МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний аерокосмічний університет ім. М. Є. Жуковського

«Харківський авіаційний інститут»

факультет програмної інженерії та бізнесу

кафедра інженерії програмного забезпечення

**Прктична робота № 2**

з дисципліни « Web-програмування мовою PHP »

*назва дисципліни*

на тему: «ВСТУП В PHP, ОСНОВИ СИНТАКСИСУ, ТИПИ ДАНИХ, ОПЕРАТОРИ, КЕРУЮЧІ КОНСТРУКЦІЇ»

Виконав: студент 3 курсу групи № 632п

освітньої програми

121 інженерія програмного забезпечення

(шифр і назва ОП)

Зайченко Ярослав Ігорович

(прізвище й ініціали студента)

Прийняв: старший викладач

Дем'яненко Владислав Анатолійович

(посада, науковий ступінь, прізвище й ініціали)

Кількість балів:

Харків – 2024

**ЗМІСТ**

[Постановка завдання 3](#_Toc179963230)

[Порядок виконання роботи 3](#_Toc179963231)

[Зміст звіту 3](#_Toc179963232)

[Теоретичні відомомості 4](#_Toc179963233)

[Виконання роботи 8](#_Toc179963234)

[Заключення 12](#_Toc179963235)

[*Додаток А* 13](#_Toc179963236)

[*Додаток Б* 14](#_Toc179963237)

[*Додаток В* 15](#_Toc179963238)

Мета роботи: вивчення синтаксису мови PHP, ознайомлення з типами даних, операторами, які керують конструкціями а також отримання практичних навичок щодо складання серверних скриптів для обробки запитів користувачів.

Постановка завдання

Розробити та реалізувати на мові PHP серверний сценарій (скрипт) для обробки запитів користувачів і представлення результатів у вигляді документа HTML.

**Завдання варіанту 3:**

Вихідна форма документа повинна містити заголовок з назвою роботи, внизу документа - дата генерації документа (оновлювана при кожному запиті) і ПІБ виконавця.

«Міні-калькулятор». Елементарні математичні дії над двома числами (5 і більше дій). Що треба зробити вибирається зі списку або через radiobutton. Забезпечити перевірку інформації, що вводиться на коректність.

Порядок виконання роботи

1. Отримати у викладача індивідуальне завдання для виконання роботи.
2. Вивчити теоретичний матеріал.
3. Визначити завдання, які повинені вирішуватися розробляємим сценарієм.
4. Розробити інтерфейс користувача у вигляді HTML форми, а також кінцевий вигляд, який генерується у вигляді HTML з результатами роботи сценарію.
5. Реалізувати на мові PHP спроектований сценарій.
6. Протестувати локально розроблений сценарій.
7. Зробити висновки по роботі.

Зміст звіту

1. Постановка задачі. Опис завдань, що вирішуються серверним сценарієм.
2. Короткий опис алгоритму роботи сценарію.
3. Опис використовуваних вхідних даних і функцій.
4. Лістинг вихідного коду сценарію з коментарями а також результати його роботи (скріншот або текстове представлення).
5. Висновки по роботі.

Теоретичні відомомості

Історія PHP

Мова PHP був розроблений як інструмент для вирішення суто практичних завдань. Його творець, Расмус Лердорф, хотів знати, скільки людей читають його online-резюме, і написав для цього простеньку CGI-оболонку на мові Perl, тобто це був набір Perl-скриптів, призначених виключно для певної мети - збору статистики відвідувань.

Для довідки. CGI (Common Gateway Interface - загальний інтерфейс шлюзів) є стандартом, який призначений для створення серверних додатків, що працюють по протоколу HTTP. Такі додатки (їх називають шлюзами або CGI-програмами) запускаються сервером в режимі реального часу. Сервер передає запити користувача CGI-програмі, яка їх обробляє і повертає результат своєї роботи на екран користувача. Таким чином, відвідувач отримує динамічну інформацію, яка може змінюватися в результаті впливу різних чинників. Сам шлюз (скрипт CGI) може бути написаний на різних мовах програмування - Сі / C ++, Fortran, Perl, TCL, UNIX Shell, Visual Basic, Python і ін.

Можливості PHP

В першу чергу PHP використовується для створення скриптів, що працюють на стороні сервера, для цього його, власне, і придумали. PHP здатний вирішувати ті ж завдання, що і будь-які інші CGI-скрипти, в тому числі обробляти дані html-форм, динамічно генерувати html сторінки і т.п. Але є й інші області, де може використовуватися PHP. Всього виділяють три основні області застосування PHP.

• Перша область, як вже говорилося, - це створення додатків (скриптів), які виконуються на стороні сервера. PHP найбільш широко використовується саме для створення такого роду скриптів. Для того щоб працювати таким чином, знадобиться PHP-парсер (тобто обробник php-скриптів) і web-сервер для обробки скрипта, браузер для перегляду результатів роботи скрипта, ну, і, звичайно, будь-якої текстовий редактор для написання самого php-коду.

• Друга область - це створення скриптів, що виконуються в командному рядку. Тобто за допомогою PHP можна створювати такі скрипти, які будуть виконуватися, незалежно від web-сервера і браузера, на конкретній машині. Для такої роботи потрібно лише парсер PHP (в цьому випадку його називають інтерпретатором командного рядка (cli, command line interpreter)). Цей спосіб роботи підходить, наприклад, для скриптів, які повинні виконуватися регулярно за допомогою різних планувальників завдань або для вирішення завдань простий обробки тексту.

• І остання область - це створення GUI-додатків (графічних інтерфейсів), що виконуються на стороні клієнта. В принципі це не найкращий спосіб використовувати PHP, особливо для початківців, але якщо ви вже досконально вивчили PHP, то такі можливості мови можуть виявитися вельми корисні. Для застосування PHP в цій області буде потрібно спеціальний інструмент - PHP-GTK, який є розширенням PHP.

Отже, сфера застосування PHP достатньо обширна і різноманітна. Проте існує безліч інших мов програмування, здатних вирішувати схожі завдання. Чому варто вивчати PHP? По-перше, PHP дуже простий у вивченні. Достатньо ознайомитися лише з основними правилами синтаксису і принципами його роботи, і можна починати писати власні програми, причому братися за такі завдання, вирішення яких на іншій мові вимагало б серйозної підготовки.

По-друге, PHP підтримується майже на всіх відомих платформах, майже у всіх операційних системах і на самих різних серверах. Це теж дуже важливо. Навряд чи комусь захочеться переходити, наприклад, від роботи під Windows до роботи під Linux або від сервера IIS до сервера Apache тільки для того, щоб вивчити ще одну мову програмування.

У PHP поєднуються дві найпопулярніші парадигми програмування - об'єктна і процедурна. В PHP4 більш повно підтримується процедурне програмування, але є можливість писати програми і в об'єктному стилі. Уже в перших пробних версіях PHP5 більшість недоліків в реалізації об'єктно-орієнтованої моделі мови, існуючих в PHP4, усунені. Таким чином, можна вибрати найбільш звичний стиль роботи.

**Основи синтаксису**

Основний синтаксис

Перше, що потрібно знати щодо синтаксису PHP, - це те, як він вбудовується в HTML-код, як інтерпретатор довідається, що це код на мові PHP. Найчастіше використовується варіант <? Php?>, І іноді скорочений варіант <? ?> (В залежності від налаштувань сервера).

Поділ інструкцій

Програма на PHP (та й будь-якою іншою мовою програмування) - це набір команд (інструкцій). Оброблювачу програми (парсеру) необхідно якось відрізняти одну команду від іншої. Для цього використовуються спеціальні символи - роздільники. У PHP інструкції розділяються так само, як і в Сі або Perl, - кожен вираз закінчується крапкою з комою.

Закриває тег «?>» Також має на увазі кінець інструкції, тому перед ним крапку з комою не ставлять.

Коментарі

PHP підтримує кілька видів коментарів: в стилі Сі, C ++ і оболонки Unix. Символи // і # позначають початок однорядкових коментарів, / \* і \* / - відповідно початок і кінець багаторядкових коментарів.

Змінні

Зміна 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 і інші назви в PHP: правильне ім'я змінної повинно починатися з букви або символу підкреслення з подальшими в будь-якій кількості буквами, цифрами або символами підкреслення. приклад:

<? Php

$ First = 'Text'; // Надаємо $ first

// значення

// 'Text'

?>

Константи

Для визначення константи існує спеціальна функція define (). Її синтаксис такий:

define ( "Імя\_константи",

"Значення\_константи",

[Незалежність\_до\_регістру])

За замовчуванням імена констант чутливі до регістру. Для кожної константи це можна змінити, вказавши в якості значення аргументу Незалежність\_від\_регістру значення True. Існує угода, за якою 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 як параметр. приклади:

<? Php

// визначаємо константу PASSWORD

define ( "PASSWORD", "qwerty");

// визначаємо регістронезавісімого

// константу PI зі значенням 3.14

define ( "PI", "3.14", True);

echo (PASSWORD); // виведе значення константи PASSWORD,

// тобто qwerty

?>

Оператори

Оператори дозволяють виконувати різні дії із змінними, константами і виразами. Що таке вираз. 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 виробляти з виразами. Наприклад:

Таблиця 1 – арифметичні оператори

|  |  |  |
| --- | --- | --- |
| Позначення | Назва | Приклад |
| + | додавання | $ A + $ b |
| - | віднімання | $ A - $ b |
| \* | множення | $ A \* $ b |
| / | розподіл | $ A / $ b |
| % | Залишок від ділення | $ A% $ b |

Таблиця 2 – строкові оператори

|  |  |  |
| --- | --- | --- |
| Позначення | Назва | Приклад |
| . | конкатенація (Додавання рядків) | $ C = $ a. $ b  (Це рядок, що складається з $ a і $ b) |

Таблиця 3 – Оператори присвоювання

|  |  |  |  |
| --- | --- | --- | --- |
| Позначення | Назва | Приклад | Позначення |
| = | Присвоєння | змінної зліва від оператора буде присвоєно значення, отримане в результаті виконання будь-яких операцій або змінної / константи з правої сторони | $ A = ($ b = 4) +5;  ($ A буде дорівнює 9, $ b буде дорівнює 4) |
| + = |  | Скорочення. Додає до змінної число і потім привласнює їй отримане значення | $ A + = 5;  (Еквівалентно $ a = $ a + 5;) |
| . = |  | Скорочено позначає комбінацію операцій конкатенації і привласнення (спочатку додається рядок, потім отриманий рядок записується в змінну) | $ B = "Привіт";  $ B. = "Всім";  (Еквівалентно $ b = $ b. "Всім";)  В результаті: $ b = "Привіт всім" |

Таблиця 4 – Логічні оператори

|  |  |  |  |
| --- | --- | --- | --- |
| позначення | Назва | опис | приклад |
| and | І | $ a і $ b істинні (True) | $ A and $ b |
| && | І |  | $ A && $ b |
| or | або | Хоча б одна з змінних $ a або $ b істинна (можливо, що і обидві) | $ A or $ b |
| || | або |  | $ A || $ b |
| xor | Що виключає або | Одна з змінних істинна. Випадок, коли вони обидві істинні, виключається | $ A xor $ b |
| ! | Інверсія (NOT) | Якщо $ a = True, то! $ A = False і навпаки | ! $ a |

Таблиця 5 – Оператори порівняння

|  |  |  |  |
| --- | --- | --- | --- |
| позначення | Назва | приклад | опис |
| == | рівність | Значення змінних рівні | $ A == $ b |
| === | еквівалентність | Рівні значення і 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 чи рівно |  | $ A> = $ b |

Таблиця 6 – Оператори інкремента і декремента

|  |  |  |  |
| --- | --- | --- | --- |
| позначення | Назва | опис | приклад |
| ++ $ a | Пре-інкремент | Збільшує $ a на одиницю і повертає $ a | <?  $ A = 4;  echo "Повинно бути 4:". $ a ++;  echo "Повинно бути 6:". ++ $ a;  ?> |
| $ A ++ | Пост-інкремент | Повертає $ a, потім збільшує $ a на одиницю |  |
| - $ a | Пре-декремент | Зменшує $ a на одиницю і повертає $ a |  |
| $ a-- | Пост-декремент | Повертає $ a, потім зменшує $ a на одиницю |  |

Типи даних

PHP підтримує вісім простих типів даних.

Чотири скалярних типи:

* boolean (логічний);
* integer (цілий);
* float (з плаваючою точкою);
* string (строковий).

Два змішаних типи:

* array (масив);
* object (об'єкт).

І два спеціальних типи:

* resource (ресурс);
* NULL.

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.

Тип boolean (логічний або логічний тип)

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

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

<? Php

$ Test = True;

?>

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 випередити число 0 (нулем), для використання шістнадцятковій системи потрібно поставити перед числом 0x.

<? Php

$ A = 1234; // десяткове число

$ A = -123; // від'ємне число

$ A = 0123; // вісімкове число (еквівалентно

// 83 в десятковій системі)

$ A = 0x1A; // шістнадцяткове число (еквівалентно

// 26 в десятковій системі)

?>

Розмір цілого залежить від платформи, хоча, як правило, максимальне значення близько двох мільярдів (це 32-бітове знакове). 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.

Тип float (числа з плаваючою точкою)

Числа з плаваючою точкою (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 точкою залежить від платформи, хоча максимум, як правило, ~ 1.8e308 з точністю близько 14 десяткових цифр.

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 довжині.

Рядок в PHP може бути визначена трьома різними способами:

* за допомогою одинарних лапок;
* за допомогою подвійних лапок;
* heredoc-синтаксисом.

Одинарні лапки

Найпростіший спосіб визначити рядок - це укласти її в одинарні лапки « '». Щоб використовувати одинарні лапки всередині рядка, як і в багатьох інших мовах, перед нею необхідно поставити символ зворотної косої межі «\», т. Е. Екранувати її. Якщо зворотна коса риска повинна йти перед одинарною лапкою або бути в кінці рядка, необхідно продублювати її «\\ '».

Якщо всередині рядка, укладеної в одинарні лапки, зворотний 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 закриває лапками.

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

Подвійні лапки

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

Таблиця 7 – керуючі послідовності

|  |  |
| --- | --- |
| 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) |
| \\ | Зворотна коса риска |
| \ $ | знак долара |
| \ " | лапки |

Повторюємо, якщо ви захочете екранувати будь-який інший символ, зворотна коса риска також буде надрукована!

Найважливішим властивістю рядків у подвійних лапках є обробка змінних.

Heredoc

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

Heredoc-текст поводиться 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.

<? Php

$ Str = <<< EOD

Приклад рядка, що охоплює кілька

рядків, з використанням

heredoc-синтаксису

EOD;

?>

Тип array (масив)

Масив в PHP є впорядкованою карту - тип, який перетворює значення в ключі. Цей тип оптимізований в декількох напрямках, тому ви можете використовувати його як власне масив, список (вектор), хеш-таблицю (що є реалізацією карти), стек, чергу і т.д. Оскільки ви можете мати в якості значення інший масив PHP, можна також легко емулювати дерева.

Визначити масив можна за допомогою конструкції array () або безпосередньо задаючи значення його елементів.

Визначення за допомогою array ()

array ([key] => value,

[Key1] => value1, ...)

Мовна конструкція array () приймає в якості параметрів пари ключ => значення, розділені комами. Символ => встановлює відповідність між значенням і його ключем. Ключ може бути як цілим числом, так і рядком, а значення може бути будь-якого наявного в PHP типу. Числовий ключ масиву часто називають індексом. Індексування масиву в PHP починається з нуля. Значення елемента масиву можна отримати, вказавши після імені масиву в квадратних дужках ключ шуканого елемента. Якщо ключ масиву представляє собою стандартну запис цілого числа, то він розглядається як число, в іншому випадку - як рядок. Тому запис $ a [ "1"] рівносильна запису $ a [1], так само як і $ a [ "- 1"] рівносильно $ a [-1].

<? Php

$ Books = array ( "php" => "PHP users guide",

12 => true);

echo $ books [ "php"];

// виведе "PHP users guide"

echo $ books [12]; // виведе 1

?>

Якщо для елемента ключ не заданий, то в якості ключа береться максимальний числовий ключ, збільшений на одиницю. Якщо вказати ключ, якому вже було присвоєно якесь значення, то воно буде перезаписано. Починаючи з PHP 4.3.0, якщо максимальний ключ - негативне число, то наступним ключем масиву буде нуль (0).

Визначення за допомогою синтаксису квадратних дужок

Створити масив можна, просто записуючи в нього значення. Як ми вже говорили, значення елемента масиву можна отримати за допомогою квадратних дужок, усередині яких потрібно вказати його ключ наприклад, $ book [ "php"]. Якщо вказати новий ключ і нове значення наприклад, $ book [ "new\_key"] = "new\_value", то в масив додасться новий елемент. Якщо ми не вкажемо ключ, а тільки дамо значення $ book [] = "new\_value", то новий елемент масиву буде мати числовий ключ, на одиницю більший максимального існуючого. Якщо масив, в який ми додаємо значення, ще не існує, то він буде створений.

<?

$ Books [ "key"] = value; // додали в масив

// $ books значення

// value з ключем key

$ Books [] = value1; / \* Додали в масив

значення value1 з

ключем 13, оскільки

максимальний ключ у

нас був 12 \* /

?>

Для того щоб змінити конкретний елемент масиву, потрібно просто привласнити йому з його ключем нове значення. Змінити ключ елемента не можна, можна тільки видалити елемент (пару ключ / значення) і додати нову. Щоб видалити елемент масиву, потрібно використовувати функцію unset ().

Зауважимо, що, коли використовуються порожні квадратні дужки, максимальний числовий ключ шукається серед ключів, існуючих в масиві з моменту останнього переіндексування. Переіндексувати масив можна за допомогою функції array\_values ​​().

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.

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

<? Php

// створюємо клас людей

class Person

{

// метод, який навчає людину PHP

function know\_php ()

{

echo "Тепер я знаю PHP";

}

}

$ Bob = new Person; // створюємо об'єкт

// класу людина

$ Bob -> know\_php (); // навчаємо його PHP

?>

Більш докладно реалізація принципів ООП в мові PHP буде розглянута в одній з наступних лекцій.

Тип resource (ресурси)

Ресурс - це спеціальна змінна, що містить посилання на зовнішній ресурс (наприклад, з'єднання з базою даних). Ресурси створюються і використовуються спеціальними функціями (наприклад, mysql\_connect (), pdf\_new () і т.п.).

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 якесь значення;
* вона була видалена за допомогою unset ().

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

Керуючі конструкції

Умовні оператори

Оператор if .. else

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

if (вираз) блок\_виконання

Тут вираз є будь-який правильний PHP-вираз (тобто все, що має значення). В 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. У FALSE перетворяться наступні значення:

* логічне False
* цілий нуль (0)
* дійсний нуль (0.0)
* порожній рядок і рядок "0"
* масив без елементів
* об'єкт без змінних (докладно про об'єкти буде розказано в одній з наступних лекцій)
* спеціальний тип NULL

1. Всі інші значення перетворяться в TRUE.

<?

$ Names = array ( "Іван", "Петро", "Семен");

if ($ names [0] == "Іван") {

echo "Привіт, Ваня!";

$ Num = 1;

$ Account = 2000;

}

if ($ num) echo "Іван перший в списку!";

$ Bax = 30;

if ($ account> 100 \* $ bax + 3)

echo "Цей рядок чи не з'явиться

на екрані, так як умова не виконана ";

?>

. Оператор else розширює if на випадок, якщо перевіряється в if вираз є невірним, і дозволяє виконати будь-які дії при таких умовах.

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

if (вираз) блок\_виконання

else блок\_виконання1

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 в 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 (вираження1) блок\_виконання1

...

else блок\_виполненіяN

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.

***Альтернативний синтаксис***

PHP пропонує альтернативний синтаксис для деяких своїх керуючих структур, а саме для if, while, for, foreach і switch. У кожному разі відкриває дужку потрібно замінити на двокрапку (:), а закриває - 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, записана в круглих дужках оператора if, виявилося істиною, буде виконуватися весь код, від двокрапки «:» до команди endif ;. Використання такого синтаксису корисно при встановленні php в html-код.

<? Php

$ Names = array ( "Іван", "Петро", "Семен");

if ($ names [0] == "Іван"):

?>

Привіт, Ваня!

<? Php endif?>

Оператор switch

Ще одна конструкція, що дозволяє перевіряти умову і виконувати в залежності від цього різні дії, - це switch. На російську мову назва даного оператора можна перевести як «перемикач». І сенс у нього саме такий. Залежно від того, яке значення має змінна, він переключається між різними блоками дії. switch дуже схожий на оператор if ... elseif ... else або набір операторів if. Структуру switch можна записати в такий спосіб:

switch (вираз або змінна) {

case значення1:

блок\_дій1

break;

case значення2:

блок\_действій2

break;

...

default:

блок\_действій\_по\_умолчанію

}

На відміну від if, тут значення виразу не наводиться до логічного типу, а просто порівнюється зі значеннями, перерахованими після ключових слів case (значення1, значення 2 і т.д.). Якщо значення виразу співпало з якимсь варіантом, то виконується відповідний блок\_дій - від двокрапки після співпала значення до кінця switch або до першого оператора break, якщо такий знайдеться. Якщо значення виразу не співпало з одним з варіантів, то виконуються дії за замовчуванням (блок\_действій\_по\_умолчанію), що знаходяться після ключового слова default. Вираз в switch обчислюється тільки один раз, а в операторі elseif - кожен раз, тому, якщо вираз досить складне, то switch працює швидше.

Якщо опустити оператор break, то, інтерпретатор буде продовжувати виконання інструкцій поки не зустріне break або не дійде до кінця блоку.

Цикли

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

**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 вираження 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 зупинений до кінця ітерації (тобто поки всі команди блоку\_виконання НЕ будуть виконані).

**Do ... while**

Цикли do..while дуже схожі на цикли while, з тією лише різницею, що істинність виразу перевіряється в кінці циклу, а не на початку. Завдяки цьому блок\_виконання цикл do ... while гарантовано виконується хоча б один раз.

структура:

do {блок\_виконання} while (вираз);

For

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

структура:

for (вираження1; вираженіе2; вираз3) {блок\_виконання}

або

for (вираження1; вираженіе2; вираз3): блок\_виконання endfor;

Тут, як ми бачимо, умова складається відразу з трьох виразів. Перший вираз вираження1 обчислюється безумовно один раз на початку циклу. На початку кожної ітерації обчислюється вираженіе2. Якщо воно є True, то цикл триває і виконуються всі команди блоку\_виконання. Якщо вираженіе2 обчислюється як False, то виконання циклу зупиняється. В кінці кожної ітерації (тобто після виконання всіх команд блоку\_виконання) обчислюється вираз3.

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

}

?>

Кожне з виразів 1, 2, 3 може бути порожнім. Якщо вираженіе2 є порожнім, то це означає, що цикл повинен виконуватися невизначений час (в цьому випадку PHP вважає це вираз завжди істинним). Це не так марно, як здається, адже цикл можна зупиняти, використовуючи оператор break.

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

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

<? Php

$ I = 0; // 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 третю 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 \* /

?>

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 раз, 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 блоке\_виполненія.

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

<? Php

$ Names = array ( "Іван", "Петро", "Семен");

foreach ($ names as $ val) {

echo "Привіт, $ val <br>";

// виведе всім вітання

}

foreach ($ names as $ k => $ val) {

// окрім привітання,

// виведемо номера в списку, тобто ключі

echo "Привіт, $ val!

Ти в списку під номером $ k <br> ";

}

?>

Оператори передачі керування

Іноді в разі особливих обставин потрібно негайно завершити роботу циклу і передати управління першої інструкції програми, розташованої за останньою інструкцією циклу. Для цього використовують оператори break і continue.

Break

Оператор break закінчує виконання поточного циклу, будь то for, foreach, while, do..while або switch. break може використовуватися з числовим аргументом, який говорить, роботу скількох керуючих структур, що містять його, потрібно завершити.

Якщо після оператора break вказати число, то перерветься саме таку кількість містять цей оператор циклів.

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 містять його керуючих конструкцій повинні завершити роботу.

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

Оператори включення

**Include**

Оператор include дозволяє включати код, що міститься в зазначеному 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");

Зауважимо, що використання оператора include еквівалентно простій вставці змістовної частини файлу params.inc в код програми include.php. АЛЕ! в момент вставки файлу відбувається перемикання з режиму обробки PHP в режим HTML. Тому код всередині включається файлу, 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 проводиться в include\_path щодо директорії поточного скрипта.
3. Параметр include\_path, який 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 основній файл include.php ви включаєте файл my\_dir / a.php, який в 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 за 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 зухвалій файлі з цього рядка і далі. Відповідно, якщо include з'являється всередині функції файла, який викликає, то код, що міститься в викликається файлі, буде вести себе так, як ніби він був визначений всередині функції. Таким чином, він успадкує область видимості цієї функції.

Крім локальних файлів, за допомогою include можна включати і зовнішні файли, вказуючи їх url-адреси. Дана можливість контролюється директивою url\_fopen\_wrappers в файлі налаштувань PHP і за замовчуванням, як правило, включена. Але в версіях PHP для Windows до PHP 4.3.0 ця можливість не підтримується зовсім, незалежно від url\_fopen\_wrappers.

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 виконувати будь-якої файл. Основна відмінність require і include полягає в тому, як вони реагують на виникнення помилки. Як вже говорилося, include видає попередження, і робота скрипта продовжується. 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 ні один рядок коду з файлу, що вставляється 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 вставляється файл.

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

Процедура розгляду запитів за допомогою PHP

Форма запиту клієнта

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

Простий запит містить метод доступу і адреса 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 поширених з них ми поговоримо трохи пізніше. Як 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-адресу запитуваного ресурсу.

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

<Повний запит>: = <Рядок Стану>

(<Загальний заголовок> | <Заголовок запиту> |

<Тема 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 і 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

В 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.

У скороченій формі опускають протокол і адресу сервера, вказуючи тільки 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.

Далі ми розглянемо найбільш поширені методи відправлення запитів.

Методи

Як вже говорилося, будь-який запит клієнта до сервера повинен починатися з вказівки методу. 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 (в рядок статусу). При використанні методу GET в поле тіла ресурсу повертається власне викликана інформація (текст HTML-документа, наприклад).

Існує різновид методу GET - умовний GET. Цей метод повідомляє серверу про те, що на запит потрібно відповісти, тільки якщо виконано умову, що міститься в поле 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, новинні і поштові повідомлення, дані для додавання в базу даних, тобто для передачі інформації великого обсягу і досить важливою. На відміну від методів GET і HEAD, в POST передається тіло ресурсу, яке і є інформацією, одержуваної з полів форм або інших джерел введення.

Використання HTML-форм для передачі даних на сервер

Як передавати дані сервера? 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 відповідні варіанти зі списку. Дані, записані в 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.

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

<html>

<h2> Форма для реєстрації учасників </h2>

<form action = "1.php" method = POST> <! - створюємо форму ->

<! - дані форми буде обробляти файл 1.php, при

відправці запиту буде використаний метод POST ->

Ім'я <br> <input type = text name = "first\_name"

value = "Введіть Ваше ім'я"> <br>

Прізвище <br> <input type = text name = "last\_name"> </br>

E-mail <br> <input type = text name = "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> </textarea>

<p> <input name = "confirm" type = checkbox

checked> Підтвердити отримання <br/>

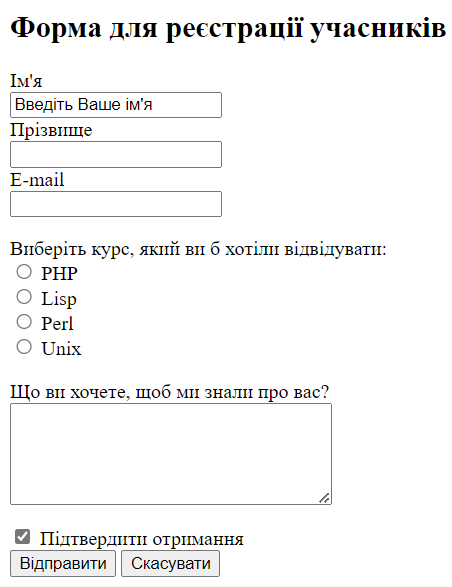
<input type = submit value = "Відправити">

<input type = reset value = "Скасувати">

</form>

</html>

Після обробки браузером цей файл буде виглядати приблизно так:



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

Для методу GET

При відправці даних форми за допомогою методу GET вміст форми додається до URL після знака питання у вигляді пар ім'я = значення, об'єднаних за допомогою амперсанта &:

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

Тут action - це URL-адресу програми, яка повинна обробляти форму (це або програма, задана в атрибуті action тега form, або сама поточна програма, якщо цей атрибут опущений). Імена name1, name2, name3 відповідають іменам елементів форми, а value1, value2, value3 - значенням цих елементів. Всі спеціальні символи, включаючи = і &, в іменах або значеннях цих параметрів будуть опущені. Тому не варто використовувати в назвах або значних елементів форми ці символи і символи кирилиці в ідентифікаторах.

Якщо в поле для введення ввести який-небудь службовий символ, то він буде переданий в його шестнадцатеричном 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 radio button значення value визначається атрибутом VALUE у тому випадку, коли кнопка відзначена. Не відзначені кнопки при складанні рядка запиту ігноруються повністю. Кілька кнопок типу checkbox можуть мати один атрибут NAME (і різні VALUE), якщо це необхідно. Кнопки типу radio button призначені для одного з усіх запропонованих варіантів і тому повинні мати однаковий атрибут NAME і різні атрибути VALUE.

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

<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 заборонено змінювати користувачеві.

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

Для методу POST

Вміст форми кодується точно так же, як для методу GET (див. Вище), але замість додавання рядка до URL вміст запиту посилається 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 ​​сервера за допомогою методу POST, подається на стандартне введення програмі, зазначеної в атрибуті action, або поточному скрипту, якщо цей атрибут опущений. Довжина посилається файлу передається в 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 в GET. Відповідно і змінити значення параметрів можна, тільки змінивши значення, введене в форму. При використанні POST користувач не бачить передані серверу дані.

Основна перевага POST запитів - це їхня велика безпеку і функціональність у порівнянні з GET-запитами. Тому метод POST частіше використовують для передачі важливої ​​інформації, а також інформації великого обсягу. Проте 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 змінних, званих змінними оточення, 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

Процедура розгляду REQUST PHP

До сих 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 кілька способів отримання доступу до даних, переданих клієнтом по протоколу HTTP.

До версії PHP 4.1.0 доступ до таких даних здійснювався по іменах переданих змінних (нагадаємо, що дані передаються у вигляді пар «ім'я змінної, символ« = », значення змінної»). Таким чином, якщо, наприклад, було передано first\_name = Nina, то всередині скрипта з'являлася змінна $ first\_name зі значенням Nina. Якщо потрібно розрізняти, яким методом були передані дані, то використовувалися асоціативні масиви $ HTTP\_POST\_VARS і $ HTTP\_GET\_VARS, ключами яких були імена переданих змінних, а значеннями - відповідно значення цих змінних. Таким чином, якщо пара first\_name = Nina передана методом GET, то $ HTTP\_GET\_VARS [ "first\_name"] = "Nina".

Використовувати в програмі імена переданих змінних безпосередньо небезпечно. Тому було вирішено починаючи з PHP 4.1.0 задіяти для звернення до змінних, переданим за допомогою HTTP-запитів, 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 школи програмування, як у наведеному вище прикладі. Тоді в файлі 1.php, що обробляє цю форму, можна написати наступне:

<? Php

$ Str = "Привіт,

". $ \_ REQUEST [" first\_name "]."

". $ \_ REQUEST [" last\_name "]."! <br> ";

$ Str. = "Ви вибрали для вивчення курс по

". $ \_ REQUEST [" kurs "];

echo $ str;

?>

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

Привіт, Вася Петров!

Ви вибрали 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 суперглобального, 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 = "Привіт,

". $ \_ POST [" first\_name "]."

". $ \_ POST [" last\_name "]."! <br> ";

$ Str. = "Ви вибрали 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 в попередньому прикладі:

Привіт, Вася Петров!

Ви вибрали для вивчення курс по PHP

Для того щоб зберегти можливість обробки скриптів більш ранніх версій, ніж PHP 4.1.0, була введена директива register\_globals, дозволяє або забороняє доступ до змінних безпосередньо за їхніми іменами. Якщо у файлі налаштувань PHP параметр register\_globals = On, то до змінних, переданим сервера методами GET і POST, можна звертатися просто по їхніх іменах (тобто можна писати $ first\_name). Якщо ж register\_globals = Off, то потрібно писати $ \_REQUEST [ "first\_name"] або $ \_POST [ "first\_name"], $ \_GET [ "first\_name"], $ HTTP\_POST\_VARS [ "first\_name"], $ HTTP\_GET\_VARS [ "first\_name"]. З точки зору безпеки цю директиву краще відключати (тобто register\_globals = Off). При включеній директиві register\_globals перераховані вище масиви також будуть містити дані, передані клієнтом.

Для можливості отримання декількох значень з форми під одним ім'ям (як масиву) елемент форми повинен мати атрибут name в вигляді name = "name\_of\_input []". Наприклад якщо в елементи форми:

<INPUT type = "text" name = "chisla []">

<INPUT type = "text" name = "chisla []">

<INPUT type = "text" name = "chisla []">

ввести значення chislo1, chislo2, chislo3, то при обробці запиту значення $ \_REQUEST [ 'chisla'] (а точніше $ \_GET [ 'chisla'] або $ \_POST [ 'chisla']) буде масивом виду

$ Chisla = array (0 => "chislo1",

1 => "chislo2",

2 => "chislo3");

Можна також вказувати імена ключів майбутнього асоціативного масиву, наприклад:

<INPUT type = "text" name = "chisla [one]">

<INPUT type = "text" name = "chisla [two]">

<INPUT type = "text" name = "chisla [three]">

При тому ж запиті значення $ \_REQUEST [ 'chisla'] буде масивом виду

$ Chisla = array ( "one" => "chislo1",

"Two" => "chislo2",

"Three" => "chislo3");

Іноді виникає необхідність дізнатися значення будь-якої змінної оточення, наприклад метод, який використовували 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 передане їй як параметр.

<?

getenv ( "REQUEST\_METHOD");

// поверне використаний метод

echo getenv ( "REMOTE\_ADDR");

// виведе IP-адреса користувача,

// послав запит

?>

Як ми вже говорили, якщо використовується метод GET, то дані передаються додаванням рядка запиту у вигляді пар «ім'я\_змінної = значення до URL-адресою ресурсу». Все, що записано в URL після знака питання, можна отримати за допомогою команди

getenv ( "QUERY\_STRING");

Завдяки цьому можна по методу GET передавати дані в якому-небудь іншому вигляді. Наприклад, вказувати тільки значення декількох параметрів через знак плюс, а в скрипті розбирати рядок запиту на частини або можна передавати значення всього одного параметра. В цьому випадку в масиві $ \_GET з'явиться порожній елемент з ключем, рівним цьому значенню (всієї рядку 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 сервера. Щоб їх побачити, хакер повинен підмінити нашу форму своєї. Тоді сервер відправить результати обробки неправильної форми не туди, куди потрібно. Щоб цього уникнути, можна перевіряти адресу сторінки, з якої були послані дані. Це можна зробити знову ж за допомогою функції getenv ():

getenv ( "HTTP\_REFERER");

*Приклад обробки запиту за допомогою PHP*

Припустимо, потрібно написати форму для реєстрації учасників школи програмування і після реєстрації відправити учаснику повідомлення. Ми не будемо відправляти що-небудь по електронній пошті, а просто сгенерируем це повідомлення і виведемо його на екран браузера. Початковий варіант форми реєстрації ми вже наводили вище. Змінимо його таким чином, щоб кожен реєструється міг вибрати скільки завгодно курсів для відвідування, і не будемо підтверджувати отримання реєстраційної форми.

<H2> Форма для реєстрації студентів </ h2>

<Form action = "1.php" method = POST>

Ім'я <br> <input type = text name = "first\_name"

value = "Введіть Ваше ім'я"> <br>

Прізвище <br> <input type = text name = "last\_name"> <br>

E-mail <br> <input type = text name = "email"> <br>

<P> Виберіть курс, який ви б хотіли відвідувати: <br>

<Input type = checkbox name = 'kurs []' value = 'PHP'> PHP <br>

<Input type = checkbox name = 'kurs []' value = 'Lisp'> Lisp <br>

<Input type = checkbox name = 'kurs []' value = 'Perl'> Perl <br>

<Input type = checkbox name = 'kurs []' value = 'Unix'> Unix <br>

<P> Що ви хочете, щоб ми знали про вас? <BR>

<Textarea name = "comment" cols = 32 rows = 5> </ textarea>

<Input type = submit value = "Відправити">

<Input type = reset value = "Скасувати">

</ Form>

**Лістинг 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 другій елемент масиву і т.д. Можна, звичайно, просто дати різні імена елементам 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 зареєстрованого людини, скрипт генерує відповідне повідомлення. Якщо людина вибрала якісь курси, то йому виводиться повідомлення про час їх проведення і про лекторів, які їх читають. Якщо людина нічого не вибрав, то виводиться повідомлення про наступних зборах заочної школи програмістів (ЗШП).

<?

// створимо масиви відповідностей курс-час його

// проведення і 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 = "Вітаю Вас, шановний". $ \_POST [ "first\_name"]

. "". $ \_POST [ "last\_name"]. "! <br>";

$ Str. = "<br> Повідомляємо Вам, що";

$ Kurses = $ \_POST [ "kurs"]; // збережемо в цієї змінної

// список обраних курсів

if (! isset ($ kurses)) {// якщо ви не вкажете один курс

$ Event = "наступні збори студентів";

$ Str. = "$ Event відбудеться $ date". MEETING\_TIME. "<br>";

} Else {// якщо хоча б один курс обраний

$ Event = "вибрані Вами лекції відбудуться $ date <ul>";

// функція count обчислює число елементів в масиві

for ($ i = 0; $ i <count ($ kurses); $ i ++) {

// для кожного обраного курсу

$ K = $ kurses [$ i]; // запам'ятовуємо назву курсу

$ Lect = $ lect. "<Li> лекція по $ 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

?>

**Скрипт 1.php, що обробляє форму form\_final.html**

Таким чином ми познайомилися з принципами обробки запитів користувачів в PHP.

Виконання роботи

Опис завдань, що вирішуються серверним сценарієм.

1. Отримання даних з форми:

* Форма, створена на HTML, дозволяє користувачеві вводити два числа та обирати математичну операцію.
* Після введення даних користувачем форма відправляє ці дані на сервер за допомогою методу POST.

1. Перевірка коректності даних:

* Сценарій перевіряє, чи введені дані є числовими значеннями. Якщо одне або обидва введених значення не є числами, сценарій виводить повідомлення про помилку.
* Також передбачена перевірка, чи обрано користувачем операцію для обчислення.

1. Виконання математичних операцій:

Залежно від вибраної операції, сценарій виконує одну з доступних математичних операцій:

* Додавання
* Віднімання
* Множення
* Ділення (з перевіркою на ділення на нуль)
* Залишок від ділення (з перевіркою на ділення на нуль)
* Зведення в ступінь

Результат обчислення зберігається в змінній $result.

1. Форматування та вивід результату:

* Сценарій формує HTML-сторінку з результатом обчислення, яка включає заголовок, обчислений результат та інформацію про дату і час генерації звіту.
* Також внизу сторінки автоматично додається футер з датою, часом генерації документа і ПІБ виконавця.

1. Відображення повідомлень про помилки:

* Якщо користувач не ввів коректні дані або не обрав операцію, сценарій формує відповідне повідомлення про помилку і відображає його замість результату.

Короткий опис алгоритму роботи сценарію.

1. Сценарій перевіряє, чи всі обов'язкові дані (number1, number2, operation) передані методом POST. Якщо будь-яке з цих полів відсутнє, користувачу виводиться повідомлення про помилку.
2. Перевірка коректності числових значень: Якщо дані присутні, сценарій перевіряє, чи є обидва значення (number1і number2) числовими. Якщо введені некоректні значення, користувачу відображається повідомлення про помилку.
3. Виконання математичної операції:

* На основі вибраної операції (operation) виконується обчислення за допомогою операцій +, -, \*, /, або зведення в ступінь.
* Якщо обрано ділення і number2 дорівнює нулю, виводиться повідомлення про помилку ділення на нуль.

1. Виведення результату: Після виконання обчислення сценарій виводить результат з відповідним заголовком. Якщо виникла помилка, повідомлення про помилку заміняє результат.
2. Футер з датою генерації та автором: В кінці сторінки відображається футер із поточною датою і часом генерації звіту, а також з ім'ям виконавця.

Опис використовуваних вхідних даних і функцій

**Вхідні дані**

Сценарій обробляє три основні типи вхідних даних, які користувач вводить у форму:

1. *number1*: Перше числове значення, яке вводить користувач. Це число використовується як один з операндів для виконання математичних операцій.
2. *number2*: Друге числове значення, яке вводить користувач. Це число є другим оперантом у математичних обчисленнях.
3. *operation*: Вибір математичної операції, яку користувач хоче виконати. Це значення визначає, яка саме операція (додавання, віднімання, множення, ділення, зведення в ступінь) буде застосована до введених чисел.

**Функції**

Сценарій використовує наступні вбудовані функції PHP для обробки даних:

1. isset():

* Перевіряє, чи були передані певні змінні. У сценарії використовується для перевірки наявності вхідних даних (number1, number2, operation) у масиві $\_POST.

1. is\_numeric():

* Використовується для перевірки, чи є введені значення числовими. Це гарантує, що користувач ввів коректні дані для подальших обчислень.

1. (float):

* Явно перетворює значення на тип float. Це дозволяє виконувати математичні операції з точністю до десяткових дробів.

1. switch:

* Структура управління, що дозволяє виконувати різні дії в залежності від значення змінної operation. Залежно від обраної операції виконується відповідна математична дія.

1. pow():

* Вбудована функція, що використовується для зведення числа в ступінь. У даному сценарії вона застосовується для операції exponentiation.

1. echo:

* Використовується для виведення результатів на екран, включаючи текст повідомлень і результат обчислення.

1. date():

* Використовується для отримання поточної дати і часу у форматі H:i:s d.m.Y, що відображається у футері.

Ці вхідні дані і функції забезпечують ефективну обробку користувацьких запитів і виконання математичних операцій, зокрема, гарантують коректність введених даних і надають користувачу результат у зручному для сприйняття форматі.

Лістинг вихідного коду з коментарями

Код сценарію, що обробляє форму калькулятора подано в Додатку А.

Результати роботи

Результати роботи сценарію подано в таблиці 8.

Таблиця 8 – результат роботи сценарію з різними вхідними данними

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Вхідне значення 1** | **Вхідне значення 2** | **Операція** | **Очікуваний результат** | **Обробка помилок** |
| 5 | 3 | Додавання | 8 | - |
| 5 | 3 | Віднімання | 2 | - |
| 5 | 3 | Множення | 15 | - |
| 5 | 3 | Ділення | 1.67 | - |
| 5 | 0 | Ділення | Помилка: ділення на нуль | Перевірка на нуль перед діленням |
| 5 | -3 | Додавання | 2 | - |
| -5 | 3 | Додавання | -2 | - |
| -5 | -3 | Додавання | -8 | - |
| 5 | 3 | Піднесення до ступеня | 125 | - |
| 5 | 0 | Піднесення до ступеня | 1 | - |
| 5 | -1 | Піднесення до ступеня | 0.2 | - |
| 0 | 3 | Додавання | 3 | - |
| 0 | 0 | Додавання | 0 | - |
| 0 | 0 | Ділення | Помилка: ділення на нуль | Перевірка на нуль перед діленням |
| abc' | 3 | Додавання | Помилка: введіть коректні числові значення | Валідація вхідних даних |
| 5 | abc' | Віднімання | Помилка: введіть коректні числові значення | Валідація вхідних даних |
| null | 3 | Додавання | Помилка: відсутні обов'язкові дані | Перевірка наявності даних |

висновок

В рамках виконання роботи було поставлено завдання розробити серверний сценарій на мові PHP, який обробляє запити користувачів і представляє результати у вигляді документа HTML. Основною метою було вивчення синтаксису мови PHP, ознайомлення з типами даних, операторами, які керують конструкціями, а також отримання практичних навичок у складанні серверних скриптів.

У процесі виконання роботи було:

* Вивчено синтаксис мови PHP, що дозволило зрозуміти основи програмування на цій мові.
* Ознайомлено з різними типами даних у PHP, такими як числові, рядкові, масиви та об'єкти, що допомогло правильно обробляти вхідні дані.
* Створено HTML-форму, що дозволяє користувачеві вводити два числові значення та вибирати математичну операцію для обчислення.
* Реалізовано обробку запитів користувачів, що включає перевірку введених даних на коректність, а також обробку помилок.
* Написано логіку для виконання елементарних математичних дій, таких як додавання, віднімання, множення, ділення та піднесення до ступеня.
* Виведено результати обчислень у зрозумілому та зручному для користувача вигляді, з зазначенням дати генерації документа та ПІБ виконавця.

В результаті виконання роботи було створено функціональний "Міні-калькулятор", який успішно обробляє запити користувачів та демонструє навички, отримані під час навчання. Робота дозволила поглибити знання з PHP та набути досвіду в розробці серверних скриптів.

*Додаток А*Машинний лістинг файлу **calculator.php**

<!DOCTYPE html>

<html lang="uk">

<head> <meta charset="UTF-8">

<title>Результат обчислення за допомогою калькулятора</title>

</head>

<body>

<?php

// Встановлюємо часовий пояс

date\_default\_timezone\_set('Europe/Kiev');

// Перевірка наявності всіх необхідних даних

if (isset($\_POST['number1'], $\_POST['number2'], $\_POST['operation'])) {

$number1 = $\_POST['number1']; // Перше число

$number2 = $\_POST['number2']; // Друге число

$operation = $\_POST['operation']; // Вибрана операція

// Перевірка коректності введених чисел

if (is\_numeric($number1) && is\_numeric($number2)) {

$number1 = (float)$number1; // Приведення до типу float

$number2 = (float)$number2; // Приведення до типу float

$result = ''; // Змінна для зберігання результату

// Виконання обраної операції

switch ($operation) {

case 'addition':

$result = $number1 + $number2; // Додавання

break;

case 'subtraction':

$result = $number1 - $number2; // Віднімання

break;

case 'multiplication':

$result = $number1 \* $number2; // Множення

break;

case 'division':

// Перевірка на ділення на нуль

if ($number2 != 0) {

$result = $number1 / $number2; // Ділення

} else {

$result = "Помилка: ділення на нуль."; // Помилка

}

break;

case 'exponentiation':

$result = pow($number1, $number2); // Зведення в ступінь

break;

default:

$result = "Невідома операція."; // Невідома операція

}

// Виведення результату

echo "<h2>Результат обчислення:</h2>";

echo "<u>Результат: $result</u>";

} else {

echo "Помилка: введіть коректні числові значення."; // Помилка валидації

}

} else {

echo "Помилка: відсутні обов'язкові дані."; // Відсутні дані

}

?>

<footer>

<hr>

<p>Дата генерації: <?php echo date('H:i:s d.m.Y'); ?></br>

Виконавець: Зайченко Ярослав Ігорович, 632п</p>

</footer>

</body>

</html>

*Додаток Б*Машинний лістинг файлу **miniCalc\_mainform.html**

<!DOCTYPE html>

<html lang="uk">

<head>

<meta charset="UTF-8">

<title>Міні-калькулятор</title>

</head>

<body>

<h2>Міні-калькулятор</h2>

<form action="calculator.php" method="POST">

<label>Число 1:</label><br>

<input type="text" name="number1" required><br><br>

<label>Число 2:</label><br>

<input type="text" name="number2" required><br><br>

<label>Оберіть операцію:</label><br>

<input type="radio" name="operation" value="addition" required> Додавання<br>

<input type="radio" name="operation" value="subtraction"> Віднімання<br>

<input type="radio" name="operation" value="multiplication"> Множення<br>

<input type="radio" name="operation" value="division"> Ділення<br>

<input type="radio" name="operation" value="exponentiation"> Піднесення до степеня<br><br>

<input type="submit" value="Обчислити">

<input type="reset" value="Скасувати">

</form>

</body>

</html>