Лабораторна робота №3

ВБУДОВАНІ ФУНКЦІЇ PHP. ЗМІННІ СЕРВЕРИ. ЗАВАНТАЖЕННЯ ФАЙЛІВ

Мета роботи:вивчення можливостей створення і використання вбудованих функцій мови, ознайомлення зі змістом змінних сервера, а також отримання практичних навичок із завантаження файлів на сервер та роботі з ними.

Досліджуваний матеріал: правила створення функцій користувача, аргументи функцій, аргументи за замовчуванням, значення, що повертаються, вбудовані функції (ел. документ – посібник з мови), принципи і методи передачі запитів на сервер, обробка запитів у PHP, основні функції виведення, масиви змінних сервера.

1. Постановка задачі

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

2. Порядок виконання роботи

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

3. 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.

4. Основні теоретичні відомості

4.1. Функції, визначені користувачем

У програмуванні, як і математики, функція є відображення безлічі її аргументів на безліч її значень. Тобто функція кожного набору значень аргументу повертає якісь значення, що є результатом її роботи.

Функцію можна визначити за допомогою наступного синтаксису:

```
function Ім'я_функції (параметр1, параметр2,
... параметр N) {
Блок_дій
return "значення повертається функцією";
}
```

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 перевизначити 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)

4.2. 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) {
```

```
// ...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) {
```

```
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>";
?>
```

4.3. 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;
```

```
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

4.4. 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"));
```

```
$month = 12 - ($ 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;
```

```
if ($par % 2 == 0) return $b;
else return $a;
}
$var =& ref(4);
echo $var, "i", $b, "<br>";
//виведе 2 та 2
$ b = 10;
echo $var, "i", $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.

4.5. 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 директорії 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).

5. Контрольні питання

1. Опишіть синтаксис, щоб створити функції користувача.
2. Які правила визначення та виклику функцій?
3. Яким чином записуються аргументи функції?
4. Що таке аргументи за промовчанням?
5. Як передаються аргументи функції?
6. Що робити, коли кількість аргументів наперед невідома?
7. Як оголосити глобальну змінну всередині функції?
8. Як і навіщо використовуються статичні змінні?
9. Скільки значень може відновити функція?
10. Чим відрізняється повернення значення та виведення значення?

11. Як функція може одночасно повернути декілька значень?
12. Навіщо використовується конструкція list()?
13. Як використовується повернення функцією посилання на значення?
14. Опишіть типовий сценарій завантаження файлу на сервер?
15. Де міститься інформація про завантажені файли?
16. Яке ім'я має файл після завантаження?
17. Який тип запиту використовується для завантаження файлу на сервер?
18. Яка функція використовується для обробки завантаженого файлу?
19. Які ви знаєте серверні змінні і для чого вони служать?

6. Варіанти завдань

У кожному варіанті вихідна форма документа повинна містити заголовок з назвою роботи, внизу документа – ПІБ виконавця (можна брати з попередньої роботи).

http://localhost/прізвище_виконавця/ім'я_скрипту.php

Варіанти:

1. «Простий споживчий кошик».

Забезпечити завантаження та розбір текстового файлу, що містить перелік товарів з описами та цінами (замість запису товару у вигляді масиву). Пропонується відокремлювати записи знаком вертикальної межі “|”, поля – знаком точка з комою «;». Наприклад:

Сир; Сир голландський; 23,51 | Олія; Олія селянська; 17,45 |

Файл зберігати в директорії скрипта з ім'ям, що містить дату та час завантаження файлу на сервер. Подальша обробка кошика як у попередній ЛР.

2. "Сортування багатовимірного масиву".

Забезпечити завантаження та розбір текстового файлу, що містить вихідну таблицю рядкових значень. Рядки таблиці відокремлюються знаком вертикальної межі “|”, елементи у рядках – знаком крапка з комою «;». Дозволяється різна кількість елементів у рядках таблиці. Файли зберігаються у каталозі скрипта і щоразу перезаписуються. Подальша обробка як у попередній ЛР.

3. "Міні-калькулятор".

Забезпечити завантаження та розбір двох текстових файлів, що містять однакову (але не завжди те саме) кількість чисел, відокремлених знаком «;». Усі операції проводити над відповідними парами чисел. Забезпечити перевірку коректності формату завантажуваних файлів (рівна кількість чисел). Файли зберігаються у каталозі скрипта і щоразу перезаписуються.

4. "Розбір рядка".

Забезпечити завантаження та розбір текстового файлу з довільним змістом (слова та числа). Виділити з файлу числа, підрахувати їхню кількість. Подальша обробка як у попередній ЛР (рахувати виділені з тексту числа, як числа, введені користувачем). Вихідний файл записується в каталог скрипта з ім'ям, що містить дату та час завантаження.

5. "Транспонування матриці".

Забезпечити завантаження та аналіз текстового файлу, що містить вихідну матрицю будь-якої розмірності. Рядки матриці пропонується відокремлювати знаком абзацу "\n", елементи в рядках - знаком крапка з комою ";". Забезпечити перевірку коректності вихідного файлу (наявність всіх елементів матриці). Вихідний файл записується в каталог скрипту і щоразу перезаписується.

6. "Тест".

Забезпечити завантаження та розбір двох текстових файлів, що містять перелік питань тесту з варіантами відповідей та правильні відповіді відповідно. Запитання (і відповіді) пропонується відокремлювати знаком абзацу “\n”, елементи у рядках – знаком точка з комою «;».

Всі питання містять лише одну правильну відповідь, питання представляється групою радіокнопок (radiobutton). Приклад пропонованого формату файлу:

1;Питання1;Відповідь11;відповідь12;відповідь13

2;Питання2;Відповідь21;відповідь22;відповідь23

Формат файлу відповідей:

1; відповідь12

2; відповідь21

Подальша обробка як у попередній ЛР. Файли запитань та відповідей зберігаються в директорії скрипту та перезаписуються.

6. Список літератури, що рекомендується

1. Розробка Web-додатків на PHP та MySQL: Пер. з англ. / Лаура Томсон, Люк Веллінг. - 2-ге вид., Випр. - СПб: ТОВ «ДіаСофт», 2003. - 672 с.
2. Аргеріх Л. та ін. Професійне PHP програмування, 2-ге видання. - Пров. з англ. – СПб: Символ-Плюс, 2003. – 1048 с., іл.
3. Кузнєцов І.В., Симдянов І.В., Голишев С.В. PHP 5. Практика розробки веб-сайтів. - СПб.: БХВ-Петербург, 2005. - 960 с.
4. Кузнєцов І.В., Симдянов І.В., Голишев С.В. PHP 5 на прикладах. - СПб.: БХВ-Петербург, 2005. - 576 с.
5. Коггзолл, Джон. PHP5. Повне керівництво: Пер. з англ. - М.: Видавничий дім "Вільямс", 2006. - 752 с.
6. Колісніченко Д.М. Самовчитель PHP 5. СПб.: Наука та техніка, 2004. - 576 с.
7. http://www.php.net