МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний аерокосмічний університет ім. М. Є. Жуковського

«Харківський авіаційний інститут»

факультет програмної інженерії та бізнесу

кафедра інженерії програмного забезпечення

**Практична робота № 4**

з дисципліни « Web-програмування мовою PHP »

*назва дисципліни*

на тему: «ВИКОРИСТАННЯ РЕГУЛЯРНИХ ВИРАЗІВ. ЗАСТОСУВАННЯ HTTP COOKIES І МЕХАНІЗМУ СЕСІЙ»

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 можливостей обробки текстових даних із застосуванням регулярних виразів, а також способів передачі даних між скриптами за допомогою HTTP cookies та механізму сесій.

Постановка завдання

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

Завдання варіанту 3:

"Гостьова книга"

Видалити всі теги розмітки HTML з тіла повідомлення. Перевірити адресу електронної пошти на коректність за допомогою регулярного виразу. У разі неправильної адреси виводити попередження (повідомлення до БД не вносити, а заново виводити у формі для додавання). Після 3 невдалих спроб введення (для рахунку спроб використовувати сесії) встановити значення cookie, за наявності якого блокується форма для додавання (або як варіант - виводиться попередження про перевищення ліміту спроб введення).

Порядок виконання роботи

1. Отримати у викладача індивідуальне завдання для виконання роботи.
2. Вивчити теоретичний матеріал.
3. Визначити завдання, які має вирішувати сценарій, що розробляється.
4. Розробити текстовий інтерфейс користувача відповідно до завдання, а також кінцевий вид документа, що генерується, з результатами роботи сценарію.
5. Реалізувати мовою PHP спроектований сценарій.
6. Протестувати локально розроблений сценарій.
7. Зробити висновки щодо роботи.

Зміст звіту

1. Постановка задачі. Опис завдань, які вирішуються серверним сценарієм.
2. Короткий опис алгоритму сценарію.
3. Опис використовуваних вхідних даних та функцій.
4. Лістинг вихідного коду сценарію з коментарями, а також результати його роботи (скриншот або текстове подання).
5. Висновки щодо роботи.

Теоретичні відомомості

Регулярний вираз це патерн, який порівнюється з рядком-суб'єктом зліва направо. Більшість символів у патерні представляють себе і збігаються з відповідними символами в рядку-суб'єкті. Як тривіальний приклад - патерн The quick brown fox збігається з частиною рядка-суб'єкта, який повністю ідентичний йому.

Метасимволи

Потужність регулярним виразам надає можливість включати до патерну альтернативи та повторення. Вони кодуються в патерне метасимволами, які не уявляють себе, а інтерпретуються особливим чином.

Є два різних набори метасимволів: патерна, що розпізнаються в будь- якому місці, крім квадратних дужок, і ті, які розпізнаються в квадратних дужках.

* \ - загальний escape-символ, використовується для екранування спеціальних символів.
* ^ - позначає початок виразу або заперечення класу символів (якщо використовується всередині []).
* $ - позначає кінець виразу.
* . - збігається з будь-яким символом, окрім символу нового рядка (за замовчуванням).
* [ - початок визначення класу символів.
* ] - кінець визначення класу символів.
* | - позначає альтернативу (логічне "або").
* ( - початок субпатерну (групування виразу).
* ) - кінець субпатерну.
* ? - квантифікатор "0 або 1" або мінімізатор інших квантифікаторів.
* - квантифікатор "0 або більше".
* + - квантифікатор "1 або більше".
* { - початок визначення min/max кількості повторів у квантифікаторі.
* } - кінець визначення min/max кількості повторів у квантифікаторі.
  + - у класі символів позначає діапазон символів (наприклад, [a-z]), але якщо використовується першим чи останнім символом у класі, сприймається як звичайний символ.

Backslash/зворотний слеш

Символ backslash використовується по-різному. По-перше, якщо після нього йде неалфавітний символ, він скасовує будь-яке спеціальне значення, яке може мати символ. Таке використання зворотного слеша як escape-символу застосовується як усередині, і поза класів символів.

Наприклад, якщо ви хочете знайти збіг із символом "\*", ви записуєте в патерні "\\*". Це буде працювати незалежно від того, чи може наступ ний символ інтерпретуватися як метасимвол, тому завжди надійніше записувати неалфавітний символ з "\", щоб специфікувати, що він уяв ляє себе. Особливо якщо ви хочете знайти збіг з backslash тоді ви записуєте "\\".

Якщо патерн компілюється із опцієюPCRE\_EXTENDED, то пробіли в патерні (крім пробілів у класі символів) і символи між "#" поза класом символів і наступним символом newline ігноруються.

По-друге, backslash надає спосіб кодування в патерні недрукованих символів видимим чином. Обмежень на появу недрукованих символів немає, за винятком двійкового нуля, який закінчує патерн, але якщо патерн готується шляхом редагування тексту, то зазвичай легше вико ристовувати одну з наступних escape-послідовностей (замін), а не бінарний символ.

* \a - alarm/попередження, тобто символ BEL (hex 07).
* \cx - "Control-x", де x — це будь-який символ.
* \e - Escape (hex 1B).
* \f - Form feed/переведення сторінки (hex 0C).
* \n - Newline/новий рядок (hex 0A).
* \r - Carriage return/повернення каретки (hex 0D).
* \t - Tab/табуляція (hex 09).
* \xhh - Символ з 16-річним кодом `hh`.
* \ddd - Символ з 8-річним кодом `ddd`, або backreference/зворотне посилання.

Ефект від застосування \cx такий: якщо "x" це символ у нижньому регістрі, він конвертується у верхній регістр. Потім біт символу 6 (hex 40) інвертується. Таким чином, \cz стає hex 1A, \c{ стає hex 3B, а \c; ставати hex 7B.

Після "x" читаються не більше двох 16-річних цифр (букви можуть бути в будь-якому регістрі).

Після "\0" читаються не більше чотирьох 8-річних цифр. В обох випадках, якщо є менше двох цифр, використовуються ті, які представлені. Таким чином, послідовність "\0\x\07" специфікує два бінарних нулі з наступним символом BEL. Переконайтеся, що ви надали дві цифри після початкового нуля, якщо наступний символ є 8-річним числом.

Обробка backslash з наступними цифрами, відмінними від 0, складніша. Поза класом символів, PCRE читає його та будь-які наступні символи як 10-річне число. Якщо число менше 10 або якщо у виразі є щонайменше стільки захоплюючих лівих дужок, вся послідовність вважається back reference\зворотним посиланням. Опис того, як цей механізм працює, буде дано пізніше у дискусії про субпатерни у дужках.

Усередині класу символів, або якщо 10-річне число більше 9 і немає такої кількості захоплюючих субпатернів, PCRE зчитує до трьох 8-річних цифр, що йдуть після backslash, і генерує один байт з останніх значних 8 бітів цього значення. Будь-які наступні цифри представляють себе. Наприклад:

* \040 - інший спосіб запису символу пробілу (space).
* \40 - те саме, за умови, що є менше 40 попередніх захоплюючих субпатернів.
* \7 - це завжди back reference/зворотне посилання.
* \11 - може бути back reference або інший спосіб запису символу табуляції (tab).
* \011 - це завжди символ табуляції (tab).
* \0113 - це символ табуляції (tab) з наступним символом "3".
* \113 - це символ з 8-річним кодом 113 (оскільки не може бути більше 99 зворотних посилань).
* \377 - це байт, що складається з бітових "1".
* \81 - це back reference або бінарний нуль з наступними двома символами "8" та "1".

Зауважте, що 8-річні значення 100 або більше повинні не вводитися з провідним 0, оскільки читається не більше трьох 8-річних цифр.

Усі послідовності, що визначають однобайтне значення, можуть використовуватися як усередині, так і поза класами символів. Крім того, всередині класу символів послідовність "b" інтерпретується як символ backspace (hex 08). Поза класом символів вона має інше значення

Третій варіант використання backslash - специфікація загального типу символів:

* \d - це будь-яке 10-річне число.
* \D - це будь-який символ, який не є 10-річним числом.
* \s - це будь-який пробільний символ.
* \S - це будь-який непробільний символ.
* \w - це будь-який "word/словниковий" символ.
* \W - це будь-який "non-word/несловниковий" символ.

Кожна пара escape-послідовностей поділяє повний набір символів на два різні набори. Будь-який цей символ збігається з однією, і лише з однією парою.

Символ "word" це будь-яка літера, або цифра, або символ підкреслення, тобто будь-який символ, який може бути частиною "word" у Perl. Визначення літер та цифр контролюється таблицями символів PCRE і може змінюватись, якщо має місце підстановка з локальною специфікою (див. раніше "Підтримка локалізації"). Наприклад, при локалізації "fr" (French) використовуються деякі символи з кодами вище 128 для введення букв з акцентами і вони збігаються з \w.

Ці послідовності типів символів можуть з'являтися як усередині, так і за межами класів символів. Кожен із них збігається з одним символом відповідного типу. Якщо поточна точка збігу є кінцем рядка-суб'єкта, всі вони зазнають невдачі, оскільки немає символу для порівняння.

Четвертий варіант - використання backslash для деяких простих тверджень. Затвердження специфікує умову, яка має бути знайдена в певній точці під час встановлення, не використовуючи жодних символів з рядка-суб'єкта. Використання субпатернів для складніших тверджень розглядається далі. Твердження зі зворотними слешами це:

* \b - це межа слова.
* \B - це не межа слова.
* \A - це початок суб'єкта (не залежить від багаторядкового режиму).
* \Z - це кінець суб'єкта або символ newline в кінці (не залежить від багаторядкового режиму).
* \z - це кінець суб'єкта (не залежить від багаторядкового режиму).

Ці твердження не можуть з'являтися в класах символів (але зауважте, що "b" має інше значення, а саме символ backspace, усередині класу символів).

Кордон слова це така позиція в рядку-суб'єкті, де поточний і попередній символи не збігаються з \w або \W (тобто один збігається з \w, а інший - з \W), або початок або кінець рядка, якщо перший або останній символ збігається з \w відповідно.

Твердження \A, \Z і \z відрізняються від традиційних circumflex і dollar (описано далі) тим, що вони збігаються тільки з початком і кінцем рядка суб'єкта, незалежно від встановлених опцій. На них не впливають опції PCRE\_NOTBOL або PCRE\_NOTEOL. Різниця між \Z та \z у тому, що \Z збігається до newline, тобто є останнім символом рядка, а також кінцем рядка, тоді як \z збігається лише наприкінці рядка.

Circumflex та dollar

Поза класом символів, у режимі підстановки за замовчуванням, символ circumflex (^) є твердженням, яке є true, тільки якщо поточна точка збігу є початком рядка-суб'єкта. Усередині класу символів circumflex має зовсім інше значення (див. далі).

Circumflex не повинен бути першим символом патерна, якщо використовуються кілька альтернатив, але повинен бути першим у кожній альтернативі, в якій з'являється, якщо патерн збігається з цією гілкою. Якщо всі наявні альтернативи починаються з circumflex (^), тобто якщо патерн обмежений для збігу тільки на початку суб'єкта, говориться, що це "заякорений/anchored" патерн. (Є також інші конструкції, які можуть викликати заякорювання патерна.)

Символ dollar є твердженням, яке TRUE, тільки якщо поточна точка збігу знаходиться в кінці рядка-суб'єкта або відразу після символу newline, який є останнім символом рядка (за замовчуванням). Dollar не повинен бути останнім символом патерна, якщо дано кілька альтернатив, але має бути останнім символом у будь-якій гілки, в якій він з'являється. Dollar не має спеціального значення у класі символів.

Значення dollar може бути змінено так, щоб він збігався тільки з кінцем рядка, через установку опціїPCRE\_DOLLAR\_ENDONLYпід час компіляції чи підстановки. Це не впливає на затвердження \Z.

Значення символів circumflex та dollar змінюється, якщо встановлена опціяPCRE\_MULTILINE. В цьому випадку вони збігаються відразу після або відразу до внутрішнього символу \n, відповідно, на додаток до збігу на початку і в кінці рядка-суб'єкта. Наприклад, патерн /^abc$/ збігається з рядком-суб'єктом "def\nabc" у багаторядковому режимі, але не інакше. Відповідно, патерни, які заякорені в однорядковому режимі, якщо всі гілки починаються з "^", не є заякореними в багаторядковому режимі. ОпціяPCRE\_DOLLAR\_ENDONLYігнорується, якщо встановленаPCRE\_MULTILINE.

Зауважте, що послідовності \A, \Z та \z можуть використовуватися для збігу з початком і кінцем суб'єкта в обох режимах, і якщо всі верви на початку патерну з \A завжди заякорені, незалежно від того, встановленаPCRE\_MULTILINEчи ні.

FULL STOP/ПОВНИЙ ЗУПИНОК

Поза класом символів точка в патерні збігається з одним із символів суб'єкта, включаючи символ, що не друкується, але не з символом (за замовчуванням) newline. Якщо встановлено опцію PCRE\_DOTALL, точки збігаються також із символами newline. Обробка точки залежить від обробки circumflex і dollar, їх ріднить тільки те, що вони обидва вводять символи newline. Крапка не має спеціального значення у класі символів.

Якщо прямо так написати у php-програмі, то нічого не працюватиме. По-перше, Ім'я\_функції та 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 кожного має стояти знак $. Жодних три крапок ставити у списку параметрів не можна. По-третє, замість слів блок\_дій у тілі функції повинен знаходитися будь-який правильний PHP-код (не обов'язково залежить від параметрів). І нарешті, після ключового слова return має йти коректне 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 факторіалу,

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

<?php  
Ім'я\_функції ("значення\_для\_параметра1", "значення\_для\_параметра2",...);  
// Приклад виклику функції  
?>

Коли можна викликати функцію? Здавалося б, дивне питання. Функцію можна викликати після визначення, тобто. у будь-якому рядку програми нижче за блок function f\_name(){...}. У PHP3 це було справді так. Але вже у PHP4 такої вимоги немає. Вся справа в тому, як інтерпретатор обробляє код, що отримується. Єдиний виняток становлять функції, що визначаються умовно (всередині умовних операторів чи інших функцій). Коли функція визначається таким чином, її визначення має передувати її виклику.

<?

$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 з 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 усередині умовного оператора Якщо функція одного разу визначена у програмі, то перевизна

чити чи видалити її не можна. Незважаючи на те, що імена функцій нечутливі до регістру, краще викликати функцію 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 всередині функції (Використовується

вкрай рідко)

Аргументи функцій

Кожна функція може мати, як ми вже говорили, список аргументів. За допомогою цих аргументів у функцію передається різна інформація (наприклад, значення числа, факторіал якого слід підрахувати). Кожен аргумент є змінною або константою.

За допомогою аргументів дані у функцію можна передавати трьома різними способами. Це передача аргументів за значенням (використовується за замовчуванням), посилання та завдання значення аргументів за умовчанням. Розглянемо ці методи докладніше.

Коли аргумент передається до функції за значенням, зміна значення аргументу всередині функції не впливає на його значення поза функцією. Щоб дозволити функції змінювати її аргументи, їх потрібно передавати за посиланням. І тому у визначенні функції перед ім'ям аргументу слід написати знак амперсанд «&».

<?php  
// напишемо функцію, яка додавала б  
// До рядка 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);  
// Викликаємо функцію  
echo $str ."><br>";  
// це виведе вже зазначену  
// радіо кнопку  
?>

приклад. Передача аргументів за посиланням

Примітка.

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

У нас є функція, що створює інформаційне повідомлення, підпис якого змінюється залежно від значення переданого їй параметра. Якщо значення параметра не встановлено, використовується підпис "Оргкомітет".

<?php

function Message($sign="Оргкомітет."){

// тут параметр sign має за замовчуванням значення

"Оргкомітет"

echo "Наступні збори відбудуться 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, Васю."

?>

приклад. Значення аргументів за умовчанням Результатом роботи цього скрипту буде:

Наступні збори відбудуться завтра. Оргкомітет.

Наступні збори відбудуться завтра.  
З повагою, Васю.

Якщо у функції кілька параметрів, то аргументи, для яких задаються значення за замовчуванням, мають бути записані після решти аргументів у визначенні функції. В іншому випадку з'явиться помилка, якщо ці аргументи будуть опущені під час виклику функції.

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

<?php  
function Add\_article($title, $description, $author="Іванов Іван"){  
echo "Заносимо до каталогу статтю: $title,"; echo "автор $author";  
echo "<br>Короткий опис: ";  
echo "$description <hr>";  
}  
Add\_article("Інформатика і ми",  
"Це стаття про інформатику...",  
"Петрів Петро");  
Add\_article("Хто такі хакери",  
"Це стаття про хакерів...");  
?>

В результаті роботи скрипту отримаємо наступне

Заносимо до каталогу статтю: Інформатика та ми,

автор Петров Петро.  
Короткий опис:  
Це стаття про інформатику...

Заносимо до каталогу статтю: Хто такі хакери,

автор Іванов Іван.

Короткий опис:

Це стаття про хакерів.

Якщо ж ми напишемо так:

<?php

function Add\_article($author="Іванов Іван",

$title, $description) {

5

// ...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

Списки аргументів змінної довжини

У PHP4 можна створювати функції зі змінним числом аргументів. Тобто ми створюємо функцію, не знаючи заздалегідь, з якими аргументами її викличуть. Для написання такої функції жодного спеціального синтаксису не потрібно. Все робиться за допомогою вбудованих функцій func\_num\_args(), func\_get\_arg(), func\_get\_args().

Функція func\_num\_args() повертає кількість аргументів, переданих у поточну функцію. Ця функція може використовуватися тільки всередині визначення функції користувача. 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  
// "Кількість аргументів функції 0" DataCheck(1,2,3);  
// виведе рядок  
// "Кількість аргументів функції 3"  
?>

приклад. Використання функції func\_num\_args()

Функція func\_get\_arg (ціло номер\_аргументу) повертає аргумент зі

списку переданих у функцію аргументів, порядковий номер якого заданий параметром\_аргументу. Аргументи функції вважаються з нуля. Як і func\_num\_args(), ця функція може використовуватися лише всередині визначення будь-якої функції.

Номер\_аргументу не може перевищувати кількість аргументів, переданих у функцію. Інакше буде згенеровано попередження і функція func\_get\_arg() поверне False.

Створимо функцію 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 перший

переданий аргумент цілим числом \*/

if ($n>=1) if (!is\_int(func\_get\_arg(0)))

$check = false;

/\* перевіряємо, чи є другий

передано аргумент рядком \*/

if ($n>=2)

if (!is\_string(func\_get\_arg(1)))

$check = false;

return $check;

}

if (DataCheck(a123, "text"))  
echo "Перевірка пройшла успішно";  
else echo "Дані не задовольняють  
умовам<br>";  
if (DataCheck(324))  
echo "Перевірка пройшла успішно";  
else echo "Дані не задовольняють умовам<br>";  
?>

приклад. Функція для перевірки типу даних, її аргументів Результатом роботи буде таке.

Дані не задовольняють умовам  
Перевірка пройшла успішно

Функція func\_get\_args() повертає масив, що складається з переліку аргументів, переданих функції. Кожен елемент масиву відповідає аргументу, переданому функції. Якщо функція використовується поза визначенням користувальницької функції, то генерується попередження.

Перепишемо попередній приклад, використовуючи цю функцію. Перевірятимемо, чи є цілим числом кожен парний аргумент, що передається 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) {

7

if (!is\_int($v)) $check = false;

// перевіряємо,

// чи є парний аргумент цілим

}

}  
return $check;  
}  
if (DataCheck("text", 324))  
echo "Перевірка пройшла успішно";  
else echo "Дані не задовольняють  
умовам<br>";  
?>

Як бачимо, комбінації функцій func\_num\_args(), func\_get\_arg() та func\_get\_args() використовуються для того, щоб функції могли мати змінний список аргументів. Ці функції були додані тільки в PHP 4. У PHP3 для того, щоб досягти такого ефекту, можна використовувати як аргумент функції масив. Наприклад, ось так можна написати скрипт, що перевіряє, чи є кожен непарний параметр функції цілим числом:

<?

function DataCheck($params){

$check = true;

$ n = count ($ params);

// Число аргументів,

// переданих у функцію

for ($i=0;$i<$n;$i++){

$v = $params[$i];

if ($i % 2 !== 0){

// перевіряємо, чи є непарний

// аргумент цілим

if (!is\_int($v)) $check = false;

}

}

return $check;

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

Використання змінних усередині функції

*Глобальні змінні*

Щоб використовувати всередині функції змінні, задані за межами неї, ці змінні потрібно оголосити як глобальні. Для цього в тілі функції слід перерахувати їхні імена після ключового слова:

global $var1, $var2;

<?

$a=1;

8

function Test\_g(){

global $a;

$a = $a\*2;

echo 'в результаті роботи функції $a=', $a;

}

echo 'поза функцією $a=',$a,', ';

Test\_g();

echo "<br>";

echo 'поза функцією $a=',$a,', ';

Test\_g();

?>

приклад. Глобальні змінні

В результаті роботи цього скрипту отримаємо:

поза функцією $a=1, в результаті роботи функції $a=2  
поза функцією $a=2, в результаті роботи функції $a=4

Коли змінна оголошується як глобальна, фактично створюється посилання глобальну змінну. Тому такий запис еквівалентний наступній (масив GLOBALS містить усі 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"].

*Статичні змінні*

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

static $var1, $var2;

Статичною змінноюможе бути надано будь-яке значення, але не посилання.

<?

function Test\_s(){

static $a = 1;

// не можна надавати вираз чи посилання

$a = $a\*2;

echo $a;

}

Test\_s(); // виведе 2

echo $a; // нічого не виведе, оскільки

// $a доступна тільки

// усередині функції

Test\_s(); // усередині функції $a=2, тому

// результатом роботи функції // буде число 4  
?>

приклад. Використання статичної змінної

Значення, що повертаються

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

Наприклад, складемо функцію, яка повертає вік людини. Якщо людина не померла, то вік вважається щодо 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); // виведе 33  
echo Age (1971, 2001); // виведе 30  
?>

У цьому прикладі можна було і не використовувати функцію return,

а просто замінити її функцією виведення echo. Однак якщо ми все ж таки робимо так, що функція повертає якесь значення (в даному випадку вік людини), то в програмі ми можемо привласнити будь-яку змінну значення цієї функції:

$ my\_age = Age (1981, 2004);

В результаті роботи функції може бути повернено лише одне значення. Декілька значень можна одержати, якщо повертати список значень (одномірний масив). Допустимо, ми хочемо отримати повний вік людини з точністю до дня.

<?php

function Full\_age($b\_day, $b\_month, $b\_year){

if (date("m")>$b\_month && date("d")>$b\_day)

{

$day = date ("d") $b\_day;

$month = date("m") $b\_month;

$year = date("Y") $b\_year;

} else {

$year = date("Y") $b\_year 1;

$ day = 31 ($ b\_day date ("d"));

10

$month = 12 ($ b\_month date("m"));

}

return array ($day,$month,$year);

}

$age = Full\_age("07", "08", "1974");  
echo "Вам $age[2] років, $age[1] місяців  
та $age[0] днів";  
// виведе "Вам 29 років, 11 місяців та 5 днів"  
?>

Коли функція повертає кілька значень для їх обробки у програмі,

зручно використовувати мовну конструкцію list(), яка дозволяє однією дією присвоїти значення відразу декільком змінним. Наприклад, у попередньому прикладі, залишивши без зміни функцію, обробити значення, що їй повертаються, можна було так:

<?  
// Завдання функції Full\_age() list($day,$month,$year) = Full\_age("07",  
"08", "1974");  
echo "Вам $year років, $month місяців та  
$ day днів ";  
?>

Взагалі конструкцію list() можна використовуватиме присвоєння

змінним значень елементів будь-якого масиву.

<?  
$arr = array("first","second");  
list($a,$b) = $arr;  
// змінною $a надається перше  
// значення масиву, $ b – друге  
echo $a," ",$b;  
// виведе рядок "first second"  
?>

приклад. Використання list()

*Повернення посилання*

В 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, якою змінною має бути присвоєне посилання. Щоб отримати з функції посилання, потрібно при оголошенні перед ім'ям написати знак амперсанд (&) і щоразу при виклику функції перед її ім'ям теж писати амперсанд (&). Зазвичай функція повертає посилання на якусь глобальну змінну (або її частину – посилання на елемент глобального масиву), посилання на статичну змінну (або її частину) або посилання на один із аргументів, якщо він був також передано за посиланням.

<?

$a = 3; $ b = 2;

function & ref($par){

global $a, $b;

11

if ($par % 2 == 0) return $b;

else return $a;

}

$var =& ref(4);

echo $var, "і", $b, "<br>";

//виведе 2 та 2

$ b = 10;

echo $var, "і", $b, "<br>";

// виведе 10 та 10

?>

приклад. Повернення посилання

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

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 у тезі form зі значенням multipart/form-data, а також елемент input типу file.

<form enctype="multipart/form-data" action="parse.php" method="post">  
<input type="hidden" name="MAX\_FILE\_SIZE" value="50000" />  
Завантажити файл: <input type="file"  
name="somefile" /><br>  
<input type="submit"  
value="Надіслати файл" />  
</form>

Зауважимо, що ми додали у формі приховане поле, яке містить у собі максимальний допустимий розмір файлу, що завантажується в байтах. При спробі завантажити файл, розмір якого більший за вказане в цьому полі значення, повинна бути зафіксована помилка, хоча це залежить від налаштування браузера та сервера. У браузері створена нами форма виглядатиме як рядок для введення тексту з додатковою кнопкою для вибору файлу з локального диска.

Тепер потрібно написати скрипт, який оброблятиме отриманий файл.

Вся інформація про завантажений на сервер файл міститься в глобальному масиві $\_FILES. Цей масив з'явився з PHP 4.1.0. Якщо включена директива register\_globals, значення переданих змінних доступні просто за їх іменами.

Якщо ми завантажили з комп'ютера-клієнта файл із ім'ям critics.htm розміром 15136 байт, то скрипт із єдиною командою print\_r($\_FILES); виведе на екран наступне:

Array ([somefile] =>

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['somefile']['name'] – Ім'я, яке мав файл на машині клієнта.
* $\_FILES['somefile']['type'] – mime-тип надісланого файлу, якщо браузер надав цю інформацію. У прикладі це text/html.
* $\_FILES['somefile']['size'] – Розмір завантаженого файлу в байтах.
* $\_FILES['somefile']['tmp\_name'] – тимчасове ім'я файлу, під яким він зберігався на сервері.
* $\_FILES['somefile']['error'] – код помилки, що виникла під час завантаження.

Тут 'somefile' – це ім'я елемента форми, за допомогою якого було здійснено завантаження файлу на сервер. Тобто вона може бути іншою, якщо елемент форми назвати інакше. Але інші ключі (name, type тощо. буд.) залишаються незмінними для будь-якої форми.

Якщо register\_globals=On, то є також додаткові змінні, такі як $myfile\_name, яка еквівалентна $\_FILES['somefile']['name'], і т.п.

Помилок при завантаженні в PHP виділяють п'ять типів і відповідно $\_FILES['somefile']['error'] може мати п'ять значень:

0 – помилки не сталося, файл завантажено успішно

1 – файл, що завантажується, перевищує розмір, встановлений директивою upload\_max\_filesize у файлі налаштувань php.ini

2 – файл, що завантажується, перевищує розмір, встановлений елементом MAX\_FILE\_SIZE форми html

3 – файл був завантажений частково

4 – файл завантажений не був

За замовчуванням завантажені файли зберігаються у часовій ди

ректорії сервера, якщо інша директорія не вказана за допомогою опції upload\_tmp\_dir у файлі налаштувань php.ini. Перемістити завантажений файл у потрібну директорію можна за допомогою move\_uploaded\_file().

Функція move\_uploaded\_file() має наступний синтаксис:

bool move\_uploaded\_file (тимчасове ім'я файлу,

місце призначення )

Ця функція перевіряє, чи файл, позначений рядком time\_name, був завантажений через механізм завантаження HTTP методом POST. Якщо це так, то файл переміщається у файл, заданий параметром місце призначення (цей параметр містить шлях до нової директорії для зберігання, так і нове ім'я файлу).

Якщо тимчасове\_ім'я\_файлу задає неправильний завантажений файл, то ніяких дій зроблено не буде, і move\_uploaded\_file() поверне

FALSE. Те саме відбудеться, якщо файл з якихось причин не може бути переміщений. І тут інтерпретатор виведе відповідне попередження. Якщо файл, заданий параметром призначення, існує, то функція move\_uploaded\_file() перезапише його.

<?

/\* У версіях PHP, раніше,

ніж 4.1.0 замість масиву

$\_FILES потрібно використовувати

масив $HTTP\_POST\_FILES \*/

$uploaddir = '/home/localhost/mysurname/uploads/'; // зберігатимемо завантажувані  
// файли до цієї директорії  
$destination = $uploaddir . $\_FILES['somefile']['name'];

// Ім'я файлу залишимо незмінним

print "<pre>";

if (move\_uploaded\_file(

$\_FILES['somefile']['tmp\_name'],

$destination)) {

/\* переміщуємо файл із тимчасової папки

у вибрану директорію для зберігання \*/

print "Файл ".$\_FILES['somefile']['name']. " успішно

завантажений <br>";

} else {

echo "Помилка при завантаженні файлу.

Деяка налагоджувальна інформація:<br>";

print\_r($\_FILES);

}

print "</pre>";

?>

*Змінні сервери*

Повний список змінних сервера, що встановлюються, міститься в документації мови (ел. документ).

Виконання роботи

Опис завдань, що вирішуються серверним сценарієм.

Серверний сценарій (PHP-скрипт) реалізує міні-калькулятор для обробки даних, отриманих із завантажених користувачем текстових файлів. Основні завдання, які вирішуються цим сценарієм, такі:

1. Завантаження і обробка файлів

Користувач завантажує два текстових файли, кожен з яких містить список чисел, розділених символом «;». Сценарій зчитує ці файли, розбиває їх на окремі числові значення та перетворює їх у числовий формат. Цей процес гарантує, що дані будуть готові для обчислень, та запобігає помилкам, пов'язаним з неправильним форматом введених даних.

1. Перевірка коректності формату файлів

Скрипт перевіряє, чи обидва завантажені файли містять однакову кількість чисел. Це критично для правильності подальших обчислень, оскільки кожна операція виконується над відповідними парами чисел. Якщо файли містять різну кількість чисел, виводиться повідомлення про помилку, і обчислення не проводяться.

1. Виконання математичних операцій над парами чисел

Користувач може обрати одну з наступних операцій: додавання, віднімання, множення, ділення та піднесення до степеня. Сценарій виконує вибрану операцію над кожною парою чисел, отриманою з обох файлів, і виводить результати для кожної пари.

1. Обробка можливих помилок у процесі обчислень

Під час обчислень сценарій перевіряє можливі помилки, наприклад, ділення на нуль. У разі виникнення такої ситуації для відповідної операції виводиться повідомлення про помилку, замість числового результату. Це допомагає забезпечити надійність та коректність роботи калькулятора.

1. Генерація звіту з результатами обчислень

Після виконання всіх обчислень сценарій генерує HTML-документ, у якому відображаються результати кожної операції над парами чисел. У звіті також зазначаються дата і час генерації, а внизу документу виводиться інформація про виконавця. Цей формат забезпечує зручне відображення та зберігання результатів роботи.

Короткий опис алгоритму роботи сценарію.

1. Завантаження файлів і перевірка даних

* Користувач завантажує два текстових файли, які сервер приймає через форму за допомогою полів file1 і file2.
* Сценарій перевіряє, чи завантажені обидва файли та обрана операція. У разі відсутності цих даних виводиться повідомлення про помилку.

1. Зчитування чисел із файлів

* Сценарій зчитує вміст кожного файлу та розбиває його на масив чисел, використовуючи символ «;» як роздільник.
* Отримані значення конвертуються в числовий формат для подальшої обробки.

1. Перевірка відповідності кількості чисел

* Після зчитування даних з файлів перевіряється, чи обидва файли містять однакову кількість чисел. Це гарантує можливість проведення операцій над кожною парою чисел.
* Якщо кількість чисел не збігається, сценарій припиняє роботу і виводить повідомлення про помилку.

1. Виконання обраної операції над парами чисел

* Залежно від вибору користувача, виконується одна з операцій (додавання, віднімання, множення, ділення або піднесення до степеня) для кожної пари чисел.
* Якщо обрано операцію ділення, сценарій додатково перевіряє, чи не дорівнює друге число нулю, щоб уникнути помилки ділення на нуль. У разі ділення на нуль виводиться відповідне повідомлення.

1. Генерація HTML-документа з результатами

* Сценарій створює HTML-документ, у якому відображаються результати обчислень для кожної пари чисел.
* У документі виводяться дата та час генерації, а також дані про виконавця (ПІБ і група).

Алгоритм сценарію забезпечує обробку файлів, виконання обчислень над числовими даними та коректне відображення результатів у HTML-документі.

Опис використовуваних вхідних даних і функцій

**Вхідні дані**

1. **Файли з числами:**

* Користувач завантажує два текстових файли (file1 і file2). Кожен з файлів містить список чисел, розділених крапкою з комою (;).
* Важливо, щоб обидва файли мали однакову кількість чисел. Якщо кількість чисел не співпадає, виводиться повідомлення про помилку.

1. **Операція:**

* Користувач вибирає одну з п'яти операцій: додавання, віднімання, множення, ділення або піднесення до степеня. Цей вибір передається через форму як значення змінної operation.
* У разі, якщо операція не вибрана або її значення не відповідає жодній з опцій, встановлюється відповідне повідомлення про помилку.

**Функції**

1. **loadNumbersFromFile($filePath)** – функція виконує завантаження чисел з файлу.

*Параметри:*

* $filePath: шлях до тимчасового файлу, який завантажив користувач.

*Логіка роботи:*

* Функція зчитує весь вміст файлу за допомогою file\_get\_contents.
* Якщо файл порожній, функція повертає порожній масив, щоб уникнути подальших помилок обробки.
* Якщо файл містить текст, він розділяється на частини за допомогою explode(";"), а отримані значення перетворюються у числа за допомогою array\_map('floatval', $numbers).

**Обробка вхідних даних:**

1. **Перевірка на наявність файлів і вибір операції:**

* Якщо користувач не завантажив обидва файли або не вибрав операцію, програма завершується з повідомленням про помилку.

1. **Перевірка на порожні файли:**

* Якщо один або обидва файли виявляються порожніми, програма виводить відповідне повідомлення і не продовжує обчислення.

1. **Перевірка на однакову кількість чисел:**

* Якщо кількість чисел у файлах не збігається, виводиться повідомлення про помилку, а обчислення не виконуються.

**Вихідні дані**

1. **Результати обчислень**:

* На сторінці виводиться заголовок, що містить назву виконуваної операції та результат для кожної пари чисел.
* В разі виникнення помилки, користувач отримує повідомлення з деталями проблеми.

1. **Дата і підпис**:

* Виводиться час та дата генерації результату роботи, а також підпис користувача.

Лістинг вихідного коду з коментарями

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

Результати роботи

Результати роботи сценарію подано в таблиці 1.

Таблиця 1 – результат роботи сценарію з різними вхідними данними

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Вхідні значення 1 файлу** | **Вхідні значення 2 файлу** | **Операція** | **Очікуваний результат** | **Обробка помилок** |
| 1;2;3 | 4;5;6 | Додавання | 1+4=5, 2+5=7, 3+6=9 | - |
| 10;20;30 | 5;5;5 | Віднімання | 10-5=5, 20-5=15, 30-5=25 | - |
| 2;3;4 | 5;6;7 | Множення | 25=10, 36=18, 4\*7=28 | - |
| 9;12;15 | 3;4;5 | Ділення | 9/3=3, 12/4=3, 15/5=3 | - |
| 2;3;4 | 1;2;3 | Піднесення до степеня | 2^1=2, 3^2=9, 4^3=64 | - |
| 2;0;4 | 2;2;0 | Ділення | 2/2=1, 0/2=0, Помилка: ділення на нуль | Помилка оброблена при діленні на нуль |
| 10;20;30 |  | Додавання | Помилка: один або обидва файли порожні | Виведення помилки для порожнього другого файлу |
|  | 4;5;6 | Віднімання | Помилка: один або обидва файли порожні | Виведення помилки для порожнього першого файлу |
| 1;2 | 3;4;5 | Множення | Помилка: кількість чисел у файлах не збігається | Виведення помилки через різну кількість чисел |
| abc;def | 1;2;3 | Додавання | Помилка: неможливо перетворити текст у числа | Програма повинна ігнорувати неправильні дані |
|  |  | Множення | Помилка: відсутні необхідні файли або не вибрано операцію | Пропущені обов'язкові вхідні дані |
| 2;3; | ;5;6 | Піднесення до степеня | Помилка: один або обидва файли порожні | Пропуск порожніх значень, обробка порожніх файлів |

висновок

В рамках виконання роботи було поставлено завдання розробити серверний сценарій на мові PHP, що реалізує "Міні-калькулятор" для обробки даних із завантажених файлів та виконання математичних операцій над парами чисел. Основною метою було здобуття навичок роботи з PHP, обробки файлів на сервері та валідації вхідних даних.

У процесі виконання роботи було:

* Досліджено синтаксис PHP для серверного програмування та роботу з формами, що дозволило надійно обробляти дані від користувача;
* Ознайомлено з типами даних у PHP, зокрема числовими та рядковими, що дозволило коректно обробляти вміст файлів із числами;
* Розроблено функцію для завантаження та перетворення чисел із файлів, а також перевірку на порожні та некоректні дані;
* Реалізовано логіку виконання математичних операцій, таких як додавання, віднімання, множення, ділення та піднесення до степеня, з обробкою можливих помилок (наприклад, ділення на нуль);
* Створено систему виведення повідомлень про помилки, що допомагає користувачу швидко зрозуміти причину некоректної роботи скрипта (різна кількість чисел, порожні файли, некоректний формат);
* Виведено результати у форматі HTML-документа із зазначенням обраної операції, дати виконання та ПІБ виконавця, що покращує зручність користування скриптом.

В результаті роботи створено ефективний "Міні-калькулятор", який обробляє дані з файлів та виконує вибрані математичні операції, демонструючи засвоєні навички програмування на PHP. Робота дозволила розширити знання з PHP, вивчити обробку файлів на сервері, забезпечення валідації та опрацювання помилок у веб-додатках.

*Додаток А*Машинний лістинг файлу **calculator\_for\_files.php**

<!DOCTYPE html>

<html lang="uk">

<head>

<meta charset="UTF-8">

<title>Міні-калькулятор з обробкою файлів</title>

</head>

<body>

<?php

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

$errormsg = "";

if (!isset($\_FILES['file1'], $\_FILES['file2'], $\_POST['operation'])) {

$errormsg .= "Помилка: відсутні необхідні файли або не вибрано операцію.<br>";

exit;

}

$file1Path = $\_FILES['file1']['tmp\_name'];

$file2Path = $\_FILES['file2']['tmp\_name'];

$operation = $\_POST['operation'];

// Встановлення назви операції для відображення

switch ($operation) {

case 'addition':

$operationName = 'Додавання';

break;

case 'subtraction':

$operationName = 'Віднімання';

break;

case 'multiplication':

$operationName = 'Множення';

break;

case 'division':

$operationName = 'Ділення';

break;

case 'exponentiation':

$operationName = 'Піднесення до степеня';

break;

default:

$operationName = 'Невідома операція';

}

// Функція для завантаження чисел з файлу

function loadNumbersFromFile($filePath)

{

$content = file\_get\_contents($filePath);

if (trim($content) === '') {

return []; // Повертаємо порожній масив, якщо файл порожній

}

$numbers = explode(";", trim($content));

return array\_map('floatval', $numbers);

}

// Завантаження чисел з файлів

$numbers1 = loadNumbersFromFile($file1Path);

$numbers2 = loadNumbersFromFile($file2Path);

// Перевірка на порожні файли

if (empty($numbers1) || empty($numbers2)) {

$errormsg .= "Помилка: один або обидва файли порожні. Будь ласка, завантажте файли з числами.<br>";

}

// Перевірка на однакову кількість чисел у файлах

if (count($numbers1) !== count($numbers2)) {

$errormsg .= "Помилка: кількість чисел у файлах не збігається.<br>";

}

if (!empty($errormsg)) {

echo "<h2>Відбулася помилка під час виконання:</h2>";

echo "$errormsg";

} else {

// Якщо всі перевірки пройдено, виконуємо обчислення

echo "<h2>Результати обчислень:</h2>";

echo "<h3>Операція, яка виконувалась: $operationName</h3>";

// Обробка кожної пари чисел

foreach ($numbers1 as $index => $number1) {

$number2 = $numbers2[$index];

$operationResult = '';

// Виконання обраної операції

switch ($operation) {

case 'addition':

$operationResult = $number1 + $number2;

break;

case 'subtraction':

$operationResult = $number1 - $number2;

break;

case 'multiplication':

$operationResult = $number1 \* $number2;

break;

case 'division':

$operationResult = ($number2 != 0) ? ($number1 / $number2) : "Помилка: ділення на нуль";

break;

case 'exponentiation':

$operationResult = pow($number1, $number2);

break;

default:

$operationResult = "Невідома операція";

}

echo "Результат для пари чисел $number1 і $number2: <u>$operationResult</u><br>";

}

}

?>

<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 з файлами</title>

</head>

<body>

<h2>Міні-калькулятор з файлами</h2>

<form action="calculator\_for\_files.php" method="POST" enctype="multipart/form-data" target="\_blank">

<label for="file1">Файл 1:</label><br>

<input type="file" id="file1" name="file1" accept=".txt" required><br><br>

<label for="file2">Файл 2:</label><br>

<input type="file" id="file2" name="file2" accept=".txt" required><br><br>

<label>Оберіть операцію:</label><br>

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

<label for="addition">Додавання</label><br>

<input type="radio" id="subtraction" name="operation" value="subtraction">

<label for="subtraction">Віднімання</label><br>

<input type="radio" id="multiplication" name="operation" value="multiplication">

<label for="multiplication">Множення</label><br>

<input type="radio" id="division" name="operation" value="division">

<label for="division">Ділення</label><br>

<input type="radio" id="exponentiation" name="operation" value="exponentiation">

<label for="exponentiation">Піднесення до степеня</label><br><br>

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

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

</form>

</body>

</html>