженні файлу.
Деяка налагоджувальна інформація: <br> ";
print_r ($ _FILES);
}
```

```
print "</ pre>";
?>
```

Приклад. Програма завантаження файлу на сервер

Розділ 3 ООП в PHP
Тема 10 Основні поняття ООП

10.1 Введення в ООП

Ми живемо у світі об'єктів. Стіл, автомобіль, ручка, класна дошка - все це об'єкти. Поряд з фізичними існують так само абстрактні об'єкти, типовими представниками яких є числа. Таким чином, об'єкт - це будь-яка фізична чи абстрактна сутність. Об'єкт - це загальнофілософське поняття, яке вивчалося філософами протягом тривалого часу.

Об'єкти характеризуються атрибутами. Так атрибутами автомобіля є максимальна швидкість, потужність двигуна, колір кузова й т.д. Атрибутами підсилювача є частотний діапазон, вихідна потужність, коефіцієнт нелінійних спотворень, рівень шуму і т. д.

Крім атрибутів об'єкти володіють деякими функціональними можливостями, які в об'єктно-орієнтованому програмуванні (ООП) називають **операціями** або **методами**. Так автомобіль може їздити, корабель - плавати, комп'ютер - робити обчислення.

Таким чином, об'єкт інкапсулює атрибути і методи, приховуючи від інших об'єктів взаємодіючих з ним і використовують його функціональність, свою реалізацію. Так для того щоб переключити телевізійну програму нам достатньо на пульті дистанційного керування набрати її номер, що запустить складний механізм, який у результаті і призведе до бажаного результату. Нам абсолютно не обов'язково знати, що відбувається в пульті дистанційного керування і телевізорі, нам лише достатньо знати, що телевізор має такою можливістю (методом) і як її можна активувати. Інкапсуляція або приховування реалізації є базовим властивістю ООП. Вона дозволяє створювати призначені для користувача об'єкти, що володіють необхідними методами і далі оперувати ними, не вдаючись у пристрій цих об'єктів.

Об'єкт - це екземпляр деякого класу об'єктів або просто класу. Так автомобіль Audi 6 є екземпляром класу автомобілів

даної моделі, приймач Sony SW-7600G так само буде представником класу однойменних приймачів. Таким чином, клас - це абстрактне поняття. Ставлення класу і об'єкту приблизно таке ж, як платонівські ідеї та об'єкти реального світу. На UML - уніфікованій мові моделювання - клас відображається у вигляді прямокутника, розділеного на три частини. У першій міститься ім'я класу, в другій - атрибути, в третій - методи.

Класи можуть бути пов'язані один з одним різними відносинами. Одним з основних таких відносин є ставлення клас - підклас, відомий в об'єктно-орієнтованому програмуванні як наслідування. Наприклад, клас автомобілів Audi 6 є підкласом легкових автомобілів, який в свою чергу входить у більший клас автомобілів, а останній є підкласом класу транспортних засобів, який крім автомобілів включає в себе літаки, кораблі поїзда і т. д. Прикладом подібних відносин, є системи класифікації в ботаніці та зоології. Ставленням, зворотним спадкоємству, є узагальнення або генералізація. Вона вказує, що якийсь клас, є більш загальним (узагальненим) класом іншого класу. Клас транспортних засобів, наприклад, є генералізацією класів автомобілів, літаків і кораблів. У UML прийнято користуватися саме поняттям генералізація, що відбилося і в символі, що представляє це ставлення: більш не зафарбована стрілка, спрямована на клас, що є узагальненням деяких класів (рис. 10.1, б).

При спадкуванні всі атрибути і методи батьківського класу успадковуються **класом-нащадком**. Спадкування може бути багаторівневим, і тоді класи, що знаходяться на нижніх рівнях ієрархії, успадкують всі властивості (атрибути і методи) всіх класів, прямими або непрямими нащадками яких вони є. Клас B успадкує атрибути і методи класу A і, отже, буде мати атрибутами A, B, C і D і методами A, B, C і D, а клас C - атрибутами A, B, C, E, F і методами A, B і E.

Крім одиничного, існує і **множинне** спадкування, коли клас успадковує відразу кілька класів (рис. 10.1, с). При цьому

він успадкує властивості всіх класів, нащадком яких вона є. Проте в мові PHP таке спадкування не реалізовано.

При спадкуванні одні методи класу можуть заміщатися іншими. Так, клас транспортних засобів буде мати узагальнений метод руху. У класах-нащадках цей метод буде конкретизований: автомобіль буде їздити, літак - літати, корабель - плавати. Така зміна семантики методу називається поліморфізмом. Поліморфізм - це виконання методом з одним і тим же ім'ям різних дій залежно від контексту, зокрема, від приналежності того чи іншого класу. У різних мовах програмування поліморфізм реалізується різними способами.

Іншим основним видом відносин між класами та об'єктами є агрегація. Вона означає, що один клас містить в собі агрегати (складових частин, підсистем) інших класів. Так автомобіль складається з кузова, двигуна, трансмісії і т.п., а до складу приймально-передавального пристрою входять передавач, приймач і антенно-фідерний пристрій. У UML агрегації позначаються у вигляді лінії з зафарбовані ромбом на кінці. Агрегація має кратність. Так автомобіль зазвичай містить один двигун, який у свою чергу може належати тільки одному автомобілю. Автомобіль може звичайно містити від двох до п'яти дверей. У свою чергу кожні двері можуть належати лише одному автомобілю.

Щоб звернутися до атрибутів і методів агрегату, необхідно спочатку отримати вказівник на його власника, а потім вже вибрати необхідні атрибути і методи.

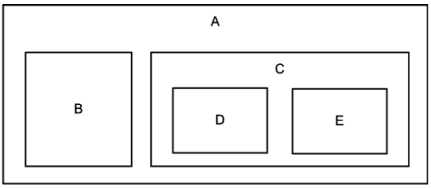
Хай об'єкт E має методи f1 () і f2 () (рис. 10.2). Щоб скористатися ними, треба спочатку отримати покажчик на кореневий об'єкт A, потім на об'єкт C, що в об'єктно-орієнтованому програмуванні зазвичай записується таким чином:

A.C

Далі отримуємо покажчик на D, так як він є агрегатом C, і, нарешті, викликаємо необхідні методи f1 () і f2 ():

A.C.D.f1 ()

A.C.D.f2 ()



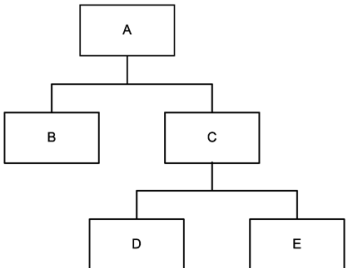


Рис. 10.2. Ієрархічне представлення вкладених об'єктів

Композиція є ще одним ставленням, родинною агрегації. Але якщо в агрегації агрегати належать класу чи об'єкту, то в композиції існує більш слабкий зв'язок. Так, студенти з вузом перебувають у відношенні композиції, тоді як факультети, які входять до складу вузу (тобто є його невід'ємною частиною або агрегатами), пов'язані з нею відношенням агрегації. На UML композиція позначається за допомогою незакрашених ромбів. Як і у випадку агрегації, ставлення композиції має кратність. Агрегація та композиція є підкласами класу відносин асоціації. Асоціація позначається у вигляді лінії без стрілок і ромбів і може приймати вигляд як агрегації, так і композиції. На ранніх етапах об'єктно-орієнтованого аналізу і проектування часто задаються відносини асоціацій, а свою конкретизацію у вигляді агрегацій і композицій вони отримують на більш пізніх етапах.

Існує хибна думка, що об'єктно-орієнтоване програмування є чимось складним і незрозумілим. Але об'єктна декомпозиція є нітрохи не менш природною і інтуїтивно зрозумілою, ніж алгоритмічна, яка неподільно панувала до появи ООП. У програмування основні поняття ООП перейшли з

інших галузей знань, таких як філософія, логіка, математика і семіотика, причому, не зазнавши особливих змін, принаймні того, що стосується суті цих понять. Об'єктний спосіб декомпозиції (подання) є природним, і застосовується протягом багатьох століть. Тому не дивно, що в процесі еволюції технології програмування він зайняв належне місце і підтримується так чи інакше практично всіма сучасними алгоритмічними мовами.

10.2 Класи і об'єкти в PHP

Термінологія

Клас. Тип даних, який визначається програмістом, який включає локальні функції і локальні дані. Клас може розглядатися як шаблон (або зразок, або форма) для створення будь-якої кількості примірників об'єкта одного і того ж типу (або класу).

Об'єкт (званий також примірником об'єкта або просто примірником). Окремий примірник структури даних, визначається класом. Визначення класу формулюється тільки один раз, після чого створюються всі необхідні об'єкти, які до нього належать.

Властивість (звана також змінна класу, атрибутом або полем-членом). Один з іменованих компонентів визначення даних у визначенні класу.

Метод (званий також функцією-членом). Компонент класу, який за своїм призначенням є функцією.

Абстракція - це надання об'єкту характеристик, які відрізняють його від усіх інших об'єктів, чітко визначаючи його концептуальні кордони. Основна ідея полягає в тому, щоб відокремити спосіб використання складових об'єктів даних від деталей їх реалізації у вигляді більш простих об'єктів, подібно до того, як функціональна абстракція поділяє спосіб використання функції і деталей її реалізації в термінах більш примітивних функцій, таким чином, дані обробляються функцією високого рівня за допомогою виклику функцій низького рівня. Такий підхід є основою об'єктно-орієнтованого

програмування. Це дозволяє працювати з об'єктами, не вдаючись в особливості їх реалізації. У кожному конкретному випадку застосовується той чи інший підхід: інкапсуляція, поліморфізм або спадкування. Наприклад, при необхідності звернутися до прихованих даних об'єкту, слід скористатися інкапсуляцією, створивши, так звану, функцію доступу або властивість.

Успадкування. Процес визначення класу на основі іншого класу. На новий (дочірній) клас за замовчуванням поширюються всі визначення змінних екземпляра і методів із старого (батьківського) класу, але можуть бути також визначені нові компоненти або «перевизначені» визначення батьківських функцій і дано нові визначення. Прийнято вважати, що клас А успадковує свої визначення від класу В, якщо клас А визначений на основі класу В зазначеним способом.

Поліморфізм - так називають явище, при якому функції (методу) з одним і тим же ім'ям відповідає різний програмний код (поліморфний код) в залежності від того, об'єкт якого класу використовується при виклику даного методу. Поліморфізм забезпечується тим, що в класі-нащадку змінюють реалізацію методу класу-предка з обов'язковим збереженням сигнатури методу. Це забезпечує збереження незмінного інтерфейсу класу-предка і дозволяє здійснити зв'язування імені методу в коді з різними класами - з об'єкта якого класу здійснюється виклик, з того класу і береться метод з такою назвою. Такий механізм називається динамічним (або пізнім) зв'язуванням - на відміну від статичного (раннього) зв'язування, здійснюваного на етапі компіляції.

Інкапсуляція - це принцип, згідно з яким будь-який клас повинен розглядатися як чорний ящик - користувач класу повинен бачити і використовувати тільки інтерфейсну частину класу (тобто список декларованих властивостей і методів класу) і не вникати в його внутрішню реалізацію. Тому дані прийнято інкапсулювати в класі таким чином, щоб доступ до них з читання або запису здійснювався не безпосередньо, а за допомогою методів. Принцип інкапсуляції (теоретично)

дозволяє мінімізувати кількість зв'язків між класами і, відповідно, спростити незалежну реалізацію та модифікацію класів.

Приховання даних - невіддільна частина ООП, керуюча областями видимості. Є логічним продовженням інкапсуляції. Метою приховування є неможливість для користувача дізнатися або зіпсувати внутрішній стан об'єкта.

Батьківський клас (або суперклас, або базовий клас). Клас, визначення якого успадковані іншим класом.

Дочірній клас (або підклас, або похідний клас). Клас, який успадковує свої визначення від іншого класу.

Конструктор - це той же метод об'єкта, який викликається автоматично при його створенні. Як правило, конструктори використовуються для присвоєння первинних значень властивостей об'єкта. В PHP4 ім'я конструктора повинно відповідати імені класу.

Деструкція - аналогічний конструктору за винятком того, що викликається при знищенні об'єкта. Наприклад в деструктор може бути вбудована функція розриву з'єднання з базою даних або збереження файлу. Деструктори не присутні в PHP4, але в 5 версії вони є. Ім'я методу деструктора заздалегідь визначено - _destructor.

10.3 Синтаксис

10.3.1 Опис класу і створення об'єкта

Синтаксис для створення класу досить простий: необхідно оголосити клас, використовуючи ключове слово class, за яким слідує назва класу і безліч фігурних дужок ({}):

```
<? Php
classMyClass
{
/ / Методи класу
}
?>
```

Після створення класу, новий клас може бути створено та збережено у змінній за допомогою ключового слова new:

$ Obj = newMyClass

Щоб побачити вміст класу, використовується функція var_dump ():

var_dump ($ Obj);

Давайте зараз спробуємо з'єднати весь перерахований вище код і додати його до вашого test.php на вашому локальному сервері.

```
<? Php
classMyClass
{
/ / Методи класу
}
$ Obj = newMyClass;
var_dump ($ obj);
?>
```

Завантажте сторінку в браузері за адресою http://localhost/test.php На дисплеї має висвітитися це:

object (MyClass) # 1 (0) {}

Можна сказати, що ви створили найпростіші програми використовуючи ООП

10.3.2 Визначення властивостей класу

Щоб додати до класу властивості, потрібно переробити наш клас приблизно так:

```
<? Php
classMyClass
{
public $ prop1 = "I'm a classproperty!";
}
$ Obj = newMyClass;
var_dump ($ obj);
?>
```

Властивості класу - це специфічні змінні, які пов'язані з об'єктом класу і можуть бути доступні виключно через клас.

public - це атрибут видимості даної змінної, про які ми поговоримо трохи пізніше. Після нього слідує змінна, якій присвоюється значення "I'm a classproperty!";

Для виведення даної властивості у вікно браузера, необхідно написати:

echo $ obj-> prop1;

Оскільки може існувати кілька об'єктів класу, визначити до якого класу яка властивість відноситься можна за допомогою "->".

Давайте змінимо test.php замінивши попередній код на цей:

```
<? Php
classMyClass
{
public $ prop1 = "I'm a classproperty!";
}
$ Obj = newMyClass;
echo $ obj-> prop1; / / Outputtheproperty
?>
```

Тепер в браузері ми побачимо:

I'm a classproperty!

10.3.3 Псевдо-змінна $ this

Псевдо-змінна $ this доступна в тому випадку, коли метод був викликаний в контексті об'єкта. $ This є посиланням на об'єкт, що викликається. Зазвичай це той об'єкт, якому належить викликаний метод, але може бути й інший об'єкт, якщо метод був викликаний статично з контексту іншого об'єкта. Це показано на наступних прикладах:

Приклад змінна $ this в об'єктно-орієнтованій мові

```
<? Php
class A
{
functionfoo ()
{
if (isset ($ this)) {
```

```
echo '$ this визначена (';
echoget_class ($ this);
echo ") \ n";
} Else {
echo "\ $ this не визначена. \ n";
}
}
}
class B
{
functionbar ()
{
A:: foo ();
}
}
$ A = new A ();
$ A->foo ();
A:: foo ();
$ B = new B ();
$ B->bar ();
B:: bar ();
?>
```

Результат виконання цього прикладу:

```
$ This визначена (a)
$ This не визначена.
$ This визначена (b)
$ This не визначена.
```

10.3.4 Визначення методів класу

Методи - це специфічні функції класу, які будуть виконувати деякі дії в класі.

Наприклад, для створення методів, які будуть встановлювати і отримувати значення властивостей класу $ Prop1, додайте наступні рядки до вашого коду:

```
<? Php
classMyClass
```

```
{
public $ prop1 = "I'm a classproperty!";
publicfunctionsetProperty ($ newval)
{
$ This-> prop1 = $ newval;
}
publicfunctiongetProperty ()
{
return $ this-> prop1. "
";
}
}
$ Obj = newMyClass;
echo $ obj-> prop1;
?>
```

Примітка:

Для виклику методів класу не забувайте використовувати знак долара.

Приклад $ obj->set ();

Тепер давайте спробуємо скористатися написаним вище класом шляхом модифікації файлу test.php:

```
<? Php
classMyClass
{
public $ prop1 = "I'm a classproperty!";
publicfunctionsetProperty ($ newval)
{
$ This-> prop1 = $ newval;
}
publicfunctiongetProperty ()
{
return $ this-> prop1. "";
}
}
$ Obj = newMyClass;
```

```
echo $ obj->getProperty (); / / Gettheproperty value
$ Obj->setProperty ("I'm a newpropertyvalue!"); / / Set a newone
echo $ obj->getProperty (); / / Readitoutagaintoshowthechange
?>
```

У браузері ви побачите:

I'm a classproperty!

I'm a newpropertyvalue!

"Справжню силу ООП можна побачити при створенні декількох об'єктів одного і того ж класу."

```
<? Php
classMyClass
{
public $ prop1 = "I'm a classproperty!";
publicfunctionsetProperty ($ newval)
{
$ This-> prop1 = $ newval;
}
publicfunctiongetProperty ()
{
return $ this-> prop1. "
";
}
}
/ / Createtwoobjects
$ Obj = newMyClass;
$ Obj2 = newMyClass;
/ / Getthevalueof $ prop1 frombothobjects
echo $ obj->getProperty ();
echo $ obj2->getProperty ();
/ / Setnewvaluesforbothobjects
$ Obj->setProperty ("I'm a newpropertyvalue!");
$ Obj2->setProperty ("I belongtothesecondinstance!");
/ / Outputbothobjects '$ prop1 value
```

```
echo $ obj->getProperty ();
echo $ obj2->getProperty ();
?>
```

Після замени вмісту файлу test.php ми побачимо:

```
I'm a classproperty!
I'm a classproperty!
I'm a newpropertyvalue!
I belongtothesecondinstance!
```

10.3.5 Використання конструкторів і деструкторів

До таких методів відноситься метод _construct (), що дозволяє задати будь-які дії при створенні об'єкта, і метод _destruct (), що дозволяє задати дії при видаленні об'єкта (наприклад: розірвати з'єднання з базою даних). Для наочності, візьмемо приклад:

```
<? Php
classMyClass
{
public $ prop1 = "I'm a classproperty!";
publicfunction _construct ()
{
echo 'Theclass ', " _CLASS_ ", ' wasinitiated! ';
}
publicfunction _destruct ()
{
echo 'Theclass '", _CLASS_, '" wasdestroyed. ';
}
publicfunctionsetProperty ($ newval)
{
$ This-> prop1 = $ newval;
}
publicfunctiongetProperty ()
{
return $ this-> prop1. "";
}
}
```

```
/ / Створення нового об'єкта
$ Obj = newMyClass;
/ / Отримуємо значення $ prop1
echo $ obj->getProperty ();
/ / Виводимо повідомлення про закінчення файлу
echo "Endoffile.";
?>
```

Після виконання даного коду ми побачимо наступний результат:

```
Theclass "MyClass" wasinitiated!
I'm a classproperty!
Endoffile.
Theclass "MyClass" wasdestroyed.
```

Хотілося б звернути вашу увагу, що після завершення виконання скрипта, PHP автоматичні звільняє пам'ять.

Примітка:

Константа _CLASS_ використовується для отримання імені класу, в якому вона викликається.

Для того, щоб знищити об'єкт вручну, нам необхідно скористатися функцією unset ()

Невеликий **приклад**:

```
<? Php
classMyClass
{
public $ prop1 = "I'm a classproperty!";
publicfunction _construct ()
{
echo 'Theclass '', _CLASS_, '' wasinitiated! ';
}
publicfunction _destruct ()
{
echo 'Theclass '', _CLASS_, '' wasdestroyed. ';
}
publicfunctionsetProperty ($ newval)
```

```
{
$ This-> prop1 = $ newval;
}
publicfunctiongetProperty ()
{
return $ this-> prop1. "";
}
}
/ / Створення об'єкта
$ Obj = newMyClass;
/ / Отримання значення $ prop1
echo $ obj->getProperty ();
[B] / / Знищуємо об'єкт [/ b]
unset ($ obj);
/ / Висновок повідомлення про закінчення файлу
echo "Endoffile.";
?>
```

Результат виконання скрипта буде наступним:

```
Theclass "MyClass" wasinitiated!
I'm a classproperty!
Theclass "MyClass" wasdestroyed.
Endoffile.
```

Конструктори в класах-батьків не викликаються автоматично. Щоб викликати конструктор, оголошений в батьківському класі, слід звернутися до методу parent:: _construct ().

10.4 Успадкування

Успадкування - один з чотирьох найважливіших механізмів об'єктно-орієнтованого програмування (поряд з інкапсуляцією, поліморфізмом і абстракцією), що дозволяє описати новий клас на основі вже існуючого (батьківського), при цьому властивості і функціональність батьківського класу запозичуються новим класом.

Говорячи простими словами спадкування, це такий механізм, який дозволяє розширювати клас за рахунок методів іншого класу. Для того, щоб додати методи і властивості іншого класу, необхідно скористатися словом extends. Наприклад, щоб створити другий клас, який розширює MyClass і додає методи, ви повинні додати наступні рядки в ваш тестовий файл:

```
<? Php
classMyClass
{
public $ prop1 = "I'm a classproperty!";
publicfunction _construct ()
{
echo 'Theclass '', _CLASS_, '' wasinitiated! ';
}
publicfunction _destruct ()
{
echo 'Theclass '', _CLASS_, '' wasdestroyed.';
}
publicfunction _toString ()
{
echo "UsingthetoStringmethod:";
return $ this->getProperty ();
}
publicfunctionsetProperty ($ newval)
{
$ This-> prop1 = $ newval;
}
publicfunctiongetProperty ()
{
return $ this-> prop1. "";
}
}
classMyOtherClassextendsMyClass
{
publicfunctionnewMethod ()
{
```

```
echo "From a newmethodin". _CLASS_. ". ";
}
}
/ / Створення об'єкта
$ Newobj = newMyOtherClass;
/ / Виводимо об'єкт
echo $ newobj->newMethod ();
/ / Використання методу батьківського класу
echo $ newobj->getProperty ();
?>
```

10.5 Заміна успадкованих методів і властивостей

Для того, щоб замінити успадкований метод батьківського класу в новому класі, необхідно просто замінити цей метод в новому класі. При цьому необхідно використовувати те ж ім'я, яке було в батьківському класі:

```
<? Php
classMyClass
{
public $ prop1 = "I'm a classproperty!";
publicfunction _construct ()
{
echo 'Theclass '", _CLASS_, '" wasinitiated!';
}
publicfunction _destruct ()
{
echo 'Theclass '", _CLASS_, '" wasdestroyed.';
}
publicfunction _toString ()
{
echo "UsingthetoStringmethod:";
return $ this->getProperty ();
}
publicfunctionsetProperty ($ newval)
{
$ This-> prop1 = $ newval;
```

```
}
publicfunctiongetProperty ()
{
return $ this-> prop1. "";
}
}
classMyOtherClassextendsMyClass
{
publicfunction _construct ()
{
echo "A newconstructorin". _CLASS_. ".";
}
publicfunctionnewMethod ()
{
echo "From a newmethodin". _CLASS_. ". ";
}
}
/ / Створення об'єкта
$ Newobj = newMyOtherClass;
/ / Виводимо об'єкт
echo $ newobj->newMethod ();
/ / Використання методу батьківського класу
echo $ newobj->getProperty ();
?>
```

У результаті ми отримаємо:
A newconstructorinMyOtherClass.
From a newmethodinMyOtherClass.
I'm a classproperty!
Theclass "MyClass" wasdestroyed.

Існують ситуації, коли необхідно замінивши метод у новому класі, викликати цей же метод, але тільки в батьківському класі. У таких випадках використовується оператор parent::

Приклад:

```
<? Php
classMyClass
{
public $ prop1 = "I'm a classproperty!";
publicfunction _construct ()
{
echo 'Theclass '", _CLASS_, '" wasinitiated! ';
}
publicfunction _destruct ()
{
echo 'Theclass '", _CLASS_, '" wasdestroyed. ';
}
publicfunction _toString ()
{
echo "UsingthetoStringmethod:";
return $ this->getProperty ();
}
publicfunctionsetProperty ($ newval)
{
$ This-> prop1 = $ newval;
}
publicfunctiongetProperty ()
{
return $ this-> prop1. "";
}
}
classMyOtherClassextendsMyClass
{
publicfunction _construct ()
{
parent:: _construct ();
echo "A newconstructorin". _CLASS_. ".";
}
publicfunctionnewMethod ()
{
```

```
echo "From a newmethodin". _CLASS_. ". ";
}
}
/ / Створюємо об'єкт
$ Newobj = newMyOtherClass;
/ / Виводимо об'єкт
echo $ newobj->newMethod ();
/ / Використання методу батьківського класу
echo $ newobj->getProperty ();
?>
```

У результаті у вікні браузера можна спостерігати:

```
Theclass "MyClass" wasinitiated!
A newconstructorinMyOtherClass.
From a newmethodinMyOtherClass.
I'm a classproperty!
Theclass "MyClass" wasdestroyed.
```

Тема 11 Поглиблення в ООП

11.1 Модифікатори доступу

Модифікатори доступу - це, по суті, інтерпретація інкапсуляції в ООП. Нагадую, що інкапсуляція - це механізм приховування реалізації об'єкта. І для реалізації інкапсуляції існують модифікатори доступу в PHP.

У PHP є три модифікатори доступу:

1. Public
2. Protected
3. Private

Почнемо з модифікаторів доступу **public.** Даний модифікатор означає, що властивість, метод або конструктор будуть доступні для всіх об'єктів, які їх використовують.

Модифікатор доступу **protected** означає, що даний елемент об'єкта може бути використаний тільки в самому об'єкті, а також у його дочірніх.

І, нарешті, модифікатор доступу **private** означає, що даний елемент об'єкта може бути використаний тільки в самому об'єкті і ніде більше.

11.1.1 Область видимості властивостей

Властивості класу повинні бути визначені через модифікатори public, private, або protected.

Приклад Оголошення властивості класу

```
<? Php
/ **
* Визначення MyClass
* /
class MyClass
{
public $ public = 'Загальний';
protected $ protected = 'Захищений';
private $ private = 'Закритий';

function printHello ()
```

```
{
echo $ this-> public;
echo $ this-> protected;
echo $ this-> private;
}
}

$ Obj = new MyClass ();
echo $ obj-> public; / / Працює
echo $ obj-> protected; / / Невиправна помилка
echo $ obj-> private; / / Невиправна помилка
$ Obj-> printHello (); / / Виводить Загальний, Захищений і Закритий

/ **
* Визначення MyClass2
* /
class MyClass2 extends MyClass
{
/ / Ми можемо перевизначити public і protected методи, але не private
protected $ protected = 'Захищений2';

function printHello ()
{
echo $ this-> public;
echo $ this-> protected;
echo $ this-> private;
}
}

$ Obj2 = new MyClass2 ();
echo $ obj2-> public; / / Працює
echo $ obj2-> private; / / Невизначений
echo $ obj2-> protected; / / Невиправна помилка
```

$ Obj2-> printHello (); / / Виводить Загальний, Захищений2 і Закритий
?>

11.1.2 Область видимості методу

Методи класу повинні бути визначені через модифікатори public, private, або protected. Методи, де визначення модифікатора відсутня, визначаються як public.

Приклад Оголошення методу

```
<? Php
/ **
* Визначення MyClass
* /
class MyClass
{
/ / Оголошення загальнодоступного конструктора
public function _construct () {}

/ / Оголошення загальнодоступного методу
public function MyPublic () {}

/ / Оголошення захищеного методу
protected function MyProtected () {}

/ / Оголошення закритого методу
private function MyPrivate () {}

/ / Це загальнодоступний метод
function Foo ()
{
$ This-> MyPublic ();
$ This-> MyProtected ();
$ This-> MyPrivate ();
}
}
```

```
$ Myclass = new MyClass;
$ Myclass-> MyPublic (); / / Працює
$ Myclass-> MyProtected (); / / Невиправна помилка
$ Myclass-> MyPrivate (); / / Невиправна помилка
$ Myclass-> Foo (); / / Працює загальний, захищений і закритий

/ **
* Визначення MyClass2
* /
class MyClass2 extends MyClass
{
/ / Це загальнодоступний метод
function Foo2 ()
{
$ This-> MyPublic ();
$ This-> MyProtected ();
$ This-> MyPrivate (); / / Невиправна помилка
}
}

$ Myclass2 = new MyClass2;
$ Myclass2-> MyPublic (); / / Працює
$ Myclass2-> Foo2 (); / / Працює загальний і захищений, закритий не працює

class Bar
{
public function test () {
$ This-> testPrivate ();
$ This-> testPublic ();
}

public function testPublic () {
echo "Bar:: testPublic \ n";
}
```

```
private function testPrivate () {
echo "Bar:: testPrivate \ n";
}
}

class Foo extends Bar
{
public function testPublic () {
echo "Foo:: testPublic \ n";
}

private function testPrivate () {
echo "Foo:: testPrivate \ n";
}
}

$ MyFoo = new foo ();
$ MyFoo-> test (); / / Bar:: testPrivate
/ / Foo:: testPublic
?>
```

11.1.3 Статичні методи і властивості

Статичні методи класу можуть бути спричинені безпосередньо у класі, а не через один із його об'єктів. Відповідно, покажчик $ this в статичних методах недоступний.

Фактично, оголошення класу зі статичними методами є, більшою мірою, методом угруповання функцій і загальних для них констант і змінних.

Застосування такого підходу гарантує, що всі класи доступу до бази даних будуть реалізовувати один інтерфейс, зменшується вірогідність конфліктності імен, спрощується існування декількох версій класу доступу до бази і т.д.

```
<? Php
class MyClass {
static function helloWorld () {
```

```
print "Hello, world";
}
}
MyClass:: helloWorld ();
?>
```

Загальне використання статичних членів показано на прикладі:

```
<? Php
class Singleton {
static private $ instance = NULL;

private function _construct () {
}

static public function getInstance () {
if (self:: $ instance == NULL) {
self:: $ instance = new Singleton ();
}
return self:: $ instance;
}
}
?>
```

11.1.4 Подвійна двокрапка

"Paamayim Nekudotayim" або просто "подвійна двокрапка". Використовуючи цю лексему, програміст може звертатися до констант, статичним або перевантаженим властивостями або методів класу.

При зверненні до цих елементів ззовні класу, програміст повинен використовувати ім'я цього класу.

"Paamayim Nekudotayim" на перший погляд може здатися дивним словосполученням для позначення подвійної двокрапки. Однак, під час створення Zend Engine версії 0.5 (який входив до PHP3), Andi і Zeev вибрали саме це позначення. "Paamayim Nekudotayim" дійсно означає "подвійна двокрапка". Просто це

позначення не змінювалося жодного разу протягом усього часу розробки PHP.

Приклад Використання :: поза оголошення класу

```
<? Php
class MyClass {
const CONST_VALUE = 'Значення константи';
}
echo MyClass:: CONST_VALUE;
?>
```

Для звернення до властивостей і методів в оголошенні класу використовуються ключові слова self і parent.

Приклад Використання :: в оголошенні класу

```
<? Php
class OtherClass extends MyClass {
public static $ my_static = 'статична змінна';
public static function doubleColon () {
echo parent:: CONST_VALUE. "\ N";
echo self:: $ my_static. "\ N";
}
}
OtherClass:: doubleColon ();
?>
```

Коли дочірній клас перевантажує методи, оголошені в класі-батьку, PHP не буде здійснювати автоматичний виклик методів, що належать класу-батькові. Цей функціонал покладається на метод, перевантажується в дочірньому класі. Це правило поширюється на конструктори і деструктори, перевантажені та "чарівні" методи.

Приклад Звернення до методу в батьківському класі

```
<? Php
class MyClass {

protected function myFunc () {
echo "MyClass:: myFunc () \ n";
}
```

```
}

class OtherClass extends MyClass {

/ * Override parent's definition * /
public function myFunc () {

/ * But still call the parent function * /
parent:: myFunc ();
echo "OtherClass:: myFunc () \ n";
}
}

$ Class = new OtherClass ();
$ Class-> myFunc ();
?>
```

11.1.5 Оператор instanceof

Підтримка перевірки залежності від інших об'єктів. Функцією is_a (), яка міститься в PHP 4, користуватися тепер не рекомендується.

```
<? Php
if ($ obj instance of Circle) {
print '$ obj is a Circle';
}
?>
```

11.2 Фінальні методи і класи

11.2.1 Метод final

Ключове слово final дозволяє вам позначати методи, щоб наслідуючий клас не міг перевантажити їх. Розмістивши перед оголошенням методів або властивостей класу ключове слово final, ви можете запобігти їх перевизначення в дочірніх класах, наприклад:

```
<? Php
class BaseClass {
```

```
public function test () {
echo "Викликаний метод BaseClass:: test () \ n";
}

final public function moreTesting () {
echo "Викликаний метод BaseClass:: moreTesting () \ n";
}
}

class ChildClass extends BaseClass {
public function moreTesting () {
echo "Викликаний метод ChildClass:: moreTesting () \ n";
}
}
/ / Виконання закінчується фатальною помилкою:
/ / Cannot override final method BaseClass:: moreTesting ()
/ / (Метод BaseClass:: moretesting () не може бути перевизначений)
?>
```

11.2.2 Класи, помічені як final

Після оголошення класу final він не може бути успадкований. Наступний приклад викличе помилку:

```
<? Php
final class FinalClass {
}

class BogusClass extends FinalClass {
}
?>
```

11.3 Абстрактні класи та методи

11.3.1 Абстрактні класи

PHP 5 підтримує визначення абстрактних класів і методів. Створювати екземпляр класу, який був оголошений абстрактним, не можна. Клас, в якому оголошено хоча б один

абстрактний метод, повинен також бути оголошений абстрактним. Методи, оголошені як абстрактні, несуть, по суті, лише описовий зміст і не можуть включати будь-який функціонал. Клас може бути оголошений як абстрактний за допомогою використання ключового слова abstract, для виключення з обробки движком опису класу. Однак, ви можете наслідувати абстрактні класи. Практичний приклад:

```
<? Php
abstract class AbstractClass {
/ * Даний метод повинен бути визначений в дочірньому класі * /
abstract protected function getValue ();
/ * Загальний метод * /
public function print () {
print $ this-> getValue ();
}
}

class ConcreteClass1 extends AbstractClass {
protected function getValue () {
return "ConcreteClass1";
}
}

class ConcreteClass2 extends AbstractClass {
protected function getValue () {
return "ConcreteClass2";
}
}

$ Class1 = new ConcreteClass1;
$ Class1-> print ();
$ Class2 = new ConcreteClass2;
$ Class2-> print ();
?>
```

11.3.2 Абстрактні методи

Метод може бути оголошений як abstract, таким чином відклавши його визначення спадкоємним класом. Клас, який включає абстрактні методи, повинен бути оголошений як abstract.

```
<? Php
abstract class MyBaseClass {
abstract function display ();
}
?>
```

11.3.3 Інтерфейсні методи

У PHP 5 з'явилася така чудова можливість ООП, як інтерфейс класів. Інтерфейс визначає методи, які повинен містити клас, який реалізує інтерфейс. Всі методи, визначені у інтерфейсі повинні бути реалізовані. Інтерфейс створюється за допомогою ключового слова interface. Всі методи в інтерфейсі повинні бути публічними.

```
interface CanWalk {
public function walk ();
}
```

Клас може реалізовувати декілька інтерфейсів. Перерахування інтерфейсів виконується після ключового слова implements:

```
class Bird implements CanWalk, CanRun, CanFly {
public function walk () {
echo "Walk!";
}
public function run () {
echo "Run!";
}
public function fly () {
echo "Fly!";
}
}
```

Інтерфейс - це семантична і синтаксична конструкція в коді програми, яка використовується для послуг, що надаються класом чи компонентом. Інтерфейс визначає межу взаємодії між класами або компонентами, специфікуючи певну абстракцію, яку здійснює сторона. На відміну від багатьох інших видів інтерфейсів, інтерфейс в ООП є строго формалізованим елементом об'єктно-орієнтованої мови і, як семантичної конструкції, широко використовується кодом програми.

Таким чином, з одного боку, **інтерфейс** - це контракт, який зобов'язується виконати клас, який реалізує його, з іншого боку, **інтерфейс** - це тип даних, тому що його опис досить чітко визначає властивості об'єктів, щоб нарівні з класом типізувати змінні. Слід, однак, підкреслити, що інтерфейс не є повноцінним типом даних, так як він задає тільки зовнішню поведінку об'єктів. Внутрішню структуру і реалізацію заданого інтерфейсом поведінку забезпечує клас, який реалізує інтерфейс; саме тому «прикладів інтерфейсу» в чистому вигляді не буває, і будь-яка змінна типу «інтерфейс» містить екземпляри конкретних класів.

11.3.4 Інтерфейси та абстрактні класи

Можна помітити, що інтерфейс, з точки зору реалізації - це просто чистий абстрактний клас, тобто клас, в якому не визначено нічого, крім абстрактних методів. Якщо мова програмування підтримує множинне успадкування і абстрактні методи (як, наприклад, C + +), то необхідність у введенні в синтаксис мови, окремого поняття «інтерфейс» не виникає. Дані сутності описуються за допомогою абстрактних класів і успадковуються класами для реалізації абстрактних методів.

Однак підтримка множинного спадкоємства в повному обсязі досить складна і викликає безліч проблем, як на рівні реалізації мови, так і на рівні архітектури додатків. Введення поняття інтерфейсів є компромісом, що дозволяє отримати багато переваг множинного спадкування (зокрема, можливість зручно визначати логічно пов'язані набори методів у вигляді сутностей, подібних класам, допускати спадкування і реалізацію), не

реалізовуючи його в повному обсязі і не стикаючись, таким чином, з більшістю викликаних ним труднощів.

11.4 Магічні методи

Імена методів _construct, _destruct (див. Конструктори і деструктори), _call, _callStatic, _get, _set, _isset, _unset, _sleep, _wakeup, _toString, _set_state і _clone зарезервовані для "магічних" методів в PHP. Не варто називати свої методи цими іменами, якщо ви не хочете використовувати їх "магічну" функціональність.

Застереження

PHP залишає за собою право на всі методи, що починаються з _, вважати "магічними". Не рекомендується використовувати імена методів з _ у PHP, якщо ви не бажаєте використовувати відповідний "магічний" функціонал.

11.4.1 _call

З PHP 5 ви можете реалізувати в класі спеціальний метод _call (), як метод для "вилову" всіх нереалізованих у даному класі методів. Метод _call (якщо він визначений) викликається при спробі викликати недоступний або неіснуючий метод.

```
<? Php
class foo {
function _call ($ name, $ arguments) {
print ("Викликали? Я - $ name!");
}
}
$ X = new foo ();
$ X-> doStuff ();
$ X-> fancy_stuff ();
?>
```

Цей спеціальний метод може бути використаний для реалізації перевантаження методів: ви можете досліджувати отримані аргументи і залежно від результату викликати підходящий для даного випадку закритий метод, наприклад:

```
<? Php
```

```
class Magic {

function _call ($ name, $ arguments) {
if ($ name == 'foo') {
if (is_int ($ arguments [0])) $ this-> foo_for_int ($ arguments [0]);
if (is_string ($ arguments [0])) $ this-> foo_for_string ($ arguments [0]);
}
}

private function foo_for_int ($ x) {
print ("o, дивіться, ціле число!");
}

private function foo_for_string ($ x) {
print ("o, дивіться, рядок!");
}
}

$ X = new Magic ();
$ X-> foo (3);
$ X-> foo ("3");
?>
```

11.4.2 _set і _get

Але це ще не все, тепер ви можете визначити методи _set і _get для пошуку всіх спроб зміни або доступу до невизначених (або недоступних) змінних.

```
<? Php
class foo {

function _set ($ name, $ val) {
print ("Привіт, ви спробували привласнити значення $ val змінної $ name");
}
```

```
function _get ($ name) {
print ("Привіт, ви намагалися звернутися до $ name");
}
}

$ X = new foo ();
$ X-> bar = 3;
print ($ x-> winky_winky);
?>
```

11.4.3 _sleep і _wakeup

Функція **serialize ()** перевіряє, чи присутній у вашому класі метод з "магічним" іменем _sleep. Він може очистити об'єкт і передбачається, що буде повернутий масив з іменами всіх змінних об'єкта, який повинен бути серіалізований. Якщо метод нічого не повертає крім NULL, то це означає, що об'єкт серіалізований і видається попередження E_NOTICE.

Зазвичай **_sleep** використовується для передачі очікуваних даних або для виконання звичайних завдань їх очищення. Також, цей метод можна виконувати в тих випадках, коли ви не хочете зберігати дуже великі об'єкти повністю.

З іншого боку, функція **unserialize ()** перевіряє наявність методу з "магічним" іменем **_wakeup**. Якщо такий є, то він може відтворити всі ресурси об'єкта, які той має.

Зазвичай **_wakeup** використовується для відновлення будь-яких з'єднань з базою даних, які могли бути втрачені під час операції серіалізації та виконання інших операцій повторної ініціалізації.

Приклад Sleep і wakeup

```
<? Php
class Connection {
protected $ link;
private $ server, $ username, $ password, $ db;
```

```
public function _construct ($ server, $ username, $ password, $ db)
{
$ This-> server = $ server;
$ This-> username = $ username;
$ This-> password = $ password;
$ This-> db = $ db;
$ This-> connect ();
}

private function connect () {
$ This-> link = mysql_connect ($ this-> server, $ this-> username, $ this-> password);
mysql_select_db ($ this-> db, $ this-> link);
}

public function _sleep () {
return array ('server', 'username', 'password', 'db');
}

public function _wakeup () {
$ This-> connect ();
}
}
?>
```

11.4.4 _toString

Метод _toString дозволяє класу вирішувати самостійно, як він повинен реагувати при перетворенні в рядок.

Приклад Простий приклад

```
<? Php
/ / Декларування простого класу
class TestClass
{
public $ foo;
```

```
public function _construct ($ foo) {
$ This-> foo = $ foo;
}

public function _toString () {
return $ this-> foo;
}
}

$ Class = new TestClass ('Привіт');
echo $ class;
?>
```

Результат виконання цього прикладу:

Привіт

Раніше, до PHP 5.2.0, метод _toString викликався тільки безпосередньо в поєднанні з функціями echo () або print (). Починаючи з PHP 5.2.0, він викликається в будь-якому рядковому контексті (наприклад, у printf () з модифікатором% s), але не в контекстах інших типів (наприклад, з% d модифікатором). Починаючи з PHP 5.2.0, перетворення об'єкта в рядок за відсутності методу _toString викликає помилку E_RECOVERABLE_ERROR.

_invoke

Метод _invoke викликається при спробі використовувати змінну-об'єкт як функцію.

Зауваження: Доступно тільки з версії PHP 5.3.0.

Приклад Using _invoke

```
<? Php
class CallableClass {
function _invoke ($ x) {
var_dump ($ x);
}
}
$ Obj = new CallableClass;
$ Obj (5);
```

```
var_dump (is_callable ($ obj));
?>
```

Результат виконання цього прикладу:

```
int (5)
bool (true)
```

11.4.5 _set_state

Цей статичний метод викликається для тих класів, які експортуються функцією **var_export ()** починаючи з PHP 5.1.0.

Параметр цього методу повинен містити масив, що складається з експортованих властивостей у вигляді array ('property' => value, ...).

Приклад Використання _set_state (починаючи з PHP 5.1.0)

```
<? Php
class A
{
public $ var1;
public $ var2;
public static function _set_state ($ an_array) / / З PHP 5.1.0
{
$ Obj = new A;
$ Obj-> var1 = $ an_array ['var1'];
$ Obj-> var2 = $ an_array ['var2'];
return $ obj;
}
}

$ A = new A;
$ A-> var1 = 5;
$ A-> var2 = 'foo';
eval ('$ b ='. var_export ($ a, true).';'); / / $ b = A:: _set_state (array (
/ / 'Var1' => 5,
/ / 'Var2' => 'foo',
/ /));
var_dump ($ b);
```

```
?>
```
Результат виконання цього прикладу:
```
object (A) # 2 (2) {
["Var1"] =>
int (5)
["Var2"] =>
string (3) "foo"
}
```

11.5 Функції для роботи з класами та об'єктами

У PHP існує кілька стандартних функцій для роботи з класами та об'єктами. Розглянемо деякі функції для роботи з класами та об'єктами в контексті PHP5.

11.5.1 get_class_methods ()

Функція get_class_methods () повертає масив імен методів класу із заданим ім'ям. Синтаксис функції get_class_methods ():

```
array get_class_methods (string імя_класса)
```

Простий приклад використання get_class_methods () - Отримання списку методів класу:

```
<? Php
...
class Airplane extends Vehicle {
public $ wingspan;
function setWingSpan ($ wingspan) {
$ This-> wingspan = $ wingspan;
}

function getWingSpan () {
return $ this-> wingspan;
}
}

$ Cls_methods = get_class_methods (Airplane);
/ / Масив $ cls_methods містить імена всіх методів,
/ / Оголошених в класах "Airplane" і "Vehicle"
```

```
?>
```

Як видно з лістингу, функція get_class_methods () дозволяє легко отримати інформацію про всі методи, підтримуваних класом.

11.5.2 get_class_vars ()

Функція get_class_vars () повертає масив імен атрибутів класу із заданим ім'ям. Синтаксис функції get_class_vars ():

array get_class_vars (string імя_класса)

Приклад використання функції get_class_vars () - отримання списку атрибутів (властивостей) класу:

```
<? Php
class Vehicle {
public $ model;
public $ current_speed;
}

class Airplane extends Vehicle {
public $ Swingspan;
}

$ A_class = "Airplane";
$ Attribs = get_class_vars ($ a_class);
/ / $ Attribs = array ("wingspan", "model", "current_speed")
?>
```

У розглянутому прикладі масив $ attribs заповнюється іменами всіх атрибутів класу Airplane.

11.5.3 get_object_vars ()

Функція **get_object_vars ()** повертає асоціативний масив з інформацією про всі атрибути об'єкту із заданим ім'ям. Синтаксис функції get_object_vars ():

array get_object_vars (object ім’я_об’єкта)

Приклад використання функції get_object_vars () - отримання інформації про змінні об'єкта:

```
<? Php
class Vehicle {
public $ wheels;
}

class Land extends Vehicle {
public $ engine;
}
class car extends Land {
var $ doors;
function car ($ doors, $ eng, $ wheels) {
$ This-> doors = $ doors;
$ This-> engine = $ eng;
$ This-> wheels = $ wheels;
}
function get_wheels () {
return $ this-> wheels;
}
}
$ Toyota = new car (2,400,4);
$ Vars = get_object_vars ($ toyota);
while (list ($ key, $ value) = each ($ vars)):
print "$ key ==> $ value ";
endwhile;
/ / Вихідні дані:
/ / Wheels ==> 4
/ / Engine ==> 400
/ / Doors ==> 2
?>
```

Функція **get_object_vars ()** дозволяє швидко отримати всю інформацію про атрибути конкретного об'єкта та їх значеннях у вигляді асоціативного масиву.

11.5.4 method_exists ()

Функція **method_exists ()** перевіряє, чи підтримується об'єктом метод із заданим ім'ям. Якщо метод підтримується, функція повертає TRUE, в іншому випадку повертається FALSE. Синтаксис функції method_exists ():

bool method_exi sts (object ім'я_об'єкта. string імя_метода)

Приклад використання методу method_exists () - перевірка підтримки методу об'єктом:

```
<? Php
class Vehicle {
/ / ...
}
class Land extends Vehicle {
public $ fourWheel;
function setFourWheelDrive () {
$ This-> fourWeel = 1;
}
}
/ / Створити об'єкт з ім'ям $ car
$ Car = new Land;
/ / Якщо метод "fourWheelDrive" підтримується класом "Land"
/ / Або "Vehicle", виклик method_exists повертає TRUE;
/ / У противному випадку повертається FALSE.
/ / У даному прикладі method_exists () повертає TRUE.
if (method_exists ($ car, "setfourWheelDrive")):
print "This car is equipped with 4-wheel drive";
else:
print "This car is not equipped with 4-wheel drive";
endif;
?>
```

У розглянутому прикладі функція method_exists () перевіряє, чи підтримується об'єктом $car метод з ім'ям setFourWheelDrive

(). Якщо метод підтримується, функція повертає логічну істину і фрагмент виводить відповідне повідомлення. В іншому випадку повертається FALSE і виводиться інше повідомлення.

11.5.5 get_class ()

Функція get_class () повертає ім'я класу, до якого належить об'єкт із заданим ім'ям. Синтаксис функції get_class ():

string get_class (object ім’я_об’єкта);

Приклад використання get_class () - отримання імені класу:

```
<? Php
class Vehicle {
}
class Land extends Vehicle {
}
/ / Створюємо об'єкт з ім'ям $ car:
$ Car = new Land;
/ / Змінній $ class_a присвоюється рядок "Land":
$ Class_a = get_class ($ car);
echo $ class_a;
?>
```

У розглянутому прикладі змінної $ class_a присвоюється ім'я класу, на основі якого був створений об'єкт $ car.

11.5.6 get_parent_class ()

Функція get_parent_class () повертає ім'я батьківського класу (якщо він є) для об'єкту із заданим ім'ям. Синтаксис функції get_parent_dass ():

string get_parent_class (object ім’я_об’єкта);

Приклад отримання імені батьківського класу функцією get_parent_class ():

```
<? Php
class Vehicle {
//...
}
```

```
class Land extends Vehicle {
//...
}
/ / Створюємо об'єкт з ім'ям $ car:
$ Car = new Land;
/ / Змінній $ parent присвоюється рядок "Vehicle":
$ Parent = get_parent_class ($ car);
?>
```

При виклику get_parent_class () змінної $ parent буде привласнений рядок "Vehicle".

11.5.7 is_subclass_of ()

Функція **is_subclass_of ()** перевіряє, чи був об'єкт створений на базі класу, що має батьківський клас із заданим ім'ям. Функція повертає TRUE, якщо перевірка дає позитивний результат, і FALSE в іншому випадку. Синтаксис функції is_subclass_of ():

```
bool is_subclass_of (object об'єкт, string імя_класса)
```

Приклад використання функції is_subdass_of ():

```
<? Php
class Vehicle {
//...
}
class Land extends Vehicle {
//...
}
$ Auto = new Land;
/ / Змінній $ is_subclass присвоюється TRUE
$ Is_subclass = is_subclass_of ($ auto, "Vehicle");
?>
```

У розглянутому прикладі змінної $ is_subclass () присвоюється ознака того, чи належить об'єкт $ auto до підкласу батьківського класу Vehicle. У наведеному фрагменті $ auto

відноситься до класу Vehicle; отже $ is_subclass () буде присвоєно значення TRUE.

11.5.8 get_declared_classes ()

Функція get_declared_classes () повертає масив з іменами всіх визначених класів. Синтаксис функції get_declared_classes ():

array get_declared_classes ()

Приклад отримання списку класів функцією get_declared_classes ():

```
<? Php
class Vehicle {
//...
}
class Land extends Vehicle {
//...
}
$ Declared_classes = get_declared_classes ();
/ / $ Declared_classes = array ("Vehicle", "Land")
?>
```

Ми розглянули лише деякі основні функції, призначені для роботи з класами та об'єктами PHP. Для ознайомлення з повним переліком таких функцій зверніться до довідника функцій PHP.

11.6 Обробка виняткових ситуацій

11.6.1 Оголошення винятків

Модель винятків **(exceptions)** в PHP 5 простіша, ніж в інших мовах програмування Виняток можна згенерувати (як кажуть, "викинути") за допомогою оператора throw, і можна перехопити (або, як кажуть, "піймати") оператором catch. Код, що викидає виключення, повинен бути оточений блоком try, для того щоб можна було перехопити виняток. Кожен блок try повинен мати як мінімум один відповідний блок catch. Так само можна використовувати кілька блоків catch, що перехоплюють різні класи винятків. Нормальне виконання буде продовжено за останнім блоком catch. Винятки так само можуть бути

згенерованими (або перегерерованої - тобто викинуті знову) оператором throw всередині блоку catch.

При генерації виключення, код наступний нижче оператора throw виконаний не буде, а PHP зробить спробу знайти перший блок catch, що перехоплює виключення даного класу. Якщо виключення не буде перехоплено, PHP видасть повідомлення про помилку: "Uncaught Exception ..." (Неперехвачений виняток), якщо звичайно не був визначений обробником помилок за допомогою функції set_exception_handler ().

Приклад Викид винятків

```
<? Php
function inverse ($ x) {
if (! $ x) {
throw new Exception ('Поділ на нуль.');
}
else return 1 / $ x;
}
try {
echo inverse (5). "\ N";
echo inverse (0). "\ N";
} Catch (Exception $ e) {
echo 'Викинуто виняток:', $ e-> getMessage (), "\ n";
}

/ / Продовження виконання
echo 'Hello World';
?>
```

Результат виконання цього прикладу:

```
0.2
Викинуто виняток: Поділ на нуль.
Hello World
```

11.6.2 Спадкування винятків

Певний користувачем клас виключення повинен бути визначений, як клас, що розширює вбудований клас Exception.

Нижче наведено методи і властивості класу Exception, доступні дочірнім класам.

Приклад Вбудований клас Exception

```
<? Php
class Exception
{
protected $ message = 'Unknown exception'; / / повідомлення
protected $ code = 0; / / Код винятку,
визначається користувачем
protected $ file; / / файл у якому було
викинуто виключення
protected $ line; / / рядок у якому було
викинуто виключення

function _construct ($ message = null, $ code = 0);

final function getMessage (); / / Повертає повідомлення
виключення
final function getCode (); / / Код винятку
final function getFile (); / / Файл, де викинуто
виняток
final function getLine (); / / Рядок, що викинув виключення
final function getTrace (); / / Масив backtrace ()
final function getTraceAsString (); / / Зворотнє трасування, як
рядок

/ * Overrideable - тобто те, що можна перевизначити * /
function _toString (); / / повинен повернути
форматований рядок, для відображення
}
?>
```

Якщо клас, успадкований від Exception перевизначає конструктор, необхідно викликати в конструкторі parent:: _construct (), щоб бути впевненим, що всі дані будуть доступні.

Метод _toString () може бути перевизначений, що б забезпечити потрібний висновок, коли об'єкт перетворюється в рядок.

Приклад Успадкування класу Exception

```
<? Php
/ **
* Визначимо свій клас винятку
* /
class MyException extends Exception
{
/ / Перевизначивши виняток так, що параметр message стане обов'язковим
public function _construct ($ message, $ code = 0) {
/ / Якийсь код
parent:: _construct ($ message, $ code);
}
/ / Перевизначивши рядкове подання об'єкта.
public function _toString () {
return _CLASS_. ": [{$ This-> code}]: {$ this-> message} \ n";
}
public function customFunction () {
echo "Ми можемо визначати нові методи в успадкованому класі \ n";
}
}
/ * Створимо клас для тестування винятку * /
class TestException
{
public $ var;
const THROW_NONE = 0;
const THROW_CUSTOM = 1;
const THROW_DEFAULT = 2;

function _construct ($ avalue = self:: THROW_NONE) {
switch ($ avalue) {
case self:: THROW_CUSTOM:
/ / Кидаємо власне виключення
```

```
throw new MyException ('1 - неправильний параметр ', 5);
break;
case self:: THROW_DEFAULT:
/ / Кидаємо вбудований виняток
throw new Exception ('2 - неприпустимий параметр ', 6);
break;
default:
/ / Ніяких винятків, об'єкт буде створений.
$ This-> var = $ avalue;
break;
}
}
}

/ / Example 1
try {
$ O = new TestException (TestException:: THROW_CUSTOM);
} Catch (MyException $ e) {/ / Will be caught
echo "Злови власне, перевизначене виключення \ n", $ e;
$ E-> customFunction ();
} Catch (Exception $ e) {/ / Буде пропущено.
echo "Злови вбудоване виключення \ n", $ e;
}
/ / Звідси буде продовжено виконання програми
var_dump ($ o);
echo "\ n \ n";

/ / Example 2
try {
$ O = new TestException (TestException::
THROW_DEFAULT);
} Catch (MyException $ e) {/ / Тип винятку не співпаде
echo "Злови перевизначене виключення \ n", $ e;
$ E-> customFunction ();
} Catch (Exception $ e) {/ / Буде перехоплено
echo "перехоплено вбудоване виключення \ n", $ e;
```

```
}
/ / Звідси буде продовжено виконання програми
var_dump ($ o);
echo "\ n \ n";

/ / Example 3
try {
$ O = new TestException (TestException:: THROW_CUSTOM);
} Catch (Exception $ e) {/ / Буде перехоплено.
echo "Злови вбудоване виключення \ n", $ e;
}
/ / Продовження виконання програми
var_dump ($ o);
echo "\ n \ n";

/ / Example 4
try {
$ O = new TestException ();
} Catch (Exception $ e) {/ / Буде пропущено, тому що виняток не викидається
echo "Злови вбудоване виключення \ n", $ e;
}
/ / Продовження виконання програми
var_dump ($ o);
echo "\ n \ n";
?>
```

Розділ 4 Проектування баз даних

Тема 12 Робота з базами даних

12.1 Бази даних: основні поняття

12.1.1 Поняття бази заних

У житті ми часто стикаємося з необхідністю зберігати будь-яку інформацію, а тому часто маємо справу і з базами даних. Наприклад, ми використовуємо записну книжку для зберігання номерів телефонів своїх друзів і планування свого часу. Телефонна книга містить інформацію про людей, що живуть в одному місті. Все це свого роду бази даних. Ну а раз це бази даних, то подивимося, як у них зберігаються дані. Наприклад, телефонна книга являє собою таблицю (табл. 1).

У цій таблиці дані - це власне номери телефонів, адреси та ПІБ., Тобто рядки «Іванов Іван Іванович», «32-43-12» і т.п., а назви стовпців цієї таблиці, тобто рядки «ПІБ», «Номер телефону» і «Адреса» задають сенс цих даних, їх семантику.

Таблиця 12.1**.** Приклад бази даних: телефонна книга

ПІБ	Номер телефону	Адреса
Іванов Іван Іванович	32-43-12	вул. Леніна, 12, 43
Ільїн Федір Іванович	32-32-34	пр. Маркса, 32, 45

Тепер уявіть, що записів у цій таблиці не дві, а дві тисячі, ви займаєтеся створенням цього довідника і десь сталася помилка (наприклад, помилка в адресі). Мабуть, важкувато буде знайти і виправити цю помилку вручну. Потрібно скористатися якимись засобами автоматизації. Для управління великою кількістю даних програмісти (не без допомоги математиків) придумали системи управління базами даних (СУБД). У порівнянні з текстовими базами даних електронні СУБД мають величезне число переваг, від можливості швидкого пошуку інформації, взаємозв'язку даних між собою, до використання цих даних у різних прикладних програмах і одночасного доступу до даних декількох користувачів.

База даних - це сукупність пов'язаних даних, організованих за певними правилами, що передбачають загальні принципи опису, зберігання і маніпулювання, незалежна від прикладних програм. База даних є інформаційною моделлю предметної області. Звернення до баз даних здійснюється за допомогою системи управління базами даних (СУБД). СУБД забезпечує підтримку створення баз даних, централізованого управління та організації доступу до них різних користувачів.

Отже, ми прийшли до висновку, що зберігати дані незалежно від програм, так, що б вони були пов'язані між собою і організовані за певними правилами, доцільно. Але питання, як зберігати дані, за якими правилами вони мають бути організовані, залишилося відкритим. Способів існує безліч (до речі, називаються вони моделями подання або зберігання даних). Найбільш популярні - об'єктна і реляційна моделі даних.

Автором реляційної моделі вважається Е. Кодд, який першим запропонував використовувати для обробки даних апарат теорії множин (об'єднання, перетин, різниця) і показав, що будь-яке представлення даних зводиться до сукупності двовимірних таблиць особливого виду, відомого в математиці як відношення.

Таким чином, реляційна база даних являє собою набір таблиць (точно таких же, як наведена вище), пов'язаних між собою. Рядок у таблиці відповідає сутності реального світу (у наведеному вище прикладі це інформація про людину).

Приклади реляційних СУБД: MySql, PostgreSql.

В основу об'єктної моделі покладена концепція об'єктно-орієнтованого програмування, в якій дані представляються у вигляді набору об'єктів і класів, пов'язаних між собою родинними відносинами, а робота з об'єктами здійснюється за допомогою прихованих (інкапсульованих) у них методів.

Приклади об'єктних СУБД: Cache, GemStone (від Servio Corporation), ONTOS (ONTOS).

Останнім часом виробники СУБД прагнуть з'єднати два ці підходи і проповідують об'єктно-реляційну модель

представлення даних. Приклади таких СУБД - IBM DB2 for Common Servers, Oracle8.

Оскільки ми збираємося працювати з Mysql, то будемо обговорювати аспекти роботи тільки з реляційними базами даних. Нам залишилося розглянути ще два важливих поняття з цієї області: ключі та індексування, після чого ми зможемо приступити до вивчення мови запитів SQL.

12.1.2 Ключі

Для початку давайте подумаємо над таким питанням: яку інформацію треба дати про людину, щоб співрозмовник точно сказав, що це саме та людина, сумнівів бути не може, іншої такої немає? Повідомити прізвище, очевидно, недостатньо, оскільки існують однофамільці. Якщо співрозмовник людина, то ми можемо приблизно пояснити, про кого мова, наприклад згадати вчинок, який вчинив той чоловік, або ще якось. Комп'ютер же такого пояснення не зрозуміє, йому потрібні чіткі правила, як визначити, про кого йде мова. У системах управління базами даних для вирішення такого завдання ввели поняття первинного ключа.

Первинний ключ (primary key, PK) - мінімальний набір полів, унікально ідентифікує запис у таблиці. Значить, первинний ключ - це в першу чергу набір полів таблиці, по-друге, кожен набір значень цих полів повинен визначати єдиний запис (рядок) в таблиці і, по-третє, цей набір полів повинен бути мінімальним з усіх, що володіють такою же властивістю. Оскільки первинний ключ визначає тільки один унікальний запис, то ніякі два записи таблиці не можуть мати однакових значень первинного ключа.

Наприклад, у нашій таблиці ПІБ та адрес дозволяють однозначно виділити запис про людину. Якщо ж говорити загалом, без зв'язку з розв'язуванням завданням, то такі знання не дозволяють точно вказати на єдину людину, оскільки існують однофамільці, що живуть у різних містах за однією адресою. Вся справа в межах, які ми самі собі ставимо. Якщо вважаємо, що знання ПІБ, телефону та адреси без вказівки міста для наших

цілей достатньо, то все чудово, тоді поля ПІБ та адрес можуть утворювати первинний ключ. У будь-якому випадку проблема створення первинного ключа лягає на плечі того, хто проектує базу даних (розробляє структуру зберігання даних). Рішенням цієї проблеми може стати або виділення характеристик, які природним чином визначають запис в таблиці (завдання так званого логічного, або природного, PK), або створення додаткового поля, призначеного саме для однозначної ідентифікації записів в таблиці (завдання так званого сурогатного, або штучного, PK). Прикладом логічного первинного ключа є номер паспорта у базі даних про паспортні дані мешканців чи ПІБ та адресу в телефонній книзі (таблиця вище). Для завдання сурогатного первинного ключа в нашу таблицю можна додати поле id (ідентифікатор), значенням якого буде ціле число, унікальне для кожного рядка таблиці. Використання таких сурогатних ключів має сенс, якщо природний первинний ключ являє собою великий набір полів або його виділення нетривіально.

Крім однозначної ідентифікації запису, первинні ключі використовуються для організації зв'язків з іншими таблицями.

Наприклад, у нас є три таблиці: містять інформацію про історичні особистості (Persons), що містить інформацію про їх винаходи (Artifacts) і яка містить зображення як особистостей, так і артефактів (Images).

Первинним ключем у всіх цих таблицях є поле id (ідентифікатор). У таблиці Artifacts є поле author, в якому записаний ідентифікатор, присвоєний автору винаходу в таблиці Persons. Кожне значення цього поля є зовнішнім ключем для первинного ключа таблиці Persons. Крім того, в таблицях Persons і Artifacts є поле photo, яке посилається на зображення в таблиці Images. Ці поля також є зовнішніми ключами для первинного ключа таблиці Images і встановлюють однозначно логічний зв'язок Persons-Images і Artifacts-Images. Тобто якщо значення зовнішнього ключа photo в таблиці особистості дорівнює 10, то це значить, що фотографія цієї особистості має id = 10 в таблиці зображень. Таким чином, зовнішні ключі

використовуються для організації зв'язків між таблицями бази даних (батьківськими і дочірніми) і для підтримки обмежень посилальної цілісності даних.

12.1.3 Індексування

Одна з основних задач, що виникають при роботі з базами даних, - це завдання пошуку. При цьому, оскільки інформації в базі даних, як правило, міститься багато, перед програмістами постає завдання не просто пошуку, а ефективного пошуку, тобто пошуку за порівняно невеликий час і з достатньою точністю. Для цього (для оптимізації продуктивності запитів) виробляють індексування деяких полів таблиці. Використовувати індекси корисно для швидкого пошуку рядків з вказаним значенням одного стовпця. Без індексу читання таблиці здійснюється по всій таблиці, починаючи з першого запису, поки не будуть знайдені відповідні рядки. Чим більша таблиця, тим більш накладні витрати. Якщо ж таблиця містить індекс по розглянутих стовпцях, то база даних може швидко визначити позицію для пошуку в середині файлу даних без перегляду всіх даних. Це відбувається тому, що база даних поміщає проіндексовані поля ближче в пам'яті, так, щоб можна було швидше знайти їх значення. Для таблиці, яка містить 1000 рядків, це буде як мінімум в 100 разів швидше в порівнянні з послідовним перебором всіх записів. Однак у випадку, коли необхідний доступ майже до всіх 1000 рядках, швидше буде послідовне читання, так як при цьому не потрібно операцій пошуку по диску. Так що іноді індекси бувають тільки перешкодою. Наприклад, якщо копіюється великий обсяг даних в таблицю, то краще не мати ніяких індексів. Проте в деяких випадках потрібно задіяти одразу кілька індексів (наприклад, для обробки запитів до часто використовуваних таблиць).

Якщо говорити про MySQL, то там існує три види індексів: PRIMARY, UNIQUE, і INDEX, а слово ключ (KEY) використовується як синонім слова індекс (INDEX). Всі індекси зберігаються в пам'яті у вигляді B-дерев.

PRIMARY - унікальний індекс (ключ) з обмеженням, що всі індексовані їм поля не можуть мати порожнього значення (тобто вони NOT NULL). Таблиця може мати тільки один первинний індекс, але він може складатися з декількох полів.

UNIQUE - ключ (індекс), що задає поля, які можуть мати тільки унікальні значення.

INDEX - звичайний індекс (як ми описали вище). У MySqL, крім того, можна індексувати рядкові поля за заданою кількістю символів від початку рядка.

12.2 Основна інформація про MySQL

12.2.1 Поняття MySQL

MySQL - це популярна система управління базами даних (СУБД), дуже часто застосовується в поєднанні з PHP.

База даних являє собою структуровану сукупність даних. Ці дані можуть бути будь-якими - від простого списку майбутніх покупок до переліку експонатів картинної галереї або величезної кількості інформації в корпоративній мережі. Для запису, вибірки й обробки даних, що зберігаються в комп'ютерній базі даних, необхідна система управління базою даних, якою і є MySQL. Оскільки комп'ютери чудово справляються з обробкою великих обсягів даних, управління базами даних відіграє центральну роль в обчисленнях. Реалізовано таке управління може бути по-різному - як у вигляді окремих утиліт, так і у вигляді коду, що входить до складу інших додатків. MySQL - це система управління реляційними базами даних. У реляційній базі даних дані зберігаються не все скопом, а в окремих таблицях, завдяки чому досягається виграш у швидкості й гнучкості. Таблиці зв'язуються між собою за допомогою відносин, завдяки чому забезпечується можливість поєднувати при виконанні запиту дані з декількох таблиць. SQL як частина системи MySQL можна охарактеризувати як мова структурованих запитів плюс найбільш поширена стандартна мова, яка використовується для доступу до баз даних. MySQL - це ПЗ з відкритим кодом. Застосовувати його і модифікувати може будь-хто. Таке ПЗ можна одержувати по Internet і

використовувати безкоштовно. При цьому кожен користувач може вивчити вихідний код і змінити його у відповідності зі своїми потребами. Використання програмного забезпечення MySQL регламентується ліцензією GPL (GNU General Public License), http://www.gnu.org/licenses/, в якій зазначено, що можна і чого не можна робити з цим програмним забезпеченням у різних ситуаціях. Чому веб-програмісти віддають перевагу СУБД MySQL? MySQL є дуже швидким, надійним і легким у використанні. Якщо вам потрібні саме ці якості, спробуйте попрацювати з даним сервером. MySQL володіє також рядом зручних можливостей, розроблених у тісному контакті з користувачами. Спочатку сервер MySQL розроблявся для керування великими базами даних з метою забезпечити більш високу швидкість роботи в порівнянні з існуючими на той момент аналогами. І ось вже протягом декількох років даний сервер успішно використовується в умовах промислової експлуатації з високими вимогами. Незважаючи на те що MySQL постійно вдосконалюється, він уже сьогодні забезпечує широкий спектр корисних функцій. Завдяки своїй доступності, швидкості та безпеки MySQL дуже добре підходить для доступу до баз даних по Internet.

Технічні можливості СУБД MySQL MySQL є системою клієнт-сервер, яка містить багато-поточний SQL-сервер, що забезпечує підтримку різних обчислювальних машин баз даних, а також кілька різних клієнтських програм і бібліотек, засоби адміністрування і широкий спектр програмних інтерфейсів (API). Ми також постачаємо сервер MySQL у вигляді багатопоточної бібліотеки, яку можна підключити до призначеної для користувача програми і отримати компактний, більш швидкий і легкий в управлінні продукт. Доступно також велику кількість програмного забезпечення для MySQL, в більшій частині - безкоштовного.

12.2.2 Пристрій MySQL

MySQL складається з двох частин: серверної і клієнтської.

Сервер MySQL постійно працює на комп'ютері. Клієнтські програми (наприклад, скрипти PHP) посилають серверу MySQL SQL-запити через механізм сокетів (тобто за допомогою мережевих засобів), сервер їх обробляє і запам'ятовує результат. Тобто скрипт (клієнт) вказує, яку інформацію він хоче отримати від сервера баз даних. Потім сервер баз даних посилає відповідь (результат) клієнту (скрипту).

Чому завжди передається не весь результат? Дуже просто: справа в тому, що розмір результуючого набору даних може бути занадто великим, і на його передачу по мережі піде надто багато часу. Та й рідко коли буває потрібно отримувати відразу весь висновок запиту (тобто всі записи, що задовольняють висловом запиту). Наприклад, нам може знадобитися лише підрахувати, скільки записів задовольняється тій чи іншій умові, або ж вибрати з даних тільки перші 10 записів. Механізм використання сокетів базується на технології клієнт-сервер, а це означає, що в системі повинна бути запущена спеціальна програма - MySQL-сервер, яка приймає і обробляє запити від програм. Тому що вся робота відбувається в дійсності на одній машині, накладні витрати по роботі з мережевими засобами незначні (установка і підтримка з'єднання з MySQL-сервером обходиться досить дешево).

Структура MySQL трирівнева: бази даних - таблиці - записи. Бази даних і таблиці MySQL фізично представляються файлами з розширеннями frm, MYD, MYI. Логічно - таблиця являє собою сукупність записів. А записи - це сукупність полів різного типу. Ім'я бази даних MySQL унікальне в межах системи, а таблиці - в межах бази даних, поля - в межах таблиці. Один сервер MySQL може підтримувати відразу декілька баз даних, доступ до яких може розмежовуватись логіном і паролем. Знаючи логін і пароль, можна працювати з конкретною базою даних. Наприклад, можна створити або видалити в ній таблицю, додати записи і т. д. Зазвичай ім'я-ідентифікатор та пароль призначаються хостинг провайдерами, які і забезпечують підтримку MySQL для своїх користувачів.

12.2.3 Поля та його типи в MySQL

База даних з точки зору MySQL (і деяких інших СУБД) - це звичайний каталог, що містить виконавчі файли певного формату - таблиці. Таблиці складаються із записів, а записи, у свою чергу, складаються з полів. Поле має два атрибути - ім'я і тип.

Тип поля може бути:

- Цілим;
- Речовим;
- Строковим;
- Бінарним;
- Дата і час;
- Перерахування та множини.

Можливі типи даних, діапазони та описи представлені в наступних таблицях:

Тип	Діапазон
TINYINT	-128 ... +127
SMALLINT	-32768 ... +32767
MEDIUMINT	-8388608 ... +8 388 607
INT	-2147483648 ... +2 147 483 647
BIGINT	-9 223 372 036 854 775 808 ... +9 223 372 036 854 775 807

Дійсні типи записуються у вигляді:

ТИП (ДОВЖИНА, ЗНАКИ) [UNSIGNED]

Довжина - це кількість знакомість, в яких буде розміщено все число при його передачі, а ЗНАКИ - це кількість знаків після десяткової точки, які будуть враховуватися. Якщо вказаний модифікатор UNSIGNED, знак числа враховуватися не буде.

Дійсні числа

Тип	Опис
FLOAT	Невелика точність
DOUBLE	Подвійна точність

REAL	Те ж, що і DOUBLE
DECIMAL	Дробове число, яке зберігається у вигляді рядка
NUMERIC	Те ж, що і DECIMAL

Будь-який рядок - це масив символів. При пошуку за допомогою оператора SELECT (ми розглянемо його далі) не враховується регістр символів: рядки "HELLO" і "Hello" вважаються однаковими.

Можна налаштувати MySQL на автоматичне перекодування символів - в цьому випадку в базі даних рядки будуть зберігатися в одному кодуванні, а виводитися - в іншій.

У більшості випадків застосовується тип VARCHAR або просто CHAR, що дозволяє зберігати рядки, що містять до 255 символів. У дужках після типу вказується довжина рядки:

VARCHAR (48);

CHAR (73);

Якщо 255 символів для вашої задачі недостатньо, можна використовувати інші види, наприклад, TEXT.

Тип	Опис
TINYTEXT	Максимальна довжина 255 символів
TEXT	Максимальна довжина 65535 символів (64 Кб)
MEDIUMTEXT	Максимальна довжина 16777215 символів
LONGTEXT	Максимальна довжина 4294967295 символів

Бінарні типи даних також можна використовувати для зберігання тексту, але при пошуку буде враховуватися регістр символів. До того ж, будь-який текстовий тип можна перетворити на бінарний, вказавши модифікатор BINARY:

VARCHAR (30) BINARY;

Бінарні типи даних

Тип	Опис
TINYBLOB	Максимум 255 символів

BLOB	Максимум 65535 символів
MEDIUMBLOB	Максимум 16777215 символів
LONGBLOB	Максимум 4294967295

Примітка: Бінарні дані не перекодовуються "на льоту", якщо встановлено перекодування символів.

Дата і час

Тип	Опис
DATE	Дата в форматі РРРР-ММ-ДД
TIME	Час у форматі ГГ: ХвХв: СС
TIMESTAMP	Дата і час у форматі timestamp, виводиться у вигляді РРРРММДД ГГХвХвСС
DATETIME	Дата і час у форматі РРРР-ММ-ДД ГГ: ХвХв: СС

Інші типи даних MySQL розглядати безглуздо, оскільки застосування їх в PHP недоцільно

12.3 Оператори і команди MySQL

Структурована мову запитів SQL дозволяє робити різні операції з базами даних: створювати таблиці, поміщати, оновлювати і видаляти з них дані, виконувати запити з таблиць і т.д. Далі ми послідовно розглянемо всі ці оператори.

Незважаючи на те, що останній стандарт SQL прийнятий в 1992 році, на сьогоднішній день немає ні однієї СУБД, де б він повністю виконувався. Більш того, в різних базах даних частина операцій здійснюється по-різному. Ми будемо дотримуватися діалекту SQL характерного для СУБД MySQL тому не всі запити можуть виконуватися для інших баз даних.

Примітка: Команди SQL не чутливі до регістру, але традиційно вони набираються прописними літерами.

12.3.1 Створення таблиць. Оператор CREATE

Створити таблицю через SQL-запит дозволяє оператор CREATE. Його синтаксис:

```
CREATE TABLE ім'я_таблиці
(
Ім’я_поля1 Тип Модифікатор
...
Ім’я_поляN Тип Модифікатор
[Первинний ключ]
[Зовнішній ключ]
)
```

Взагалі, за допомогою оператора CREATE можна створювати й інші об'єкти, але ми їх розглядати не будемо, оскільки їх застосування досить обмежена. В якості модифікаторів можна використовувати такі значення: • NOT NULL - поле не може містити невизначеного значення (NULL), тобто поле повинно бути явно ініціалізований;

• PRIMARY KEY - поле буде первинним ключем (ідентифікатором запису), за яким можна однозначно ідентифікувати запис;

• AUTO_INCREMENT - при вставці нового запису значення цього поля буде автоматично збільшено на одиницю, тому в таблиці не буде двох записів з однаковим значенням цього поля;

• DEFAULT - задає значення, яке буде використано за замовчуванням, якщо при вставці запису поле не буде ініціалізувати явно. Значення за замовчуванням задається так:

```
Ім'я_поля Тип DEFAULT Значення, наприклад
NO INT DEFAULT 0
NAME INT DEFAULT 'Петров'
```

Тепер створимо таблиці - "Товар", "Клієнти", "Замовлення":

```
CREATE TABLE CLIENTS
(
C_NO int NOT NULL,
FIO char (40) NOT NULL,
ADDR char (30) NOT NULL,
```

```
CITY char (15) NOT NULL,
PHONE char (11) NOT NULL
);
```

Таблиця CLIENTS містить поля C_NO (номер клієнта), FIO (Прізвище, ім'я, по батькові), ADDR (Адреса), CITY (Місто) і PHONE (Телефон). Всі ці поля не можуть містити порожнього значення (NOT NULL).

```
CREATE TABLE TOOLS
(
T_NO int NOT NULL,
DSEC char (40) NOT NULL,
PRICE double (9,2) NOT NULL,
QTY double (9,2) NOT NULL
);
```

Дана таблиця буде містити дані про товари. Тип double (9,2) означає, що 9 знаків відносимо під цілу частину, і два - під дробову. QTY - це кількість товару на складі.

```
CREATE TABLE ORDERS
(
O_NO int NOT NULL,
DATE date NOT NULL,
C_NO int NOT NULL,
T_NO int NOT NULL,
QUANTITY double (9,2) NOT NULL,
AMOUNT double (9,2) NOT NULL
);
```

Ця таблиця містить відомості про замовлення - номер замовлення (O_NO), дату замовлення (DATE), номер клієнта (C_NO), номер товару (T_NO), кількість (QUANTITY) та суму всього замовлення AMOUNT (тобто AMOUNT = T_NO * TOOL_PRICE).

12.3.2 Додавання даних в таблицю. Оператор INSERT

Для додавання записів використовується оператор INSERT:

```
INSERT INTO ім'я_таблиці [(Список полів)]
VALUES (Список констант);
```

Після виконання оператора INSERT буде створений новий запис, в якості значень полів будуть використані відповідні константи, зазначені в списку VALUES.

Тепер додамо дані в наші таблиці. Додати дані можна за допомогою оператора INSERT. Розглянемо приклад використання оператора INSERT:

```
INSERT INTO CLIENTS
VALUES (1, 'Іванов І.І.', 'Вокзальна 3', 'Київ', '09599911100 ');
```

Значення, що додаються повинні відповідати тому порядку, в якому поля перераховані в операторі CREATE. Якщо ви хочете додавати інформацію в іншому порядку, то ви повинні вказати цей порядок в операторі INSERT, наприклад:

```
INSERT INTO CLIENTS (FIO, ADDRESS, C_NO, PHONE, CITY)
VALUES ('Петров', 'Світ', '29 ', 2,' - ',' Рівне ');
```

За допомогою INSERT ми можемо додавати дані і в певні поля, наприклад, C_NO і FIO:

```
INSERT INTO CLIENTS (C_NO, FIO)
VALUES (1, 'Іванов');
```

Проте, в нашому випадку сервер MySQL не виконає такий запит, оскільки всі інші поля рівні NULL (порожнє значення), а наша таблиця не приймає порожні значення. Аналогічно можна додати дані в інші таблиці.

Як приклад, додамо дані в таблицю TOOLS:

```
INSERT INTO TOOLS
VALUES (1, 'Клавіатура ABC', 340.98);
```

Зверніть увагу, що ми поки не вказали первинні ключі таблиці, тому нам ніхто не заважає додати до таблиці однакові записи. Додати дату в поле DATE можна за допомогою функції TO_DATE:

```
INSERT INTO ORDERS
VALUES (1, TO_DATE ("01 / 03/05, 'DD / MM / YY'), 1,1,1,340.98);
```

Даний запис означає, що першого березня 2005 Іванов І.І. (C_NO = 1) замовив одну (QUANTITY = 1) клавіатуру ABC (T_NO = 1).

12.3.3Оновлення записів. Оператор UPDATE

Синтаксис оператора UPDATE, який використовується для оновлення записів, виглядає так:

```
UPDATE ім'я_таблиці
SET Поле1 = Значення1, ... , ПолеN = ЗначенняN
[WHERE Умова];
```

Якщо не задана умова WHERE, то буде модифікована вся таблиця, а це може спричинити за собою непередбачувані наслідки, оскільки для всіх записів будуть встановлені однакові значення полів, тому завжди вказуйте умову WHERE.

Припустимо, нам необхідно оновити запис, якщо, наприклад, клієнт Іванов переїхав в інше місто і нам потрібно відзначити цю подію в базі даних. Зробимо наступне:

```
UPDATE CLIENTS
SET CITY = 'Псков'
WHERE C_NO = 1;
```

Даний запит потрібно розуміти так: знайти запис, поле C_NO якої = 1 (це код клієнта Іванова), та встановити значення CITY рівним "Київ".

12.3.4 Видалення записів. Оператор DELETE

Якщо нам необхідно видалити всіх клієнтів, номери яких перевищують 5, то ми вчинимо наступним чином:

```
DELETE FORM CLIENTS
WHERE C_NO> 5;
```

За допомогою оператора DELETE можна видалити всі записи таблиці, вказавши умова, яке підійде для всіх записів, наприклад:

```
DELETE FROM CLIENTS;
```

Якщо друга частина оператора DELETE-WHERE не вказана, отже, дія оператора поширюється на всі записи відразу.

12.3.5 Вибір записів. Оператор SELECT

Додавання, зміна та видалення записів - це, звичайно, дуже важливі команди, але ви часто будете використовувати оператор

SELECT, який вибирає дані з таблиці. Синтаксис цього оператора більш складний:

SELECT [DISTINCT | ALL] {* | [Поле1 AS псевдонім] [,..., полеN AS псевдонім]}

FROM Ім'я_таблиці1 [,..., Ім'я_таблиціN]

[WHERE умова]

[GROUP BY список полів] [HAVING умова]

[ORDER BY список полів]

Ми повністю не будемо розглядати оператор SELECT, краще це робити на конкретному прикладі. Зараз ми розглянемо оператор SELECT у загальних рисах. Наприклад, для виведення всіх записів з таблиці CLIENTS зробіть наступне:

SELECT * FROM CLIENTS;

У результаті ви отримаєте таку відповідь сервера:

C_NO FIO ADDR CITY PHONE

1 Іванов І.І. Вокзальна 3 Київ 09599911100

1 Іванов І.І. Вокзальна 3 Київ 09599911100

2 Петров П.П. Світ 29 Рівне 3438920437

Зверніть увагу на перші два записи - вони однакові. Теоретично, додавання однакових записів можливо - адже ми не вказали первинний ключ таблиці. Якщо ви хочете виключити однакові записи з відповіді сервера (але не з таблиці), використовуйте запит:

SELECT DISTINCT *

FROM CLIENTS;

Припустимо, ви хочете вивести тільки прізвище і номер телефону клієнта, тоді використовуйте наступний запит:

SELECT DISTINCT FIO, PHONE

FROM CLIENTS;

Якщо вам потрібно вивести всі товари, ціна на які перевищує 800, то скористайтеся таким запитом:

SELECT *

FROM TOOLS

WHERE PRICE> 800;

Ви можете використовувати наступні оператори відносин: <,>, =, <>, <=,> =.

Якщо у вашій таблиці присутні кілька однофамільців, то для виведення інформації про всі з них, використовуйте модифікатор LIKE, наприклад:

```
SELECT *
FROM CLIENTS
WHERE FIO LIKE '% Іванов%';
```

Наведений запит можна задати так: вивести інформацію про клієнтів, прізвище яких схожі на 'Іванов'.

Якщо вам необхідно вибрати дані з різних таблиць, то перед іменем поля потрібно вказувати ім'я таблиці. Ось запит, який дозволяє вивести імена всіх клієнтів, які хоча б один раз купували товар:

```
SELECT DISTINCT CLIENTS.FIO
FROM CLIENTS, ORDERS
WHERE CLIENTS.C_NO = ORDERS.C_NO;
```

Оператор SELECT дозволяє Використовувати вкладені запити, однак MySQL їх не підтримує.

Внутрішні функції MIN, MAX, AVG, SUM

При роботі з оператором SELECT вам доступні кілька дуже корисних внутрішніх функцій MySQL, який обчислює кількість елементів (COUNT), суму елементів (SUM), максимальне і мінімальне значення (MAX і MIN), а також середнє значення (AVG).

Наступні оператори виведуть, відповідно, кількість записів у таблиці CLIENTS, найдорожчий товар та суму цін всіх товарів:

```
SELECT COUNT (*)
FROM CLIENTS;
SELECT MAX (PRICE)
FORM TOOLS;
SELECT SUM (PRICE)
FROM TOOLS;
```

12.3.6 Угрупування записів

Оператор SELECT дозволяє групувати повернені значення. Наприклад, клієнт Іванов (C_NO = 1) кілька разів замовляв якийсь товар. Значить, його номер зустрічається в таблиці

ORDERS кілька разів. Інший клієнт також міг зробити кілька замовлень. Ми можемо згрупувати всі записи по полю C_NO (номер клієнта), а потім вивести суму замовлення кожного клієнта.

```
SELECT CLIENTS.FIO, SUM (ORDERS.AMOUNT) AS TOTALSUM
FROM CLIENTS, ORDERS
WHERE CLIENTS.C_NO = ORDERS.C_NO
GROUP BY ORDERS.C_NO;
```

Угруповання виконує оператор GROUP BY, який є частиною оператора SELECT. Оператор GROUP BY можна обмежити за допомогою HAVING. Цей оператор використовується для відбору рядків, що повертаються GROUP BY. HAVING можна вважати аналогом WHERE, але тільки для GROUP BY: HAVING <умова>

Наприклад, нас цікавлять тільки клієнти, які замовили товарів на загальну суму, що перевищує 1500:

```
SELECT CLIENTS.FIO, SUM (ORDERS.AMOUNT) AS TOTALSUM
FROM CLIENTS, ORDERS
WHERE CLIENTS.C_NO = ORDERS.C_NO
GROUP BY ORDERS.C_NO
HAVING TOTALSUM> 1500;
```

У цьому запиті ми використовували псевдонім стовпця TOTALSUM. У деяких серверах SQL для визначення псевдоніму не потрібно писати службове слово AS, а деякі вимагають застосування знака рівності:

SUM (ORDERS.AMOUNT) TOTALSUM або TOTALSUM = SUM (ORDERS.AMOUNT)

Сортування записів

Поки ми не встановили первинний ключ, сортування таблиці не виконується. Дані будуть відображені в порядку їх занесення в таблицю. Для сортування по полю C_NO результат виведення таблиці CLIENTS використовують наступний оператор (сама таблиця при цьому не сортується):

```
SELECT *
```

```
FROM CLIENTS
ORDER BY C_NO;
```

12.3.7 Ключі

Припустимо, що хтось додав у таблицю CLIENTS запис:

1 Сидоров Свободи 7 Калінінград 0113452103

У той же час, до цього номер 1 був закріплений за Івановим. У нас вийшло, що один і той же номер відповідає різним клієнтам. Щоб уникнути такої плутанини, необхідно Використовувати первинні ключі:

```
ALTER TABLE CUSTOMER
ADD PRIMARY KEY (C_NO);
```

Після цього запиту поле C_NO може містити лише унікальні значення. В якості первинного ключа не можна використовувати поле, допустиме значення NULL. Створити первинний ключ можна й простіше - при створенні таблиці наступним чином:

```
CREATE TABLE CLIENTS
(
C_NO int NOT NULL,
FIO char (50) NOT NULL,
ADDR char (55) NOT NULL,
CITY char (20) NOT NULL,
PRIMARY KEY (C_NO);
);
```

Таблиця ORDERS містить відомості про замовлення. По полю C_MO цієї таблиці ідентифікується замовник. Припустимо, що в таблицю ORDERS хтось ввів значення, якого немає в таблиці CLIENTS. Хто замовив товар? Нам потрібно не допустити подібної ситуації, тому слід використовувати подібний запит:

```
ALTER TABLE ORDERS
ADD FOREIGN KEY (C_NO) REFERENCES CLIENTS;
```

Введені в таблицю ORDERS номери клієнтів C_NO повинні існувати в таблиці CLIENTS. Аналогічно потрібно додати зовнішній ключ по полю T_NO. Ця можливість називається декларативної цілісністю.

Команда ALTER використовується не тільки для надання ключів. Вона призначена для реорганізації таблиці в цілому. Ви хочете додати ще одне поле? Або встановити список допустимих значень для кожного з полів. Все це можна зробити за допомогою команди ALTER:

```
ALTER TABLE CLIENTS
ADD ZIP char (7) NULL;
```

Цей оператор додає в таблицю CLIENTS нове поле ZIP типу char. Зверніть увагу, що ви не можете додати нове поле зі значенням NOT NULL в таблицю, в якій вже є дані. Наприклад, якщо компанія працює тільки з клієнтами Києва і Рівного, то доцільно ввести список допустимих значень для таблиці CLIENTS:

```
ALTER TABLE CLIENTS
ADD CONSTRAINT INVALID_STATE SHECK (CITY IN ('Київ', 'Рівне'));
```

12.3.8 Використання зовнішніх ключів

Тепер заглибимося у вивчення SQL. Ви вже знаєте, як додавати первинний ключ, тепер додамо зовнішній ключ при створенні таблиці. Зовнішні ключі використовуються для зв'язку однієї таблиці з іншою. Наприклад, у таблиці CLIENTS у нас є два клієнти - Іванов (C_NO = 1) і Петров (C_NO = 2). Оператор в магазині при оформленні замовлення помилився і вказав неіснуючий номер, наприклад, C_NO = 3. Як ми потім зможемо ідентифікувати клієнта? Для вирішення такої проблеми і існують зовнішні ключі:

```
CREATE TABLE T
(
/ * Опис полів таблиці * /
FOREING KEY KEY_NAME (LIST)
REFERENCES ANOTHER_TABLE [(LIST2)]
[ON DELETE OPTION]
[ON UPDATE OPTION]
);
```

Тут:

• **KEY_NAME** - Ім'я ключа. Ім'я не є обов'язковим, але рекомендується завжди вказувати ім'я ключа - якщо ви не вкажете ім'я ключа, ви потім не зможете його видалити;

• **LIST** - це список полів, що входять в зовнішній ключ.

• **ANOTHER_TABLE** - це інша таблиця, за якою встановлюється не зовнішній ключ, а необов'язковий елемент;

• **LIST2** - це список полів цієї таблиці. Типи полів у списку LIST повинні збігатися з типами полів у списку LIST2.

Припустимо, що в першій таблиці у нас є поля - NO і NAME - цілого і символьного типу відповідно. У другій таблиці у нас є поля з однаковими іменами та типами. Визначення зовнішнього ключа:

FOREIGN KEY KEY_NAME (NO, NAME)
REFERENCES ANOTHER_TABLE (NAME, NO)

Це визначення некоректно, тому що типи полів NO і NAME не збігаються. Потрібно використовувати таке визначення:

FOREIGN KEY KEY_NAME (NO, NAME)
REFERENCES ANOTHER_TABLE (NO, NAME)
[ON DELETE <OPTION>]
[ON UPDATE <OPTION>]

Якщо поля мають одинакові імена, як у нашому випадку, список LIST2 краще взагалі не вказувати. Необов'язкові параметри ON DELETE <OPTION> і ON UPDATE <OPTION> визначають дію з оновлення інформації в базі даних, при видаленні інформації з таблиці і при її оновленні. А дії можуть бути наступними:

• **CASCADE** - видалення або оновлення значення скрізь, де воно зустрічається. Наприклад, у нас є таблиця клієнтів і замовлень. Ми хочемо видалити запис клієнта з номером C_NO = 1. З таблиці замовлень будуть видалені відомості про всі замовлення, зроблених клієнтом;

• **NOACTION** - ви не зможете видалити інформацію з таблиці клієнтів до тих пір, поки ви не видалите всі замовлення, зроблені цим клієнтом. Тобто дія NOACTION забороняє видаляти запис з основної таблиці, якщо вона використовується в дочірній таблиці;

• **SETNULL** - всі значення в дочірній таблиці будуть замінені на NULL (якщо значення NULL допускаються);

• За допомогою параметра SET_DEFAULT ви можете вказати значення за замовчуванням. Наприклад, якщо ви встановлюєте SET_DEFAULT 1, то при видаленні клієнта з будь-яким номером його замовлення будуть приписуватися клієнту з номером 1, який є в таблиці CLIENTS.

12.3.9 Видалення полів і таблиць. Оператор DROP

Стандартом SQL не передбачено видалення стовпців, проте в MySQL ми це можемо зробити:

ALTER TABLE CLIENTS

DROP ZIP;

А видалити таблицю ще простіше:

DROP ORDERS;

Відключення від СУБД

Використовуючи запит DISCONNECT можна відключитися від використовуваної бази даних, а потім, використовуючи запит CONNECT, підключитися до іншої базі даних. У деяких серверах SQL запит DISCONNECT не працює, а замість CONNECT застосовується запит USE.

З використанням PHP не потрібно використовувати дані запити, бо для від'єднання від сервера MySQL використовується функція mysql_close (), а для підключення до сервера MySQL використовується функція mysql_connect ().

12.4 Функції PHP для роботи з MySQL

Розглянемо основні функції PHP, застосовувані для роботи з MySQL сервером.

12.4.1 Функції з'єднання з сервером MySQL

Основною функцією для з'єднання з сервером MySQL є mysql_connect (), яка підключає скрипт до сервера баз даних MySQL та виконує авторизацію користувача базою даних. Синтаксис у даної функції такий:

mysql_connect ([string $ hostname] [, string $ user] [, sting $ password]);

Як ви напевно помітили, всі параметри даної функції є необов'язковими, оскільки значення за замовчуванням можна прописати у конфігураційному файлі php.ini. Якщо ви хочете вказати інше ім'я MySQL-хоста, користувача і пароль, ви завжди можете це зробити. Параметр $ hostname може бути вказаний у вигляді: хост: порт.

Функція повертає ідентифікатор (типу int) з'єднання, вся наступна робота здійснюється тільки через цей ідентифікатор. При наступному виклику функції mysql_connect () з тими ж параметрами нове з'єднання не буде відкрито, а функція поверне ідентифікатор існуючого з'єднання.

Для закриття з'єднання призначена функція mysql_close (int $ connection_id).

Взагалі, з'єднання можна і не закривати - воно буде закрито автоматично при завершенні роботи PHP скрипта. Якщо ви використовуєте більше одного з'єднання, при виклику mysql_close () потрібно вказати ідентифікатор з'єднання, яке ви хочете закрити. Взагалі не закривати з'єднання - поганий стиль, краще закривати з'єднання з MySQL самостійно, а не сподіваючись на автоматизм PHP, хоча це ваше право.

Якщо ви будете використовувати тільки одне з'єднання з базою даних MySQL за весь час роботи сценарію, можна не зберігати його ідентифікатор і не вказувати ідентифікатор при виклику інших функцій.

Функція mysql_connect () встановлює звичайне з'єднання з MySQL. Однак, PHP підтримує постійні з'єднання - для цього використовуйте функцію mysql_pconnect (). Аргументи цієї функції такі ж, як і у mysql_connect ().

У чому різниця між постійним з'єднанням і звичайним з'єднанням з MySQL? Постійне з'єднання не закривається після завершення роботи скрипта, навіть якщо скрипт викликав функцію mysql_close (). З'єднання прив'язується до PID нащадка веб-сервера Apache (від імені якого він і працює) і закривається лише тоді, коли видаляється процес-власник (наприклад, при завершенні роботи).

PHP працює з постійними сполуками приблизно так: при виконанні функції mysql_pconnect () PHP перевіряє, чи було раніше встановлено з'єднання. Якщо так, то повертається його ідентифікатор, а якщо ні, то відкривається нове з'єднання і повертається ідентифікатор.

Постійні з'єднання дозволяють значно знизити навантаження на сервер, а також підвищити швидкість роботи PHP скриптів, що використовують бази даних.

При роботі з постійними сполуками потрібно стежити, щоб максимальна кількість клієнтів Apache не перевищувала максимального числа клієнтів MySQL, то параметр MaxClient (в конфігураційному файлі Apache - httpd.conf) повинен бути менше або дорівнювати параметру max_user_connection (параметр MySQL).

12.4.2 Функція вибору бази даних

Функція mysql_select_db (string $ db [, int $ id]) вибирає базу даних, з якою буде працювати PHP скрипт. Якщо відкрито не більше одного з'єднання, можна не вказувати параметр $ id.

```
/ / Спроба встановити з'єднання з MySQL:
if (! mysql_connect ($ server, $ user, $ password)) {
echo "Помилка підключення до сервера MySQL";
exit;
}
/ / Підключились, тепер вибираємо базу даних:
mysql_select_db ($ db);
```

12.4.3 Функції помилок

Якщо станеться помилка з'єднання з MySQL, то ви отримаєте відповідне повідомлення і скрипт завершить свою роботу. Це не завжди буває зручно, перш за все, при налагодженні скриптів. Тому, в PHP є наступні дві функції:

- mysql_errno (int $ id);
- mysql_error (int $ id);

Перша функція повертає код помилки, а друга - повідомлення про помилку. У результаті ми можемо використовувати наступне:

echo "ERROR". mysql_errno (). "". mysql_error (). "\ n";

Тепер ви будете знати, через що сталася помилка - ви побачите відповідним чином оформлене повідомлення.

12.4.4 Функції виконання запитів до сервера баз даних

Всі запити до поточної бази даних відправляються функцією mysql_query (). Цій функції потрібно передати один параметр - текст запиту. Текст запиту може містити символи з пробілами і символи нового рядка (\ n). Текст повинен бути складений за правилами синтаксису SQL.

Приклад запиту:

$ Q = mysql_query ("SELECT * FROM mytable");

Наведений запит повинен повернути вміст таблиці mytable. Результат запиту присвоюється змінній $ q. Результат - це набір даних, який після виконання запиту потрібно обробити певним чином.

12.4.5 Функції обробки результатів запиту

Якщо запит, виконаний за допомогою функції mysql_query () успішно виконався, то в результаті клієнт отримає набір записів, який може бути оброблений наступними функціями PHP:

• mysql_result () - отримати необхідний елемент з набору записів;

• mysql_fetch_array () - занести запис в масив;

• mysql_fetch_row () - занести запис в масив;

• mysql_fetch_assoc () - занести запис в асоціативний масив;

• mysql_fetch_object () - занести запис в об'єкт.

Також можна визначити кількість записів і полів в результаті запиту. Функція mysql_num_rows () дозволяє дізнатися, скільки записів містить результат запиту:

$ Q = mysql_query ("SELECT * FROM mytable");

echo "Таблиця mytable". mysql_num_rows ($ q). "записів";

Запис складається з полів (колонок). За допомогою функції mysql_num_fields () можна дізнатися, скільки полів містить кожний запис результату:

```
$ Q = mysql_query ("SELECT * FROM mytable");
echo "Таблиця mytable". mysql_num_fields ($ q). "полів";
```

У нас також є можливість дізнатися значення кожного поля. Це можна зробити за допомогою наступної функції:

```
mysql_result (int $ result, int $ row, mixed $ field);
```

Параметр функції $ row задає номер запису, а параметр $ field - ім'я чи порядковий номер поля.

Припустимо, SQL-запит повернув наступний набір даних:

```
Email Name Last_Name
----------------------------------
ivanov@ukr.net Ivan Ivanov
petrov@ukr.net Petr Petrov
```

Вивести це в браузер можна наступним чином:

```
$ Rows = mysql_num_rows ($ q);
$ Fields = mysql_num_fields ($ q);
echo "<pre>";
for ($ c = 0; $ c <$ rows; $ c + +) {
for ($ cc = 0; $ cc <$ fields; $ cc + +) {
echo mysql_result ($ q, $ c, $ cc). "\ t";
echo "\ n";
}
}
echo "</ pre>";
```

Слід зазначити, що функція mysql_result () універсальна: знаючи кількість записів і кількість полів, можна "обійти" весь результат, але в теж час, швидкість роботи даної функції досить низька. Тому, для обробки великих наборів записів рекомендується використовувати функції mysql_fecth_row (), mysql_fecth_array (), і.т.д.

Функція mysql_fecth_row (int $ res) отримує відразу весь рядок, відповідного поточного запису результату $ res. Кожен наступний виклик функції переміщає покажчик запиту на наступну позицію (як при роботі з файлами) і отримує

наступний запис. Якщо більше немає записів, то функція повертає FALSE.

Приклад використання даної функції:

```
$ Q = mysql_query ("SELECT * FROM mytable WHERE month = \" $ db_m \ "AND day = \" $ db_d \ ");
for ($ c = 0; $ c
{
$ F = mysql_fetch_row ($ q);
echo $ f;
}
```

Використовувати функцію mysql_fecth_row () не завжди зручно, тому що значення всіх полів одного запису знаходяться всі в одному рядку. Зручніше використовувати функцію mysql_fetch_array (), яка повертає асоціативний масив, ключами якого будуть імена полів.

Функція mysql_fecth_array (int $ res [, int $ result_type]) повертає не асоціативний масив, а масив, заданий необов'язковим параметром $ result_type, який може приймати такі значення:

• MYSQL_ASSOC - повертає асоціативний масив;

• MYSQL_NUM-повертає масив з числовими індексами, як у функції mysql_fecth_row ();

• MYSQL_BOTH - повертає масив з подвійними індексами, тобто ви можете працювати з ним, як з асоціативним масивом і як зі списком (MYSQL_BOTH - це значення за умовчанням параметр $ result_type.

У PHP є функція, що повертає асоціативний масив з одним індексом:

```
mysql_fetch_assoc (int $ res);
```

Фактично, дана функція є синонімом для mysql_fetch_array ($ res, MYSQL_ASSOC);

Приклад використання функції mysql_fecth_array ():

```
$ Q = mysql_query ("SELECT * FROM mytable WHERE month = \" $ db_m \ "AND day = \" $ db_d \ ");
for ($ c = 0; $ c
{
```

$ F = mysql_fetch_array ($ q);
echo "$ f [email] $ f [name] $ f [month] $ f [day]
";
}

Як видно, використовувати функцію mysql_fetch_array () набагато зручніше, ніж mysql_fetch_row ().

Функції отримання інформації про результати SQL-запитів

PHP надає ще кілька корисних функцій, які дозволяють дізнатися інформацію про результат SQL-запитів.

• Функція mysql_field_name (int $ result, int $ offset) повертає ім'я поля, що знаходиться в результаті $ result з номером $ offset (нумерація починається з 0). Іншими словами, функція повертає ім'я поля з номером $ offset.

• Функція mysql_field_type (int $ result, int $ offset) повертає тип поля з номером $ offset в результаті $ result (номер задається щодо результату, а не таблиці);

• Функція mysql_field_flags (int $ result, int $ offset) повертає перерахування через пробіл прапори (модифікатори), які є у поля з номером $ offset. Перерахуємо всі підтримувані MySQL прапори:

12.4.6 Прапорець (flag) та його опис

not_Null Поле не може містити невизначеного значення (NULL), то поле повинно бути явно ініціалізоване. Primary_Key Поле буде первинним ключем - ідентифікатором запису, за яким можна однозначно ідентифікувати запис; auto_increment При вставці нового запису значення цього поля буде автоматично збільшено на одиницю, тому в таблиці ніколи не буде двох записів з однаковим значенням цього поля;

Unique_Key Поле має містити унікальне значення;

Multiple_Key Індекс

Blob Поле може містити бінарний блок даних

Unsigned Поле містить беззнакові числа

Zerofill Замість прогалин використовуються символи з кодом \ 0

Binary Поле містить двійкові дані

enum Поле може містити один елемент з декількох можливих (елемент перерахування)

timestamp У полі автоматично заноситься поточна дата і час при його модифікації

Функція mysql_field_flags () повертає прапори у вигляді рядка, в якій прапори поділяються пробілами.

12.4.7 Приклад використання функцій PHP-MySQL

Скрипт виведення вмісту таблиці MySQL у вигляді HTML:

```
<? Php
$ Host = "localhost";
$ User = "user";
$ Password = "secret_password";
/ / Виробляємо спробу підключення до сервера MySQL:
if (! mysql_connect ($ host, $ user, $ password))
{
echo "<h2> MySQL Error! </ h2>";
exit;
}
/ / Вибираємо базу даних:
mysql_select_db ($ db);
/ / Виводимо заголовок таблиці:
echo "<table border=\"1\" width=\"100%\"
bgcolor=\"#FFFFE1\">";
echo "<tr><td> Email </ td><td> Ім'я </ td><td> Місяць </ td>";
echo "<td> Число </ td><td> Пол </ td></ tr>";
/ / SQL-запит:
$ Q = mysql_query ("SELECT * FROM mytable");
/ / Виводимо таблицю:
for ($ c = 0; $ c
{
echo "<tr>";
$ F = mysql_fetch_array ($ q);
echo "<td> $ f [email] </ td><td> $ f [name] </ td><td> $ f
[month] </ td>";
echo "<td> $ f [day] </ td><td> $ [s] </ td>";
```

```
echo "</ tr>"; }
echo "</ table>";
?>
```

12.5 PDO – універсальний засіб роботи з MySQL вPHP

12.5.1 Визначення PDO

PHP Data Objects (PDO) - розширення для PHP, що надає розробнику простий і універсальний інтерфейс для доступу до різних баз даних.

PDO пропонує єдині методи для роботи з різними базами даних, хоча текст запитів може трохи відрізнятися. Так як багато СУБД реалізують свій діалект SQL, який в тій чи іншій мірі підтримує стандарти ANSI і ISO, то при використанні простих запитів можна домогтися сумісності між різними мовами. На практиці це означає, що можна досить легко перейти на іншу СУБД, при цьому не змінюючи або частково змінюючи код програми.

PDO не використовує абстрактних шарів для підключення до БД, на зразок ODBC, а використовує для різних БД їх «рідні» драйвери, що дозволяє добитися високої продуктивності. В даний час для PDO існують драйвери практично до всіх загальновідомим СУБД і інтерфейсів. Втім, є і драйвер для підключення до ODBC

Ім'я драйвера	**Підтримувані СУБД**
PDO_DBLIB	FreeTDS / Microsoft SQL Server / Sybase
PDO_FIREBIRD	Firebird / Interbase 6
PDO_IBM	IBM DB2
PDO_INFORMIX	IBM Informix Dynamic Server
PDO_MYSQL	MySQL 3.x / 4.x / 5.x
PDO_OCI	Oracle Call Interface
PDO_ODBC	ODBC v3 (IBM DB2, unixODBC і win32 ODBC)
PDO_PGSQL	PostgreSQL
PDO_SQLITE	SQLite 3 і SQLite 2
PDO_4D	4D

PDO входить до складу PHP 5.1, і поставлялося як PECL-розширення для PHP версії 5.0. У більш ранніх версіях PDO не працює, так як вимагає нових функцій ядра інтерпретатора

12.5.2 Приклад застосування PDO

Установка підключення до MySql:

```
try {
$ Db = new PDO ( 'mysql: host = localhost; dbname = testdb', $ login, $ passwd);
} Catch (PDOException $ e) {
echo $ e-> getMessage ();
}

// закриття з'єднання і звільнення ресурсів
$ Db = NULL;
```

PDO надає два способи виконання запиту:

• підготовка змінних - рекомендується для швидкості && безпеки

• пряме виконання

Запити які модифікують інформацію виконуються через метод exec ():

```
$ Db-> exec ( "INSERT INTO users (login) VALUES ( 'john')");
$ Id = $ db-> lastInsertId ();
$ Db-> exec ( "UPDATE user SET login = 'alex'")
```

Метод exec повертає значення - кількість порушених рядів, або FALSE у випадку помилки.

Запити які отримують інформацію, виконуються через метод query ():

```
$ Res = $ db-> query ( "SELECT * FROM users");
```

Значення, що повертається FALSE у випадку помилки.

Екранування спец символів (escaping значень): $ db-> quote ($ _ POST ['login'])

Можливі способи вибірки даних:

- масив (числовий або асоціативний індекс)
- рядок (для стовпця)

- об'єкти
- cllback функція
- лінива вибірка
- лінива вибірка
- ітератори

Вибірка масиву:

```
$ Res = $ db-> query ( "SELECT * FROM users");
while ($ row = $ res-> fetch (PDO :: FETCH_NUM)) {
// $ row - масив з числовими ключами
}
 $ Res = $ db-> query ( "SELECT * FROM users");
while ($ row = $ res-> fetch (PDO :: FETCH_ASSOC)) {
// $ row - асоціативний масив значень, ключі - назви стовпців
}
$ Res = $ db-> query ( "SELECT * FROM users");
while ($ row = $ res-> fetch (PDO :: FETCH_BOTH)) {
// $ row - числовий і ассоіатівний масив
}
```

Вибірка одного стовпчика:

```
$ Column = $ db-> query ( "SELECT id FROM users WHERE login = 'login' AND password = 'password'");
$ User = $ column-> fetchColumn ())
```

Вибірка об'єкта:

```
$ Res = $ db-> query ( "SELECT * FROM users");
while ($ obj = $ res-> fetch (PDO :: FETCH_OBJ)) {
// $ obj - екземпляр об'єкта stdClass, імена стовпців - властивості об'єкта
}
```

Вибірка у вигляді ітератора (інтерфейс Итератор)

```
$ Res = $ db-> query ( "SELECT * FROM users", PDO :: FETCH_ASSOC);
foreach ($ res as $ row) {
// $ row - асоціативний масив значень
}
```

Лінива вибірка - повертає результат у вигляді об'єкта, але фактичне наповнення об'єкта значенням відбувається тільки при першому зверненні до цього поля:

$ Res = $ db-> query ("SELECT * FROM users", PDO :: FETCH_LAZY);

foreach ($ res as $ row) {

echo $ row ['name']; // отримання значення

}

Отримання всіх значень вибірки відразу:

$ Qry = "SELECT * FROM users";

$ Res = $ db-> query ($ qry) -> fetchAll (PDO :: FETCH_ASSOC);

// $ res - масив всієї вибірки, кожен рядок представлена асоціативним масивом

Обробка cllback-функцією результату вибірки:

function add_salt ($ login, $ passw) {...}

$ Res = $ db-> query ("SELECT * FROM users");

$ Res-> fetchAll (PDO :: FETCH_FUNC, "add_salt");

Можливості запитів із заздалегідь підготовленим виконанням:

– компілюється один раз, виконується необхідну кількість разів

– чіткий розподіл між структурою і вхідними даними (запобігання SQL ін'єкція)

– швидке виконання в порівнянні з query () / exec (), навіть для одиничного виконання

Приклад запиту:

$ Stmt = $ db-> prepare ("SELECT * FROM users WHERE id =?");

$ Stmt-> execute (array ($ _ GET ['id']));

$ Stmt-> fetch (PDO :: FETCH_ASSOC);

Параметри для запиту можуть бути дані у вигляді імен та бути прив'язані до змінних:

$ Db-> prepare ('SELECT * FROM users WHERE name =: name AND email =: email');

```
$ Db-> execute (array ( ': name' => john, ': email' => 'john@domain.com'));
$ Result = $ db-> fetchAll ();
print_r ($ result);
$ Dbh-> execute (array ( ': name' => 'alex', ': email' => 'alex@domain.com'));
print_r ($ result);
// приклад використання методу bindParam ()
try {
$ Sql = $ db-> prepare ( "INSERT INTO USERS (name, email) VALUES (: name,: email)");
$ Sql-> bindParam ( ': name', $ name);
$ Sql-> bindParam ( ': email', $ email);
// призначення значень і вставка нового рядка
$ Name = 'Joy';
$email='joy@domain.com ';
$ Dbh-> exec ();
$ Name = 'debian';
$email='debian@domain.com ';
$ Dbh-> exec ();
}
catch (PDOException $ e) {
$ Dbh-> rollBack ();
echo 'Error:'. $ e-> getMessage ();
}
```

Транзакції. Майже всі драйвери PDO можуть працювати з транзакціями:

```
$ Db-> beginTransaction ();
if ($ db-> exec ($ qry) === FALSE) {
$ Db-> rollback ();
}
$ Db-> commit ();
```

Отримання meta інформації про запит - кількість стовпців і інформація про всіх шпальтах:

```
$ Res = $ db-> query ($ qry);
$ Ncols = $ res-> columnCount ();
```

```
for ($ i = 0; $ i <$ ncols; $ i ++) {
$ Meta_data = $ res-> getColumnMeta ($ i);
}
```

Розширення можливостей PDO:

```
class DB extends PDO
{
function query ($ qry, $ mode = NULL)
{
$ Res = parent :: query ($ qry, $ mode);
if (! $ res) {
var_dump ($ qry, $ this-> errorInfo ());
return null;
} Else {
return $ res;
}
}
}
```

12.6 Інструмент адміністрування СУБД MySQL - phpMyAdmin

phpMyAdmin - веб-додаток з відкритим кодом, написаний на мові PHP і представляє собою веб-інтерфейс для адміністрування СУБД MySQL. phpMyAdmin дозволяє через браузер здійснювати адміністрування сервера MySQL, запускати команди SQL і переглядати вміст таблиць і баз даних. Додаток користується великою популярністю у веб-розробників, оскільки дозволяє управляти СУБД MySQL без безпосереднього введення SQL команд, надаючи дружній інтерфейс.

На сьогоднішній день phpMyAdmin широко застосовується на практиці. Останнє пов'язано з тим, що розробники інтенсивно розвивають свій продукт, враховуючи всі нововведення СУБД MySQL. Переважна більшість російських провайдерів використовують цей додаток як панель управління для того, щоб надати своїм клієнтам можливість адміністрування виділених їм баз даних.

Додаток розповсюджується під ліцензією GNU General Public License і тому багато інших розробники інтегрують його у свої розробки, наприклад XAMPP, Denwer, AppServ.

Проект на даний момент часу локалізований на більш ніж 50-тьох мовах. Можливості phpMyAdmin може керувати як цілим MySQL сервером (для цього необхідні права суперкористувача), так і окремою базою даних. Можливо мультикористувацьке використання. В останньому випадку користувачі можуть користуватися тільки призначеним ним базами.

На даний момент phpMyAdmin дозволяє:

• створювати і видаляти бази даних

• створювати, копіювати, видаляти, перейменовувати та змінювати таблиці

• здійснювати супровід таблиць

• видаляти, редагувати і додавати поля

• виконувати SQL-запити, в тому числі пакетні SQL-запити

• керувати ключами

• завантажувати текстові файли в таблиці

• створювати (*) і переглядати дані таблиць

• експортувати (*) дані у форматах CSV, XML, PDF, ISO / IEC 26300 - OpenDocument Text and Spreadsheet, Word, Excel і LATEX

• адміністрування декількох серверів

• керувати користувачами MySQL та привілеями

• перевіряти цілісність посилальних даних у таблицях MyISAM

• використовувати запит за зразком (Query-by-example - QBE), створювати комплексні запити, автоматично з'єднуючись із зазначеними таблицями

• створювати графічну схему бази даних у форматі PDF

• здійснювати пошук в базі даних або в її розділах

• модифікувати збережені дані в різні формати, що використовуються в представлених функціях, наприклад, відображення BLOB-даних як зображень або як файли посилання і т.д.

• підтримує InnoDB таблиці та зовнішні ключі

• підтримує mysqli, покращене розширення MySQL
• перекладено більш ніж на 50 мов

Структура - На цій сторінці виводиться список таблиць даної бази даних, а також інструментарій для роботи із створеними таблицями. Саме на цій сторінці є блок створення нової сторінки.

SQL - Вікно для введення запитів до бази даних.

Експорт - Експортування даних.

Шукати - Пошук тексту по всіх таблицях даної бази даних.

Запит за прикладом - Інструментарій для складання запиту до бази даних.

Операції - Виконання операцій над базою даних, у тому числі: зміна назви БД, копіювання БД, зміна кодування БД.

Знищити - Видалити базу даних з сервера MySQL без можливості відновлення.

Спробуємо створити нову таблицю. Дамо їй назву "books" і в полі "поля" введемо значення 3, тобто саме стільки полів буде містити наша нова таблиця. Тиснемо "Пішов".

На сторінці бачимо 3 ряди полів, тобто це 3 поля. Для кожного поля треба проставити свої параметри:

Поле - Найменування поля. Заповнюйте латиницею, цифрами та знаком підкреслення, інші символи краще не використовувати

Тип даних - Для початку зупинимося на найбільш широковідомих:

• цілі числа. TINYINT - приймає значення від -128 до 127 або від 0 до 255. SMALLINT - приймає значення від -32768 до 32767 або від 0 до 65535. INT - приймає значення від -231 до 231 - 1 або від 0 до 232-1.

• числа з плаваючою крапкою. FLOAT - мінімальне значення + / -1.175494351-39, максимальне значення + / -3.402823466 +38.

• текстові поля. CHAR - рядок від 1 до 255 символів. VARCHAR будь-якої заданої довжини. TEXT - тестове поле.

Довжини / Значення - числове поле, в яке заповнюється довжина тестового поля, довжина числа і т.д.

Порівняння - кодування поля

Атрибути - для числових полів вказувати буде число беззнакове чи ні.

Нуль - якщо якесь поле буде використовуватися вкрай рідко, то можна задати що воно буде нульове.

За замовчуванням - значення за замовчуванням для поля.

Додатково - містить параметр AUTO_INCREMENT. Даний параметр задає, що при додаванні нових даних у таблицю дане поле буде збільшуватися на одиницю. Даний параметр необхідно використовувати для PRIMARY KEY.

Первинний – прапорець первинного ключа.

Індекс - даний прапорець відзначає, чи буде для цього поля створений індекс (що таке індекси, розглянемо в іншій статті).

Унікальний – прапорець попереджає систему, що дані в цьому полі повинні бути унікальними, тобто при додаванні даних в це поле, які вже будуть в БД, система відбракує цей запит і нові дані не будуть записані.

PS: перше поле ID завжди проставляють первинний + AUTO_INCREMENT.

Назвемо перше поле ID, тип проставимо INT і вкажемо що це первинний ключ з автоінкрементом.

Друге поле у нас буде містити назву книги, поле назвемо name з типом CHAR і довжиною 255.

Третє поле буде відповідати за автора книги, поле буде називатися autor з типом CHAR і довжиною 100.

Тиснемо "Пішов". Якщо Ви все заповнили правильно, то побачите, що з'явилася нова таблиця "books". У лівій частині під назвою бази даних також з'явилася назва нової таблиці "books", натиснемо на нього. Ми потрапимо на сторінку роботи з таблицями, крім описаних вище кнопок роботи з базою даних, з'являються ще 3 пункти:

Огляд - перегляд всіх внесених даних у таблицю.

Вставити - вставити дані в таблицю.

Очистити - очистити дані з таблиці, тобто видалення даних.

Натиснемо на "Вставити".

Ми бачимо поля для введення ID, найменування книги і автора. Оскільки ми ID зробили автоінкрементні, то заповнювати його не треба, система сама його заповнить.

Введемо в поле name значення "Збірка віршів А. С. Пушкіна", а в полі autor "А. С. Пушкін" і натиснемо "Пішов". Далі у верхньому меню натиснемо "огляд" і подивимося що у нас додалося:

Розділ 5 Приклади та типові задачі PHP

1. Базові обчислення і умовні оператори PHP

1.1 Математичні операції

1. Дано два числа 5 і 7. Знайти їх суму і твір.

Розв'язок:

```
1. <?php
2. $a = 5;
3.     $Ь = 7;
4. $c = $a + $Ь:
5.     $d= $a*$b;
6.     echo $c, $d;
7.     ?>
```

2. Дано два числа 4 і б. Знайдіть суму їх квадратів.
3. Дано три числа 3, 5, 8. Знайдіть їх середнє арифметичне.
4. Дано три числа x = 2, y = 6 і z = 9. Знайдіть (x + l) 4-2 (z-2xA2 + yA2)
5. Дано три ненульових числа $ a = 4. $b = 8. $ c = 3. Знайдіть всілякі результати ділення суми двох з них на час, що залишився третє число.
6. Дано два числа 17 і 54. Знайдіть суму 40% від першого числа і 84% від другого числа.
7. Дано тризначне числа. Знайдіть суму його цифр.

1.2. Умовний оператор

1. Дано число 15. Якщо воно більше 10, то збільште його на 100. інакше зменшіть на 30.

Розв'язок:

```
1. <?php
2.   $a = 15;
3.   if ($a>10)
4.   {$a = $a + 100};
5.   else
6.   {$a = $a - 30};
7. echo $a
```

8. ?>

2. Дано натуральне число 8. Якщо воно парне, то зменшите його в 2 рази, інакше збільште в 3 рази.

3. Дано число. Якщо воно не менше 50. то виведіть квадрат цього числа, якщо ж це число більше 10 і менше 30, то виведіть нуль, в інших випадках виведіть слово "Помилка".

4. Дано два числа $ a = 15, $ Ь = 4. Вивести найбільше з них.

5. Дано два числа $ a = 19, $ b = 143. Вивести ‘Так’, якщо вони відрізняються на 100. інакше вивести "Ні"

6. Дано два натуральних числа. Вивести ‘Так’, якщо вони відрізняються не більше ніж на 20, інакше вивести "Ні"

7. В даному тризначному числі переставте цифри так. щоб нове число виявилося найбільшим з можливих.

1.3. Робота з формою

1. Користувач вводить номер дня тижня. Вивести назву дня тижня.

Розв’язок:

```
1.<?php
2. if ($a = 1) {echo Monday};
3. if ($a = 2) {echo Tuesday};
4. if ($a = 3) {echo Wednesday};
5. if ($a = 4) {echo Thursday};
6. if ($a = 5) {echo Friday};
7. if ($a = 6) {echo $aturday};
8. if ($a = 7) {echo $unday};
9. else
10.      {echo "неверное число"}
11.      ?>
```

2. Користувач вводить свій вік. Якщо він більше 80 років, то вивести ‘Вітаю Вас, шановний’, інакше ‘Успіхів!’.

3. Користувач вибирає зі списку країн} ‘(Туреччина. Єгипет або Італія), вводить кількість днів для відпочинку і вказує, чи є у нього знижка (чекбокс). Вивести вартість відпочинку, яка

обчислюється як добуток кількості днів на 400. Датее це число збільшується на 10%, якщо обраний Єгипет, і на 12%, якщо обрана Італія. II далі це число зменшується на 5%, якщо вказана знижка.

2. Масиви в PHP

2.1 Масиви

1. Дано масив з елементами ‘Привіт,’, ‘світ’, ‘і’, ‘!’. Необхідно вивести на екран фразу ‘Прівіт, світ!’.

Розвязок:

Слово ‘Привіт,’ зберігається під номером 0. це означає, що для доступу до нього ми повинні написати $ arr [0].

Для доступу до слова ‘світ’ ми повинні написати $arr [l]. а $arr [2] містить в собі ‘!’. Далі за допомогою оператора ‘крапка’ ми складємо три наші рядки (‘Привіт,’, ‘світ’ і ‘!’) В один рядок таким чином: $arr [0].$arr [l].$arr [2]. і виведемо на екран за допомогою echo.

2. Дано масив з елементами ‘Привіт,’, ‘світ’ і ‘!’. Необхідно записати в змінну $text фразу ‘Привіт, світ!’, А потім вивести на екран вміст цієї змінної.

3. Дано масив з числами. Запишіть в новий масив тільки ті числа, в яких є цифра 5.

4. Створіть масив виду [1, [2], [[3]], [[[4]]]], замість 4 може бути .Любе ціле значення. Наприклад, якщо 5 - тоді буде масив [1, [2], [[3]], [[[4]]], [[[[5]]]]].

5. 5.1 Напишіть функцію, яка буде зливати два масиви таким чином: з. наприклад, [1, 2, 3] і [‘a’, ‘b’, ‘c’] вона зробить [1, ‘a’, 2, ‘b’, 3,’ c ‘].

5.2 Зробіть аналогічну функцію, яка параметрами буде приймати не два масиви, а довільну кількість (нехай функція параметром приймає двомірний масив, де його підмасиви - це те, що ми будемо зливати).

6. * Дано масив виду [1, ‘‘, 2, ‘‘, ‘‘, 3] - тобто в ньому є порожні рядки. Видаліть всі такі елементи з цього масиву.

7. ** Напишіть функцію, яка коректно буде зчитувати годинник і хвилини. Приклади: на вхід функції подається таке – 1г20хв + 50хв - в результаті функція виведе 2г10хв.

2.2 Асоціативні масиви

1. Створіть масив заробітних плат $arr. Виведіть на екран зарплату Колі.

Розвязок: щоб вивести зарплату Колі слід вивести значення елемента масиву з ключем ‘Коля’:

```
1. <?php
2. $arr = [‘Коля’=>‘1000$’, ‘Вася’=> ‘500$’,
‘Петро’=>‘200$’];
3. Echo $arr[‘Коля’];
4. ?>
```

2. Створіть масив $arr з елементами 1, 2, 3, 4, 5 двома різними способами.

3. Створіть масив $arr. a => 1, b => 2, c => 3. Виведіть на екран елемент з ключем ‘b’

4. Створіть масив $arr. Знайдіть суму елементів цього масиву. $arr = [‘a’ => 1, ‘b’ => 2, ‘c’ => 3];

5. Створіть асоціативний масив днів тижня. Ключами в ньому повинні служити номера днів від початку тижня (понеділок - повинен мати ключ 1, вівторок - 2 і т.д.). Виведіть на екран поточний день тижні. (Наприклад сьогодні - четвер)

2.3 Багатовимірні масиви

1. Дано багатовимірний масив. $arr = [
   ‘sp’=>[‘azul, ‘rojo’, ‘verde’],
   ‘en’=>[‘blue’, ‘red’, ‘green’],
   ];

Виведіть з його допомогою слово ‘azul’.

Розв’язок:

```
1. <?php
2. $arr = [
3. ‘sp’=>[‘azul’, ‘rojo’, ‘verde’],
```

4. ‘en’=>[‘blue’, ‘red’, ‘green’],
5.];
6. var_dump($arr[‘sp’]);
7. echo $arr[‘sp’][0]; *//виведе ‘azul’*

2. Створіть масив $arr = [‘a’, ‘b’, ‘c’]. Виведіть значення масиву на екран за допомогою функції var_dump() За допомогою масиву $arrз попереднього завдання виведіть на екран вміст першого, другого і третього елементів.

3. Створіть масив $arr = [‘a’, ‘b’, ‘c’, ‘d’] і з його допомогою виведіть на екран рядок ‘a+b, c+d ‘.

4. Створіть масив $arr з елементами 2, 7, 5, 4. Помножте перший елемент масиву на другий, а третій елемент на четвертий. Результати просумуйте і присвойте змінній $a. Виведіть на екран значення цієї змінної.

5. Заповніть масив $arr числами від 1 до 5. Не оголошуйте масив, а просто заповніть його привласненням $arr [] = нове значення.

6. Створіть двомірний масив. Перші два ключа - це ‘ukr’ і ‘en’. Нехай перший ключ містить елемент, який є масивом назв днів тижня українською, а другий - англійською. Виведіть за допомогою цього масиву понеділок українською і середу англійською (нехай понеділок - це перший день).

7. Доповніть попереднє завдання за умови, що у змінній $lang зберігається мова (вона приймає одне із значень або ‘ukr’. Або ‘en’), а в змінній$day - номер дня. Виведіть словом день тижня, відповідний змінним $lang і $day.

8. **Дано багатовимірний масив $arr. Напишіть фікцію, яка приймає рядок ‘string1.string2.string3’ - літери розділені крапками. А повертає елемент багатовимірного масиву $arr[‘string1][‘string2’][‘string3’]. Кількість точок в рядку може бути будь-якою, вкладеність масиву теж будь-яка.За умови, що ключі масиву не містять точок.

3 Задачі на ifelse, switch-case

1. Змінна $num може приймати одне із значень: 1. 2. 3 або 4. Якщо вона має значення ‘1, то в змінну $resultзапишемо ‘зима’, якщо має значення ‘2’ - ‘літо’ і так далі. Вирішіть задачу через switch-case.

Розв’язок:

```
1.  <?php
2.  $num = 2;
3.  switch ($num) {
4.  case 1:
5.  $result = ‘зима’;
6.  break;
7.  case 2:
8.  $result = ‘весна’;
9.  break;
10. case3:
11. $result =‘літо’;
12. break:
13. case 4:
14. $result =‘осінь’;
15. break:}
16. echo $result:
17. ?>
```

1. В змінній$day знаходиться якесь число з інтервалу від 1 до 31. Визначте в яку декаду місяця потрапляє це число (в першу, другу або третю).

2. В змінній $month знаходиться якесь число з інтервалу від 1 до 12. Визначте в яку пору року потрапляє цей місяць (зима, літо, весна, осінь).

3. В змінній $year зберігається рік. Визначте, чи є він високосним. *Підказка:* Рік буде високосним в двох випадках: або він ділиться на 4. але при цьому не ділиться на 100, або ділиться на 400. Так. роки 1700, 1800 і 1900 не є високосними, так як вони діляться на 100 і не діляться на 400. Роки 1600 і 2000 - високосні, так як вони діляться на 400.

4. Дано рядок з символами, наприклад, ‘abcde’. Перевірте, що першим символом цього рядка є буква ‘a’. Якщо це так - виведіть "так", в іншому випадку виведіть "ні".

5. Дано рядок з цифрами, наприклад, ‘12345’. Перевірте, що першим символом цього рядка є цифра 1, 2 або 3. Якщо це так - виведіть "так", в іншому випадку виведіть "ні".

6. Дано рядок з 3-х цифр. Знайдіть суму цих цифр. Тобто складіть як числа перший символ рядка, другий і третій.

7. Дано рядок з 6-ти цифр. Перевірте, що сума перших трьох цифр дорівнює сумі других трьох цифр. Якщо це так - виведіть "так", в іншому випадку виведіть "ні".

4 Цикли в PHP

4.1 Робота з while, for

1. Заповніть масив 10-ю іксами за допомогою циклу

Розв’язок:

```
1. <?php
2. $arr = []:
3. for ($i = l; $i<= 10; $i++) {
4. $arr[] = ‘x’;
5. }
6. var_dump($arr);
7. ?>
```

2. Заповніть масив числами від 1 до 10 за допомогою циклу.
3. Заповніть масив числами від 10 до 1 за допомогою циклу.
4. Виведіть стовпець чисел від 1 до 100.

4.2 Робота з foreach

5. Дано масив з елементами ‘html’, ‘css’, ‘php’, ‘js’. За допомогою циклу foreach виведіть ці слова в стовпчик.

6. Дано масив з елементами 1. 2, 3, 4, 5. За допомогою циклу foreach знайдіть суму елементів цього масиву. Запишіть її в змінну $result.

7. Дано масив з елементами 1, 2, 3, 4, 5. За допомогою циклу foreach знайдіть суму квадратів елементів цього масиву. Результат запишіть змінну $result.

8. Дано масив $arr. $arr = [‘green’ => ‘зелений’. ‘red’ =>‘ червоний ‘,’ blue ‘=>‘ блакитний ‘]; За допомогою циклу foreach виведіть на екран стовпець ключів і елементів в форматі ‘green - зелений’.

9. Дано масив $air з ключами ‘Коля’. ‘Вася’, ‘Петя’ і з елементами ‘200’, ‘300’, ‘400’. За допомогою циклу foreach виведіть на екран стовпець рядків такого формату: ‘Коля - зарплата 200 доларів.’

5 Рядки в PHP

5.1 Робота з реєстром символів

Для вирішення завдань даного блоку вам знадобляться наступні функції: strtolower, strtoupper, ucfirst, Icfirst, ucwords.

1. Дано рядок ‘php’. Зробіть з нього рядок ‘PHP’

Розв’язок:

```
1. <?php
2. echo strtolower('PHP');
3. ?>
```

2. Дано рядок ‘London’. Зробіть з нього рядок ‘ london ‘.
3. Дано рядок ‘london is the capital of great britain’. Зробіть з нього рядок ‘London Is The Capital O fGreat Britain’.
4. Дано рядок ‘LONDON’. Зробіть з нього рядок ‘London’.

Робота з strlen

5. Дано рядок ‘html css php’. Знайдіть кількість символів в цьому рядку.
6. Дано змінну$password. в якій зберігається пароль користувача. Якщо кількість символів пароля більше 5-ти і менше 10-ти. то виведіть користувачеві повідомлення про те, що пароль підходить, інакше повідомлення про те, що потрібно придумати інший пароль.

Робота з substr

7. Дано рядок ‘html css php’. Виріжте з нього і виведіть на екран слово ‘html’, слово ‘css’ і слово ‘php’.
8. Дано рядок. Виріжте і виведіть на екран останні 3 символу цього рядка.

9. Дано рядок. Перевірте, що він починається на ‘http: //’. Якщо це так. виведіть "так", якщо не так

10. Дано рядок. Перевірте, що він починається на ‘http: //’ або на ‘https: //’. Якщо це так. виведіть "так", якщо не так - "ні".

11. Дано рядок. Перевірте, що він закінчується на ‘.png’. Якщо це так. Виведіть"Так", якщо не так - "ні".

12. Дано рядок. Перевірте, що він закінчується на ‘.png’ або на ‘.jpg’. Якщо це так. виведіть "так", якщо не так - "ні".

13. Дано рядок. Якщо в цьому рядку більше 5-ти символів виведіть з нього перші 5 символів, додайте три крапки в кінець і виведіть на екран. Якщо ж в цьому рядку 5 і менше символів - просто виведіть цей рядок на екран.

Робота з str_replace

14. Дано рядок ‘31.12.2013 ‘. Замініть всі крапки на дефіси.

15. Дано рядок $str. Замініть в ньому всі букви ‘a’ на цифру 1, літери ‘b’ - на 2, а літери ‘c’ - на 3.

16. Дано рядок з буквами і цифрами, наприклад, ‘1a2b3c4d5e6f7g8h’. Видаліть з нього всі цифри. Тобто в нашому випадку повинна вийде рядок ‘abcbdefgh’.

Робота з subsn_replace

17. Дано рядок $str. Виріжте з нього підрядок з 3-го символу (відлік з нуля). 5 штук і замість нього вставте ‘!!!’.

Робота з strpos, stnpos

18. Дано рядок ‘abc abc abc’. Визначте позицію першої літери ‘b’.

19. Дано рядок ‘abc abc abc’. Визначте позицію останньої літери ‘b’.

20. Дано рядок ‘abc abc abc’. Визначте позицію першої знайденої літери ‘b’, якщо почати пошук не з розпочатку рядка, а з позиції 3.

21. Дано рядок ‘aa aa aa aa’. Визначте позицію другого пробілу.

22. Перевірте, що в рядку є дві точки поспіль. Якщо це так - виведіть ‘є’, якщо не так - "ні".

23. Перевірте, що рядок починається на ‘http: //’. Якщо це так - виведіть "так", якщо не так - "ні".

Робота з explode, implode

24. Дано рядок ‘html css php’. За допомогою функції explode запишіть кожне слово цього рядка в окремий елемент масиву

25. Дано масив з елементами ‘html’, ‘css’, ‘php’. За допомогою функції miplode створіть рядок з цих елементів, розділених комами.

26. В змінної $date знаходиться дата в форматі ‘2016-12-31’. Перетворіть цю дату в формат ‘31.12.2016 ‘.

Робота з str_split

27. Дано рядок ‘1234567890’. Розбийте її на масив з елементами ‘12’, ‘34’, ‘56’, ‘78’, ‘90’.

28. Дано рядок ‘1234567890’. Розбийте її на масив з елементами ‘1’, ‘2’, ‘3’, ‘4’, ‘5’, ‘6’, ‘7’, ‘8’, ‘9’, ‘0’.

29. Дано рядок ‘1234567890’. Зробіть з нього рядок ‘12-34-56-78-90’ не використовуючи цикл.

Робота з trim, ltrimltrim

30. Дано рядок. Очистіть його від кінцевих пробілів.

31. Дано рядок ‘/ php /’. Зробіть з нього рядок ‘php’, видаливши кінцеві слеші.

32. Дано рядок ‘слова слова слова.’. В кінці цього рядка може бути крапка, а може і не бути. Зробіть так. щоб в кінці цього рядка гарантовано стояла крапка. Тобто: якщо цієї точки немає - її треба додати, а якщо є - нічого не робити. Завдання - вирішите через rtrim без if.

Робота з str_rev

33. Дано рядок ‘12345’. Зробіть з нього рядок ‘54321’.

34. Перевірте, чи є слово паліндромом (однаково читається у всіх напрямках, приклади таких слів: madam, otto, kayak, nun, level).

Робота з str_shuffle

35. Дано рядок. Перемішайте символи цього рядка у випадковому порядку.

36. Створіть рядок з 6-ти випадкових маленьких латинських літер так. щоб букви не повторювалися. Потрібно

зробити так, щоб в нашій рядку могла бути будь-яка латинська буква, а не обмежений набір.

Робота з number_fomat

37. Дано рядок ‘12345678’. Зробіть з нього рядок ‘12 345 678 ‘.

Робота з str_repeat

38. Намалюйте піраміду, як показано на малюнку, тільки у вашій піраміди має бути 9 рядів, а не 5. Вирішіть задачу за допомогою одного циклу і функції str_repeat.

39. Намалюйте піраміду з символів. Вирішіть задачу за допомогою одного циклу і функції str_repeat.

Робота з stnp_tags і html_specialchars

40. Дано рядок ‘html, <b>php</ b>, js’. Видаліть теги з цього рядка.

41. Дано рядок $str. Видаліть всі теги з цього рядка, крім тегів <b> і <i>.

42. Дано рядок ‘html, <b>php</ b>, js’. Виведіть її на екран ‘як є’: тобто браузер не повинен перетворити <b> в жирний.

Робота з ehr і ord

43. Дізнайтеся код символів ‘a’, ‘b,’ с ‘, пробілу.

44. Вивчіть таблицю ASCII. Визначте межі, в яких розташовуються букви англійського алфавіту. Виведіть на екран символ з кодом 33.

45. Запишіть в змінну $str випадковий символ - велику букву латинського алфавіту. Підказка: за допомогою таблиці ASCII визначте які цілі числа відповідають великим літерами латинського алфавіту.

46. Запишіть в змінну $str випадковий рядок довжиною $len, що складається з маленьких літер латинського алфавіту. Підказка: скористайтеся циклом for або while.

47. Дано буква англійського алфавіту. Дізнайтеся, він маленька або велика.

Робота з stick, strrchr

48. Дано рядок ‘ab-cd-ef’. За допомогою функції strehr виведіть на екран рядок’cd-ef’

49. Дано рядок ‘ab-cd-ef’. За допомогою функції strrchr виведіть на екран рядок ‘-ef’.

Робота з strstr

50. Дано рядок ‘ab-cd-ef’. За допомогою функції strstr виведіть на екран рядок ‘-cd-ef’.

5.2 Застосування рядків у PHP

1. Користувач вводить назви міст через пробіл. Переставте назви так, щоб назви були впорядковані за алфавітом.
2. Перетворіть рядок ‘var_test_text’ в ‘varTestText’. Скрипт, звичайно ж, має працювати з будь-якими аналогічними рядками.
3. Дано рядок наступного виду: 5 цифр, потім пробіл, потім ще 5 цифр. Наприклад, Дано такий рядок $str = ‘12345 67890’. Зробіть з нього рядок ‘54321 ‘;
4. Дано масив з числами. Виведіть на екран всі числа, в яких є цифра 3.
5. Дано масив з числами. Порахуйте сумарна кількість цифр 3 в цих числах.
6. Дано рядок з цифрами, наприклад: ‘12345678’. Підсумуйте числа цього рядка ось таким чином: 12 + 34 + 56 + 78. Рядок, звичайно ж, може бути довільною (але тільки з цифрами всередині)

6 Файли в PHP

6.1 Робота з файлами

1. Створіть файл ‘test.txt’ і запишіть в нього фразу ‘Привіт, світ!’.

Розвязок:

```
1. <?php
2. file_put_contents (‘test.txt’, ‘Привіт, світ!’);
3. ?>
```

2. Зчитайте дані з файлу ‘test.txt’ і виведіть їх на екран.
3. Перейменуйте файл ‘test.txt’ в ‘myfile.txt’.
4. Зробіть папку ‘folder’, та перемістіть в неї файли ‘test.txt’ та ‘myfile.txt’.

5. Створіть копію файлу ‘myfile.txt’ і назвіть її ‘otherfile.txt’.

6. Визначте розмір файлу ‘otherfile.txt’. Виведіть його на екран. Виведіть його розмір в байтах, мегабайтах, гігабайтах.

7. Видаліть файл ‘otherfile.txt’.

8. Перевірте наявність (існування) файлів ‘otherfile.txt’ і ‘myfile.txt’.

6.2 Робота з папками

Робота з mkdir, rmdir

9. Створіть папку ‘test’.

10. Перейменуйте папку ‘test’ на ‘www’

11. Видаліть папку ‘www’.

12. Дано масив з рядками. Створіть в папці ‘test’ папки, назвами яких слугують елементи цього масиву.

Робота з scandir, is_dir, is_file, PHP_EOL

13. Виведіть иа екран назву всіх файлів і підпапок з папки ‘test’.

14. Виведіть на екран назву всіх файлів, але не підпапок з папки ‘test’.

15. В папці ‘test’ є файли і папки. Виведіть на екран вміст всіх файлів, які лежать безпосередньо в папці ‘test’.

16. Виведіть на екран назву всіх файлів з розширенням txt з папки ‘test’.

17. Знайдіть всі файли з папки ‘test’ і вставте в початок кожного файлу повний шлях до нього (текст файлу повинен залишитися в ньому і починатися з нового рядка).

18. Виведіть на екран імена всіх підпапок з папки ‘test’ і їх підпапок (може бути будь-який рівень вкладеності).

19. Виведіть на екран вміст всіх файлів з папки ‘test’ і її підпапок (може бути будь-який рівень вкладеності).

20. Знайдіть всі файли з папки ‘test’ і її підпапок будь-якого рівня вкладеності і вставте на початку кожного файлу повний шлях до нього (текст файлу повинен залишитися в ньому і починатися з нового рядка).

21. Знайдіть всі файли з папки ‘test’, в їх вмісті знайдіть тег <h1> текст </h1>. Перейменуйте всі файли на вміст між тегами <h1>.

22. Видаліть з папки ‘test’ всі файли розміром більше 1Мб.

23. Є папка з файлами, дізнайтеся розмір цієї папки.

24. Є папка з підпапками. Дізнайтеся розміри всіх підпапок папки і виведіть їх на екран.

7 Cookie в PHP

1. Ініціюйте змінну для підрахунку кількості відвідувань. Якщо відповідні дані передавалися через cookie зберігайте їх в цю змінну. Наростити лічильник відвідувань. Встановіть відповідні cookie.

Розв’язок:

```
<?php
$visit_counter = 0;
if (isset($_COOKIE [‘visitCounter’ ]) &&
is_numeric($_COOKIE[‘visitCounter’])){
$visit counter = $_COOKIE[‘visitCounter ];
$visit_counter++;
}
setcookie ('visitCounter', $visit counter);
?>
```

2. Ініціюйте змінну для зберігання значення останнього відвідування сторінки. Якщо відповідні дані передавалися з cookie, відфільтруйте їх і збережіть в цю змінну. Встановіть відповідні cookie.

3. Виведіть інформацію про кількість відвідувань і дату останнього відвідування.

8 Завдання на Session в PHP

1. Зробіть дві сторінки: index.php і hello.php. При вході на index.php запитується за допомогою форми ім’я користувача, запишіть його в cookie. При заході на hello.php користувач вітається фразою "Привіт,% І’мя%!" .

Розвязок:

Сторінка index.php:

```
1. <form action= "" method ="GET ">
2. <input type = "text" name = "username">
3. <input type = "subinit">
4. </ form>

5. <? Php
6. // Якщо форма була відправлена і ім’я не пусте:
7. if (! Empty ($_REQUEST [ ‘username’])) {
9. // Пишемо ім’я в куки:
10. setcookie ( ‘username’, $_REQUEST [ ‘username’], time() + 3600, ‘/’);
11. ?>
```

Сторінка hello.php:

```
1. <? Php
2. // Якщо є дані в cookie про ім’я користувача:
3. if (! empty ($_COOKIE [ ‘username’])) {
4. echo $_COOKIE [ ‘username’]; // виведемо ім’я на екран
5. }
6. ?>
```

2. Запитайте у користувача телефон за допомогою форми. Потім зробіть так щоб в іншій формі (поля: ім’я. прізвище, телефон) при її відкритті поле телефон було автоматично заповнено.

3. Створіть три сторінки з формами, що містять по одному полю введення на сторінку. Оброблювач кожної наступної форми буде вести на наступну сторінку. На першій сторінці запитаємо ім’я, на другій сторінці прізвище, на третій вік, а на четвертій виводяться всі дані використовуючи сесію.

4. Створіть в сесії масив для зберігання всіх відвіданих сторінок і збережіть в якості його чергового елементу шлях до поточної сторінки. Виведіть в циклі журнал відвіданих користувачами сторінок.

9 Бази даних MySQL в PHP

1. Дано масив з таблицями (‘user’, ‘profile’), іменами полів (id. Name і так далі) і їх типами:

$tables = array (‘user’ => [‘id’ => ‘int’, ‘name’ => ‘varchar 256’], ‘profile’ => [‘id’ => ‘int’, ‘value’ => ‘text’]);

Відкрийте цей масив в базу даних (створіть в базі даних таблиці з заданими іменами і полями).

Розв’язок:

```
1.  <?php
2.  $tables = array (‘user’=>[‘id’=>‘int’, ‘name’=>‘varchar 256’], ‘profile’=>[‘id’=>‘int’, ‘value’=>‘text’]);
3.  function tables ($tables) {
4.  $connect = new mysqli(‘$RV,’LOGIN’,’P$WD’,’DB’);
5.  foreach ($tables as $key=>$cols){
6.  $connect->query("CREATE TABLE $key(ololo int (10))");
7.  foreach ($cols as $col=>$value){
8.  $value=explode(‘ ‘, $value);
9.  if(empty($value[l])) $value[l]=50;
10. $connect->query("ALTER TABLE $key .ADD $col $value[0]($value[l]) NOT NULL");
11. }
12. $connect->query("ALTER TABLE $key DROP ololo");
13.  }
14. tables($tables);
15. ?>
```

2. Сформувати базу даних "облікові відомості про учнів", що включає такі поля: Прізвище, Клас, Рік вступу, Позаурочна секція. Ввести в базу даних умовну інфрмацію про 5 учнів.

3. Написати програму перегляду бази даних і додавання в неї нового запису. В результаті має бути отримано три файли index.php. mysql.php. view.php.

10 Робота з класами

1. Дано папка з картинками. При оновленні сторінки виведіть картинку випадковим чином. Вирішіть два завдання:

a. Зображення можуть повторюватися.

b. Показана картинка не буде повторюватися поки не будуть показані всі картинки з папки. Тобто все картинки з папки потрібно показати тільки 1 раз, але випадковим чином. Коли всі будуть показані - починаємо заново: показуємо по другому колу, але тільки 1 раз.

Розвязок:

```
1. <?php
2. $dir="img";
3. $images = scandir($dir); //отримуємо перелік картинок в папці
4. unset ($images [0]);
5. unset ($images[l]);
6. $images = array_values($miages);
7. $i=rand(0,(count($images)-l));

8. class ImgDB {
9. const DB_NAME = "rotator.db";
10. private $_db:

11. function _construct(array $data){ //створюємо, базу даних картинок
12. if(is_file(self::DB_NAME)){
13. $this->_db = new $QLite3(self::DB_NAME);
14. }else{
15. $this->_db = new $QLite3(self::DB_NAME);
16. $sql= ‘CREATE TABLE imgs(id INTEGER PRIMARY KEY AUTOINCREMENT.filename TEXT)’;
17. $this->_db->exec($sql) or die($this->_db->lastErrorMsg());
18. foreach ($data as $k=>$v){
19. $sql = "INSERT INTO imgs(filename) VALUES (‘ $v’);";
```

20. $this->_db->exec($sql) or die($this->_db->lastErrorMsg());
21. }

22. ***public function*** *showImgOnce($data){*
23. $sql= ‘$ELECT * FROM imgs ORDER BY RANDOM() LIMIT 1;’;
24. $res=$this->_db->query($sql) or die($this->_db->lastErrorMsg());
25. $res = $this->db2arr($res);
26. $id = $res[0][‘id’];
27. $sql= "select count(*) from imgs:";
28. if (($this->_db->queiy($sql))== 1){
29. foreach ($data as $k=>$v){
30. $sql= "IN$ERT INTO nngs(filename) VALUES (‘ $v’);";
31. $this->_db->exec($sql) or die($this->_db->lastErrorMsg());
32. }
33. $sql= "DELETE FROM imgs WHERE id=$id";
34. $del=$this->_db->exec($sql) or die($this->_db->lastErrorMsgO);
35. retimi $res[0]["filename"];
36. }
37. ***protected function*** *db2arr($res){*
38. $arr= array():
39. while($row=$res->fetchArray(SQLITE3_ASSOC))
40. $arr[] = $row;
41. return $arr;
42. }
43. }

44. $imgdata = **new** ImgDB($images); *//створюємо об’єкт класу*

45. echo‘<img src="/pictures/’.$dir.’/’.$imgdata->showImgOnce($images).’">’;

46. ?>

11. Завдяння для саморозвитку

1. Написати функцію, яка перетворює такі рядки var_php_test в varPhpTest.

2. Числа Фібоначі - це ряд чисел, в якому кожне наступне число дорівнює сумі двох попередніх чисел. Виведіть на екран перші 100 чисел.

3. Обріжте довгий рядок так, щоб його довжина була не більше N символів, при цьому слова не повинні бути розірвані - обрізання завжди проходить по пробілові і новий рядок завжди менше N. Приклад: $ n = 8; $str = ‘abcde abc abc abcde’ => ‘abcde
 abc abc
 abcde’;

4. Реалізуйте трикутник Паскаля довільного розміру (повинен намалюватися в браузері).

5. Напишіть функцію, яка визначає стать по ПІБ.

6. Зробіть функцію, яка буде приймати 2 числа, а повертати їх найменше спільне кратне.

7. Напишіть функцію, яка знаходить різницю між двома датами в роках, місяцях, днях, хвилинах, годинах, секундах.

8. Реалізуйте функцію flatten, яка в разі, якщо масив володіє рівнями вкладення, приведе його до елементарного виду (вкладеність може бути будь-якої глибини). ***Приклад:*** flatten ([l, [2]. [3, [[[4]]]]]) поверне [1, 2, 3, 4].

9. Дано папка з файлами і підпапками. В підпапках можуть бути свої папки і файли і так далі. Виведіть на екран шляхи до всіх файлів, розташованих в цих папках.

Список рекомендованої літератури

1. Зандстра Мэтт PHP: объекты, шаблоны и методики программирования, 5-е изд. : Пер. с англ. – СПб. : ООО “Диалектика", 2019. – 736 с.

2. Кузнецов М. Самоучитель PHP 7/ Максим Кузнецов, Игорь Симдянов – СПб.: БХВ-Петербург. – 2018. – 448 с. ISBN: 978-5-9775-3817-6

3. Маклафин Б. PHP и MySQL. Исчерпывающее руководство. – СПб.: Питер, 2013. – 512 с.: ил. ISBN 978-5-459-01550-8

4. Никсон Р. Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript и CSS. 2-е изд. – СПб.: Питер, 2013. – 560 с.: ил. – (Серия «Бестселлеры O’Reilly»). ISBN 978-5-496-00187-8

5. Hopkins Callum Jump Start PHP / Callum Hopkins // SitePoint Pty. – 2013 [Електронне видання] ISBN 978-0-9874674-1-6 (ebook)

6. Meloni J. PHP, MySQL & JavaScript All in One, Sixth Edition / Pearson: The world's learning company. Sams Teach Yourself [Електронне видання] Джерело: www.pearson.com ISBN-13: 978-0-672-33770-3 ISBN-10: 0-672-33770-3

7. Tatroe Kevin Programming PHP, Third Edition / Kevin Tatroe, Peter MacIntyre, and Rasmus Lerdorf // O’Reilly Media, Inc. – 2013 [Електронне видання] ISBN 978-1-449-39277-2

Мережеві ресурси

1. Колос В. Підручник з PHP / Віталій Колос // [Електронний ресурс] Режим доступу: http://maque.org.ua/wordpress/?p=10615

2. Ленгсторф Джейсон PHP и jQuery для профессионалов [Електронний підручник] Джерело: www.apress.com.

3. https://www.apserver.org.ua/ - Підручник PHP

4. http://www.php.su/ – PHP, MySQL та інші веб-технології

5. https://phpbuilder.ru/ua/learn – PhpBuilder.ru ваш надійний путівник по веб програмуванню

6. http://php720.com/ - основи розробки PHP

7. https://websunsea.github.io/ - W3Schools:Уроки по PHP

8. https://w3schoolsua.github.io/php/index.html - PHP підручник

Навчальне видання

Кіндрат П.В.

Розробка веб-застосувань засобами PHP

Навчально-методичний посібник

Технічний редактор:
Кіндрат П.В.

Комп'ютерна верстка та макет:
Катерина Стецюк

Підписано до друку 05.03.2021р. Формат 60x84 1/16 папір офсет.
Гарнітура «Times». Друк офсет. Ум. друк. арк. 21.8.

Наклад 100 пр.
Редакційно-видавничий відділ
громадської організації „СОМ-ЦЕНТР"
м.Рівне, вул. С.Бандери, 31-А/1

310