1 Основи серверної мови створення сценаріїв – PHP

Мета: набути практичних навичок з використання базових операторів PHP.

Завдання: дослідити функціонування базових операторів PHP

Обладнання: ЕОМ, програмне забезпечення з відкритим кодом (opensource): PHP8, Apache 2.4.

1.1 Стилі PHP-дескрипторів

Існують чотири стилі PHP-дескрипторів (наведені нижче фрагменти коду еквівалентні):

– XML-стиль

```
<?php echo '<p> Hello world!</p>'; .
```

Даний стиль найбільш поширений (у даних методичних вказівках використовується саме такий стиль), адміністратори серверів не мають можливості вимкнути його, тому він гарантовано доступний у всіх випадках, що важливо у випадку розробки програмного коду, виконання якого передбачається у різних середовищах.

– Скорочений стиль

```
<? echo '<p> Hello world!</p>'; .
```

Це найпростіший стиль, що відповідає стилю інструкцій обробки мови SGML (Standard Generalized Markup Language). Використання його не бажане, оскільки системні адміністратори часто його вимикають для запобігання конфліктів з XML-документами.

– SCRIPT-стиль

```
<script language='php'>
     echo  '<p> Hello world!</p>';
</script>.
```

SCRIPT -стиль часом застосовують у випадку виникнення проблем з іншими стилями у HTML-редакторах.

– ASP-стиль

```
<% echo  '<p> Hello world!</p>'; %>
```

Аналогічний стиль дескриптора використовується у технологіях ASP (Active Server Pages). Такому стилеві надають перевагу при роботі з редактором, орієнтованим на ASP.NET.

1.2 Оператори PHP

Дії, котрі повинен виконати інтерпретатор PHP, задаються операторами PHP, розташованими між відкриваючими та закриваючими дескрипторами:

```
echo '<p> Hello world!</p>';
```

Оператор `echo` виконує виведення у вікно браузера переданого йому рядка. Для розділення операторів використовується символ-крапка з комою: “;”.

1.3 Пробіли

Порожні символи, такі як порожній рядок (повернення каретки), пробіли між словами та символи табуляції, утворюють категорію пробільних. Браузери ігнорують пробільні символи у HTML-коді. Аналогічно діє і механізм PHP. Пробіли між PHP-операторами не є необхідними, проте вони підвищують читабельність коду. Фрагменти

```
echo 'Вітаємо';
echo 'на нашому сайті';
echo 'Вітаємо'; echo'на нашому сайті';
```

еквівалентні, але перша версія візуально сприймається легше.

1.4 Коментарі

Зазвичай коментарями супроводжуються більшість PHP-сценаріїв, за виключенням найпростіших.

Інтерпретатор PHP ігнорує текст, розміщений у коментарі. Для синтаксичного аналізатора коментар рівнозначний пробілам.

PHP підтримує коментарі у стилі C, C++ і сценаріїв оболонки.

```
/*
     Приклад
     багаторядкового коментаря С-стилю
*/
```

Багаторядкові коментарі починаються з символів `/*` і закінчуються символами `*/`. Коментарі не можуть бути вкладеними.

Також використовують однорядкові коментарі у стилі C++:

```
echo '<p>Замовлення виконано</p>'; // початок виведення замовлення
```

або у стилі сценаріїв оболонки

```
echo '<p>Замовлення виконано</p>'; # початок виведення замовлення.
```

1.5 Доступ до змінних форми

Сенс використання форми полягає в отриманні скриптом інформації, введеної користувачем у текстові поля форми. Розглянемо приклад:

```
<html>
<head>
```

```
<title>Приклад форми</title>
</head>
<body>
<form action="form.php" method="post">
<input type="text" name="formvariable">
<input type="submit" value="Додати">
</form>
</body>
</html>
```

Даний HTML-скрипт створює форму для введення даних і кнопку “Додати”, яка передає введене скриптові form.php

Доступ до вмісту поля PHP-скриптом отримують наступним чином (файл `form.php`):

```
<?php
$variable=$_POST['formvariable'];
echo $variable;
?>
```

У даному випадкові передача даних реалізована за допомогою методу `POST`. У цьому випадкові відбувається неявна передача: користувач не помічає ніяких зовнішніх ознак процесу і не може здійснити припущення про імена змінних. Така передача зручна з точки зору безпеки.

Існує інший спосіб передачі даних – `GET`. При цьому дані передаються за допомогою сформованого додатку до назви файлу, що отримує дані. Зовні виглядає як лінк, при цьому користувач отримує доступ до імен змінних, що передаються, та їхніх значень.

```
<html>
<head>
<title>Приклад форми</title>
</head>
<body>
<form action="form.php" method="get">
<input type="text" name="formvariable">
<input type="submit" value="Додати">
</form>
</body>
</html>
```

Даний приклад відрізняється від попереднього методом передачі: замість `POST` використано `GET`. Нехай у поле введено значення “test”. Після натискання кнопки “Додати” буде особливим чином активізовано скрипт `form.php`: до адреса файла буде додано символ “?”, назву змінної, переданої з форми, знак “=” та значення змінної:

```
http://localhost/form.php?formvariable=test
```

`PHP`-скрипт при цьому змінюють наступним чином:

```
<?php
$variable=$_GET['formvariable'];
echo $variable;
```

```
?>
```

Масиви `$_POST` та `$_GET` належать до категорії суперглобальних. Як у першому, так і у другому випадках, доступ до даних форми можна отримати через масив `$_REQUEST`.

```
<?php
$variable=$_REQUEST['formvariable'];
echo $variable;
?>
```

Детально використання масивів буде розглянуто у лабораторній роботі №2.
Зазвичай блоки отримання даних з форми розташовують на початкові скрипта.

1.6 Конкатенація рядків

Операція конкатенації рядків використовується для об'єднання рядків (фрагментів тексту) у єдиний текст. Вона часто застосовується при виведенні даних у браузер за допомогою оператора `echo`. Реалізується за домового символу оператора конкатенації – крапки ".".

```
<?php
$text1="Hello, ";
$text2="world!";
echo $text1.$text2;
?>
```

```
<?php
$text1="world!";
echo "Hello, ".$text1;
?>
```

Довільну змінну, відмінну від змінної масиву, можна помістити у подвійні лапки і застосувати до неї оператор `echo`. Зверніть увагу на результат виконання двох наступних скриптів:

```
<?php
$text1="world!";
echo "Hello, $text1";
?>
```

Буде виведено у вікно браузера: Hello, world!

```
<?php
$text1="world!";
echo 'Hello, $text1';
?>
```

Результат: Hello, $text1.

Якщо ім’я змінної знаходиться між подвійними лапками, то ім’я замінюється значенням змінної, якщо ім’я змінної чи довільний текст обмежені одинарними лапками, то вони передаються без змін.

1.7 Ідентифікатори

Ідентифікатори – це імена змінних, функцій та класів. Використання ідентифікаторів регламентується наступними правилами:

– ідентифікатори можуть мати довільну довжину і складатися з букв, цифр та символів підкреслювання "_";

– ідентифікатори не можуть починатися з цифри;

– у PHP ідентифікатори чутливі до регістру символів. Змінні `$formvariable` `$FormVariable` – це різні змінні. Виключення складають вбудовані PHP-функції – їхні імена можуть бути представлені довільним регістром;

– змінні можуть мати імена, що співпадають з іменами вбудованих функції. Однак, це може призвести до плутанини, тому подібних ситуацій слід уникати. Не можна також створювати функції, чиї імена співпадають з іменами вбудованих функцій.

Імена змінних у PHP починаються знаком долара ($). Пропуск цього знаку – поширена помилка.

1.8 Типи змінних

Тип змінної характеризується видом даних, котрі вона зберігає. PHP підтримує наступні базові типи даних:

– Integer (цілий) – використовується для представлення цілих чисел;

– Float (ще їх називають double – подвійної точності) – використовується для представлення дійсних чисел;

– String (рядковий) – використовується для представлення символів;

– Boolean (булевий) – використовується для зберігання значень true та false;

– Array (масив) – використовується для зберігання декількох елементів даних (буде розглянутий у лабораторній роботі №2);

– Object (об’єкт) – використовується для зберігання екземплярів класу (буде розглянутий у лабораторній роботі №5).

Доступні також два спеціальних типи – NULL та resource (ресурс). Змінні, котрим не присвоєно конкретного значення, котрі не визначені або набувають значення NULL, належать до типу NULL. Деякі вбудовані функції (такі, як для роботи з базами даних) повертають змінну ресурсного типу (наприклад, з’єднання з базами даних).

Також PHP підтримує типи pdfdoc та pdfinfo, якщо встановлена підтримка обробки PDF-документів.

Мова PHP – слабо типізована. Тип змінної визначається типом присвоєного їй значення.

1.9 Перетворення типів

За допомогою механізму перетворення типів можна переводити змінну або конкретне значення до іншого типу. Перетворення відбувається так само, як і на мові С.

```
<?php
 $a=2.34;      // Змінна $a має тип float
 $a=(int)$a;   // Тепер змінна $a має тип integer
 echo $a;
```

1.10 Змінні змінних

Усі мови програмування дозволяють змінювати значення змінної, деякі – тип змінної і зовсім небагато мов дозволяю змінювати ім'я змінної.

В основі цієї можливості – ідея використання значення однієї змінної як імені іншої.

Нехай існує змінна $variable. Тоді:

```
<?php
$variable=NULL;
$varname="variable";
$$varname="Hello, world!";
echo $variable;
```

Запис `$$varname="Hello, world!"` еквівалентний наступному: `$variable="Hello, world!".`

Константи

Разом із використанням змінних, PHP надає можливість оголошення констант. Як і змінна, константа містить значення, але її значення встановлюється одноразово і не може бути зміненим у сценарії.

Константи оголошуються за допомогою функції `define`

```
<?php
define('CONSVALUE',10);
echo CONSVALUE;
```

Зверніть увагу на те, що імена констант записують символами у верхньому регістрі. Дана особливість перейшла з мови С. Дотримуватися її не обов'язково, проте вона значно спрощує читання і супровід коду.

Важливо: при звертанні до констант не використовується знак долара ($).

Область дії змінних

Термін “область дії” належить тим розділам сценаріїв, в яких можливий доступ до деякої конкретної змінної, іншими словами, це – область, з довільного місця якої видно дану змінну. У PHP використовують шість базових правил визначення області дії:

– вбудовані суперглобальні змінні видно з довільного місця сценарію;

– константи, щойно вони оголошені, видно глобально, тобто можуть використовуватися як зовні так і всередині функцій;

– глобальні змінні, оголошені у сценарії, видно у довільному місці сценарію, але не всередині функцій;

– змінні, що використовуються всередині функцій, що оголошені як глобальні, посилаються на глобальні змінні з тими самими іменами;

– змінні, що використовуються всередині функцій, що оголошені як статичні, невидимі поза межами функцій, проте вони зберігають свої значення поміж двома викликами цих функцій;

– змінні, створені всередині функцій, є локальними стосовно своєї функції і припиняють існування після завершення функції.

Вбудовані суперглобальні змінні у PHP версій від 4.1 і вище наступні:

– `$_GLOBALS` – масив глобальних змінних. Дає можливість доступу до глобальних змінних всередині функції, наприклад: `$_GLOBALS['myvariable']`;

– `$_SERVER` – масив змінних середовища сервера;

– `$_GET` – масив змінних, переданих у сценарій за допомогою методу GET;

– `$_POST` – масив змінних, переданих у сценарій за допомогою методу POST;

– `$_COOKIE` – масив cookie-змінних;

– `$_FILES` – масив змінних завантаження файлів;

– `$_ENV` – масив змінних оточення;

– `$_REQUEST` – масив, пов’язаний із введенням даних користувачем, включаючи `$_GET, $_POST, $_COOKIE`;

– `$_SESSION` – масив змінних сеансу.

Посилання

Операція посилання позначається як “`&`” і може використовуватися у поєднанні з операцією присвоювання.

Зазвичай, коли значення змінної `$a` присвоюється змінній `$b`, створюється копія змінної `$a`, яка зберігається у пам’яті. Якщо у майбутньому значення `$a` буде змінене, то `$b` зміни не торкнуться.

```
$a=5;
$b=$a;     // $b=5;
$a=6;      // як і раніше, $b=5
```

Створення копії можна уникнути, використавши операцію посилання `&`:

```
$a=5;
$b=&$a;    // $b дорівнює 5;
$a=6;      // зверніть увагу: $b дорівнює 6
```

`$a` та `$b` вказують на одну і ту ж ділянку пам’яті. Цей зв’язок можна розірвати, “скинувши” одну зі змінних:

```
unset($a);
```

При цьому значення $b продовжуватиме існувати.

Операції порівняння

Операції порівняння наведено у табл.1.1. Результатом виконання операції порівняння є true або false.

Таблиця 1.1 – Операції порівняння PHP

Операція	Назва	Використання
==	дорівнює	$a == $b
===	тотожно	$a === $b
!=	не дорівнює	$a != $b
!==	не тотожно	$a !== $b
<>	не дорівнює	$a <> $b
<	менше	$a < $b
>	більше	$a > $b
<=	менше або дорівнює	$a <= $b
>=	більше або дорівнює	$a >= $b

Операція перевірки тотожності повертає значення true тільки у тому випадкові, якщо обидва операнди рівні і мають однаковий тип. Наприклад:

```
$a=0;
$b='0';
echo($a==$b);  // результат:  true
echo($a===$b); // результат: false
```

Логічні операції слугують для комбінування результатів логічних умов. Перелік логічних операцій разом з описом їхнього застосування наведено у табл. 1.2.

Таблиця 1.2 – Логічні операції PHP

Операція	Назва	Використання	Результат
!	НЕ	!$b	Повертається true, якщо значення $a дорівнює false та навпаки
&&	І	$a && $b	Повертається true, якщо обидві змінні $a та $b мають значення true, в іншому випадкові повертається значення false
||	АБО	$a || $b	Повертається true, якщо довільна зі змінних $a або $b має значення true, інакше повертається false
and	І	$a and $b	Те саме, що і &&, але з меншим пріоритетом
or	АБО	$a or $b	Те саме, що і ||, але з меншим пріоритетом

Оператори `if`

Для прийняття рішень використовується оператор `if`. Операторові необхідно задати умову. Якщо умова рівна true, то виконується блок коду, розташований після оператора. Умова в операторі if записується поміж круглими дужками “`(…)`”. Приклад:

```
<?php
$a=0;
$b=3;
if($a<$b) echo '$a < $b';
```

1.1.16. Блоки коду

Часто всередині такого умовного оператора, як if, необхідно виконати більше одного оператора. У такому випадку відповідна послідовність операторів записується у вигляді блоку. Для оголошення блоку оператори необхідно оточити фігурними дужками:

```
if($a<$b)
{
     echo '$a < $b';
     $a=$b;
}
```

Таким чином при виконанні умови `($a<$b)` буде виконано обидва оператори

```
echo '$a < $b';
$a=$b;
```

Якщо ж умова не виконуватиметься, то обидва оператори будуть проігнорованими.

Оператори `else`

Оператор `else` дозволяє визначити альтернативну дію, котра повинна виконатися, якщо значення умови в операторі `if` виявиться `false`. Наприклад:

```
<?php
$a=0;
$b=3;
echo '$a='.$a.'; $b='.$b.'<br />';
if($a<$b)
     echo '$a < $b';
else
     echo '$a < $b';
?>
```

Результат виконання скрипта:

$a=0; $b=3
$a < $b

Вкладання операторів `if` один в одного дозволяє будувати складні ланцюжки.

Оператори `elseif`

У багатьох випадках прийняття рішення передбачає вибір відповідного варіанту з деякої множини можливих варіантів. Послідовність цієї множити можна створити за допомогою оператора комбінації операторів `if – else.` За наявності послідовності умов програма може перевіряти кожну з них до тих пір, поки не знайде таку, значення якої буде `true`. Наприклад:

```
<?php
$a=23;
if($a>0 && $a<10) echo 'Результат - у першому інтервалі';
elseif($a>=10 && $a<20) echo 'Результат - у другому
інтервалі';
elseif($a>=20 && $a<30) echo 'Результат - у третьому
інтервалі';
?>
```

Після виконання скрипта отримують: “Результат - у третьому інтервалі”

Оператори `switch`

Оператор `switch` працює аналогічно до оператора `if`, але надає можливість умовному виразові мати як результат більше двох значень. В операторі `if` умова набуває значення `true` або `false`. Натомість в операторі `switch` умова може набувати довільну кількість значень у тих випадках, коли результат його обчислення – простий тип (`integer`, `string` чи `float`). Для забезпечення реагування на кожне таке значення слід передбачити для нього відповідний оператор `case`, а також (не обов’язково) визначити дії, що виконуватимуться по замовчуванню, якщо виникне випадок, не передбачений оператором case. Наприклад:

```
<?php
$a=1;
switch ($a)
{
case 1:
     echo '$a=1';
     break;
case 2:
     echo '$a=2';
     break;
default:
     echo '$a='.$a;
     break;
}
```

Результат виконання: $a=1.

Цикли while

Найпростішим циклом у PHP є цикл while. Різниця між оператором if та оператором while полягає у тому, що у випадку виконання умови оператор if виконує блок коду тільки один раз, а оператор while повторює його до тих пір, поки умова виконується.

Наприклад:

```
<?php
$var=1;
while($var<=5)
{
     echo $var.'<br />';
     $var++;
}
```

Цикли for

Цикл типу while можна записати і у більш компактній формі за допомогою оператора for. Базова структура оператора for має вигляд:

```
for(вираз 1; умова; вираз 2)
     вираз 3;
```

Вираз 1 виконується один раз на початкові циклу. Як правило, він встановлює початкове значення змінної циклу.

Вираз умова перевіряється перед кожною ітерацією. Якщо цей вираз повертає значення false – цикл зупиняється.

Вираз 2 виконується у кінці кожної ітерації. Зазвичай у ньому змінюється значення лічильника.

Вираз 3 виконується один раз під час кожної ітерації. Це, як правило, – блок коду, який містить тіло циклу. Наступний приклад виводить табличку з двох стовпчиків і п’яти рядків.

```
<?php
echo "<table>";
for($distance=50; $distance<=250; $distance+=50)
{
     echo "<tr>\n <td align='right'>$distance</td>\n";
     echo "  <td align=right'>".$distance."</td>\n</tr>\n";
}
echo "</table>";
```

Цикли do…while

Загальні структура оператора do…while має вигляд:

```
do
     вираз;
while(умова 1);
```

Цикл `do…while` відрізняється від циклу `while` тим, що в ньому умова перевіряється укінці. Отже, у циклі `do…while` оператор або блок операторів всередині циклу завжди виконується, принаймі, один раз.

```
<?php
$a=10;
do
{
     echo $a.'<br />';
     $a-=1;
}
while($a>=0);
?>
```

1.2 Контрольні запитання

1. Які типи PHP-дескрипторів існують на даний час?
2. Поясніть суть оператора у PHP.
3. Які особливості використання пробілів?
4. Які типи коментарів використовують при програмуванні на PHP?
5. Яким чином реалізують доступ до змінних форми?
6. Що таке конкатенація рядків та як вона реалізується?
7. Поясніть зміст поняття “ідентифікатор” на PHP.
8. Які типи змінних підтримує PHP?
9. Поясніть суть поняття “змінна змінних”?
10. Яким чиному оголошуються константи на PHP?
11. Поясніть суть поняття “область дії змінних”?
12. Яке призначення посилань?
13. Операції порівняння на PHP.
14. Яким чином реалізують розгалуження у програмі?
15. Які оператори для створення циклів надає PHP?

Масиви у PHP. Багатократне використання коду. Створення функцій.

Мета: набути практичних навичок з використання масивів та багатократного використання програмного коду.

Завдання: дослідити використання масивів, створення функцій користувача та можливості включення у програмний код раніше розроблених файлів.

Обладнання: ЕОМ, програмне забезпечення з відкритим кодом (opensource): PHP8, Apache 2.4.

Теоретичні відомості

Використання масивів у PHP

У більшості мов програмування масиви мають числові індекси, котрі, як правило, починаються з нуля чи одиниці.

PHP дозволяє використовувати як індекси числа або рядки.

Масив створюється наступним чином:

```
$a = array('data1','data2','data3');
$b = array(1,2,3);
```

Відповідно, доступ до елементів масиву, отримують так:

```
echo '$a[0]='.$a[0].'<br />';
echo '$a[1]='.$a[1].'<br />';
echo '$a[2]='.$a[2].'<br />';

echo '$b[0]='.$b[0].'<br />';
echo '$b[1]='.$b[1].'<br />';
echo '$b[2]='.$b[2].'<br />';
```

Якщо у масиві необхідно зберегти зростаючу послідовність чисел, використовують вункцію `range()`. Аналогічно функція утворює послідовність символів:

```
<?
$nombers = range(1,130);
$symbols=range('a','j');
for($i=0; $i<10; $i++)
echo $nombers[$i].'---'.$symbols[$i].'<br />';
```

Даний приклад генерує масив із десяти чисел у діапазоні від 1 до 10 та послідовність символів від “a” до “j”.

Для доступу до елементів масиву зручно використовувати цикл `foreach`, який призначений, власне, для роботи з масивами.

```
<?php
$arr = range(1,20);
foreach ($arr as $current)
     echo $current.'<br />';
```

Наведений код почергово зберігає кожний елемент масиву у змінній `$current` і потім виводить її значення у вікно браузера.

При створенні масивів у наведених прикладах PHP було надано можливість присвоїти кожному елементові масиву індекс по замовчуванню, тобто перший доданий елемент став 0-м, другий – 1-м і так далі. PHP також підтримує масиви, у яких з кожним елементом можна пов'язати (асоціювати) довільний ключ (індекс).

У наступному прикладові символьні назви слугують як ключі, а числа – як значення.

```
<?php
     $arr =
array('the_first'=>1,'the_second'=>2,'the_third'=>3);
     echo $arr['the_first'].'<br />';
     echo $arr['the_second'].'<br />';
     echo $arr['the_third'].'<br />';
```

Створити масив з багатьма елементами можна також, оголосивши масив з одним елементом і додаючи до нього пізніше решту потрібних елементів:

```
<?php
     $arr = array('the_first'=>1);
     $arr['the_second']=2;
     $arr['the_third']=3;
     foreach ($arr as $current)
          echo $current.'<br />';
```

Оскільки в асоціативних масивах індекси не є числами, то для роботи з такими масивами неможливо скористатися лічильником у циклі `for`. У цьому випадкові слід застосувати цикл `foerach` або конструкції `list()` та `each()`.

При роботі з асоціативними масивами цикл `foreach` зазнає незначних змін.

```
<?php
$arr =
array('the_first'=>1,'the_second'=>2,'the_third'=>3);
foreach ($arr as $current)
     echo $current.'<br />';
```

Якщо необхідно отримати доступ до індексів масиву, використовують конструкцію:

```
<?php
$arr =
array('the_first'=>1,'the_second'=>2,'the_third'=>3);
foreach ($arr as $key => $value)
     echo '$arr['.$key.']='.$value.'<br />';
```

Результат:

$arr[the_first]=1
$arr[the_second]=2
$arr[the_third]=3.
Наведений нижче код виводить вміст масиву за допомогою конструкції `each()`:

```
<?php
$arr =
array('the_first'=>1,'the_second'=>2,'the_third'=>3);
while( $element = each($arr))
{
     echo '$arr['.$element['key'].']='.
$element['value'].'<br />';
}
```

У цьому прикладі змінна `$element` – це масив. При викликові функції `each()` вона повертає масив з чотирма значеннями і чотирма індексами. Комірки `key` та `0` містять ключ поточного елементу, `value` та `1` – його значення.

Важливо: при використанні функції `each()` слід пам'ятати, що масив відстежує поточний елемент. Якщо в одному і тому ж сценарії необхідно двічі скористатися значеннями одного і того ж масиву, то потрібно за допомогою функції `reset()` встановити поточний елемент на початок масиву.

Наведемо ряд корисних функцій для роботи з масивами.

`sort($array)` – впорядковує елементи за алфавітом чи порядком зростання;
`asort($array)` – те саме, що і sort(), але для масивів з описовими індексами;
`ksort($array)` – впорядковує ключі у масиві з описовими індексами за зростанням;
`rsort($array)` – впорядковує елементи за алфавітом чи порядком спадання;
`rasort($array)` – те саме, що і sort(), але для масивів з описовими індексами;
`rksort($array)` – впорядковує ключі у масиві з описовими індексами за спаданням.

У PHP відсутня можливість порівняння двох масивів, тому для сортування багатовимірного масиву необхідно створювати деякий ручний метод.

`shuffle($array)` – розташовує елементи масиву випадковим чином;
`array_push($array, $data)` – додає елемент `$data` у кінець масиву `$array;`
`$array = array_reverse($array)` – змінює порядок розташування елементів на зворотній;
`next($array)` або `each($array)` – переміщують показник на елемент вперед на один елемент; `next` – спочатку змінює покажчик, потім – повертає значення, `each` – навпаки;
`end($array)` – встановлює покажчик на елемент у кінець масиву;
`prev($array)` – встановлює покажчик на елемент на одну позицію ближче до початку масиву;
`reset($array)` – встановлює покажчик масиву на його початок.
`sizeof($array)` – підраховує кількість елементів у масиві.

2.1.2 Багатократне використання коду

Одне із завдань, які зазвичай ставлять перед собою розробники програмного забезпечення – повторне використання існуючого програмного коду замість написання нового. Таким чином вдається знизити вартість розробки, підвищити надійність за забезпечити певну уніфікацію.

PHP надає два оператори `require()` та `include()` для забезпечення повторного використання програмного коду.

Оператор `require('шлях до файлу')` слугує для включення повного коду файлу-параметру оператора у тому місці файла реципієнта, в якому знаходиться сам оператор.

При цьому розширення імені файла, який включається, ігнорується, тобто він може бути названий довільним чином. Файл, що включається, повинен мати PHP-дескриптори. Інакше вміст файла буде розглянуто як простий текст.

Оператор `require()` часто застосовується для створення шаблонів сторінок.

Оператор `include()` практично не відрізняється від оператора `require()` за винятком того, що `require()` у випадку невдалого виконання видає повідомлення про непоправну помилку, а `include()` – тільки попередження.

2.1.3 Створення функцій

Функцією називають незалежний модуль коду, який встановлює інтерфейс введення, виконує певну задачу і за необхідності повертає результат.

Імена функцій у PHP не чутливі до регістру.

Оголошення функції створює нову функцію. Оголошення починається ключовим словом `function`, воно присвоює функції ім'я, задає необхідні параметри і містить код, котрий виконується при кожному викликові функції.

Оголошення функції має вигляд:

```
function my_function()
{
     Echo 'викликано функцію';
}
```

Виклик функції реалізується за допомогою оператора `my_function()`.

Вбудовані функції PHP доступні для усіх сценаріїв, а функції користувача – тільки у тому сценарії, у якому вони оголошені. Є сенс додати усі функції, що часто використовуються, в окремий файл і за допомогою оператора `require()` надати доступ до них.

Імена функцій користувача вибираються, враховуючи обмеження:

– функція не може мати ім'я таке саме, як у вже існуючої функції;

– ім'я функції може складатися тільки з букв, цифр та символів підкреслювання;

– ім'я функції не може починатися з цифри.

Для нормального функціонування більшість функцій вимагають передачі у них параметрів. Наприклад (виводиться таблиця з 3 рядків і 1 стовпчика):

```
<?php
function create_table($data)
{
```

```
        echo '<table border=1>';
        reset($data); // вказуємо початок
        $value=current($data);
        while($value)
        {
            echo "<tr><td>$value</td></tr>\n";
            $value = next($data);
        }
        echo '</table>';
    }
    $my_array = array('Рядок 1', 'Рядок 2','Рядок 3');
    create_table($my_array);
```

Як і вбудовані функції, функції користувача можуть мати обов'язкові та необов'язкові параметри. Модифікуємо попередній приклад:

```
    <?php
    function create_table2($data, $border=1, $cellpadding=4,
$cellspacing=4)
    {
        echo "<table border=$border cellpadding=$cellpadding"
                ." cellspacing=$cellspacing>\n";
        reset($data); // вказуємо початок
        $value=current($data);
        while($value)
        {
            echo "<tr><td>$value</td></tr>\n";
            $value = next($data);
        }
        echo '</table>';
    }

    $my_array = array('Рядок 1', 'Рядок 2','Рядок 3');
    create_table2($my_array,3,8,8);
```

Перший параметр функції `create_table2()` як і раніше – обов'язковий. Наступні три параметри – не обов'язкові, оскільки для них визначені значення по замовчуванню. Необов'язкові параметри можна передавати не всі, а тільки деякі з них. Параметри присвоюються зліва-направо.

Слід пам'ятати, що пропуск необов'язкового параметру і задавання наступного після нього параметру не допускається. У наведеному прикладі це означає, що за необхідності задати `$cellpadding` пропустити `$border` не можна.

Існує можливість дізнатися, скільки та які саме параметри передано.

```
    <?php
    function create_table2($data, $border=1, $cellpadding=4,
$cellspacing=4)
    {

        echo "<table border=$border cellpadding=$cellpadding"
                ." cellspacing=$cellspacing>\n";
```

```
        reset($data); // вказуємо початок
        $value=current($data);
        while($value)
        {
             echo "<tr><td>$value</td></tr>\n";
             $value = next($data);
        }
        echo '</table>';
        echo 'Кількість параметрів: ';
        echo func_num_args().'<br />';
        $args = func_get_args();
        foreach ($args as $arg)
             echo $arg.'<br />';
}

$my_array = array('Рядок 1', 'Рядок 2','Рядок 3');
create_table2($my_array,3,8,8);
```

Функція `func_num_args()` визначає, скільки аргументів було передано функції користувача, а `func_get_args()` повертає масив, який містить ці аргументи.

Звичний спосіб передачі параметрів називається передачею за значенням. При цьому зміна значення глобального параметра функції з тіла функції неможлива. Для вирішення проблеми використовується так звана передача за посиланням, яка реалізується шляхом додавання символу амперсанда (`&`) до перед іменем параметра у визначенні функції.

Ключове слово `return` завершує виконання функції і передає виконання наступному операторові після виклику функції. Також функція завершує свою роботу, коли усі оператори виконані.

Найпоширеніша причина використання оператора `return` з метою передчасного завершення функції – умова виникнення помилки. Також `return` використовують, коли функція повинна повертати результат.

PHP підтримує рекурсивні функції. Рекурсивною функцією називають функцію, котра викликає сама себе. Такі функції особливо корисні для переміщення по динамічних структурах даних, таких як зв’язані списки та дерева.

Слід пам’ятати, що рекурсивна функція створює у пам’яті копії самої себе і, відповідно, веде до непродуктивних витрат ресурсів.

2.2 Хід роботи

1. Ознайомитися із теоретичними відомостями (виконується до початку лабораторного заняття).
2. Дослідити функціонування операторів, дія яких розглядається у теоретичних відомостях.
3. Виконати практичне завдання, поставлене викладачем (зазвичай суть практичного завдання полягає у розробці невеликого сценарію, функціональність якого визначається темою, що розглядається, ти стислими термінами розробки під час лабораторного заняття).
4. Зробити висновки.
5. Захистити роботу та відповісти на контрольні запитання.

Контрольні запитання

1. Яким чином створюють масиви. Ініціалізація елементів масивів.
2. Яким чином організувати доступ до елементів масиву за допомогою циклів на PHP.
3. Асоціативні масиви. Призначення. Доступ до елементів.
4. Сортування елементів масивів.
5. Призначення операторів `require()` та `include()`.
6. Яким чином створюють функції на PHP?

Лабораторна робота №3
Збереження і відновлення даних. Робота з файлами
Тривалість: 2 акад. години

Мета: набути практичних навичок роботи з файлами.

Завдання: дослідити функціонування операторів, пов'язаних із роботою з файлами.

Обладнання: ЕОМ, програмне забезпечення з відкритим кодом (opensource): PHP8, Apache 2.4.

3.1 Теоретичні відомості

Існують два способи збереження даних: у двовимірних файлах та базах даних. Двовимірний файл може мати множину форматів, але у загальному випадкові вважають, що це – текстовий файл.

Запис та читання файлів в PHP практично ідентичні виконанню цих операцій на мові С.

Для відкриття файлу використовують функцію `fopen()`. При цьому необхідно вказати режим файлу. Режим файлу надає операційній системі механізм вибору способу обробки запитів на доступ, що надходять від інших користувачів, а також метод перевірки, чи має користувач-програміст доступ та права на роботу з конкретним файлом.

Відкриваючи файл, слід пам'ятати наступне:

– файл можна відкрити тільки для читання, тільки для запису або для читання та запису;

– при виконанні запису у файл, можливо, виникне потреба перезаписувати вміст файлу чи додавати нові дані у кінець файлу. Також може виникнути потреба завершити виконання програми, не перезаписуючи файл, якщо він існує;

– при спробі додати запис у файл в системі, яка розрізняє бінарні та текстові файли, можливо буде необхідно сказати тип файлу.

Відкривання файлу здійснюється за допомогою оператора `fopen()`. Обов'язкові його параметри наступні:

```
resource fopen ( string $filename , string $mode)
```

`$filename – ім'я файлу, що відкривається. Адресу можна задати в абсолютній або відносній формі. Остання вказує на основу дерева документів WEB-сервера. В абсолютній формі для UNIX-систем адреса основи дерева документів починається з символу "/", для Windows-систем – це, зазвичай, "C:\". Використання абсолютної форми небезпечне, оскільки адреса основи дерева документів може змінюватися.`

`Доступ до кореня дерева каталогів можна отримати, скориставшись функцією getenv():`

```
<?php
$ docroot = getenv("DOCUMENT_ROOT") ;
echo $ docroot;
?>
```

`Також можна використати значення $_SERVER['DOCUMENT_ROOT']:`

```
<?php
     $docroot = $_SERVER['DOCUMENT_ROOT'];
     echo $docroot;
?>
```

Другий параметр функції fopen() – $mode – це режим файлу. Режими файлів наведено у таблиці 3.1.

Таблиця 3.1 – Режими файлів для функції fopen()

Режим	Значення
r	Режим читання. Відкриває файл для читання, починаючи з початку
r+	Режим читання. Відкриває файл для читання і запису, починаючи з початку
w	Режим запису. Відкриває файл для запису, починаючи з початку. Якщо файл існує – його вміст видаляється. Якщо не існує – робиться спроба створити його
w+	Режим запису. Відкриває файл для запису і читання, починаючи з початку. Якщо файл існує – його вміст видаляється. Якщо не існує – робиться спроба створити його
x	Режим обережного запису. Відкриває файл для запису, починаючи з початку файлу. Якщо файл вже існує, він не відкривається, fopen() повертає значення false, а PHP генерує попередження
x+	Режим обережного запису. Відкриває файл для запису і читання, починаючи з початку файлу. Якщо файл вже існує, він не відкривається, fopen() повертає значення false, а PHP генерує попередження
a	Режим додавання. Відкриває файл тільки для додавання (запису), починаючи з кінця існуючого вмісту, якщо такий існує. Якщо файл не існує – намагається його створити
a+	Режим додавання. Відкриває файл тільки для додавання (запису) і читання, починаючи з кінця існуючого вмісту, якщо такий існує. Якщо файл не існує – намагається його створити
b	Бінарний режим. Використовується у поєднанні з одним із інших режимів. Користувач може скористатися ним, коли файлова система розрізняє бінарні та текстові файли. Windows розрізняє їх, UNIX – ні. Розробники PHP рекомендують завжди використовувати цей режим. Режим використовується по замовчуванню
t	Текстовий режим. Використовується у поєднанні з одним із інших режимів. Режим актуальний тільки для Windows-систем. Застосовування його не рекомендується за виключенням випадків, коли здійснюється перенесення існуючого коду з опцією b

Типова помилка при відкриванні файлу – відсутність дозволу на читання його чи на запис у нього (як правило, виникає у UNIX-системах).

Якщо виклик функції fopen() завершується невдачею, вона повертає false. Обробку помилок можна зробити зручнішою для користувача шляхом усунення повідомлення про помилку від PHP і обробити його самостійно:

```
<?php
     $docroot= $_SERVER['DOCUMENT_ROOT'];
     @$fp=fopen($docroot."/test.txt","r");
     if(!$fp)
     {
          echo "помилка відкривання файлу";
          exit;
     }
     fclose($fp);
```

Закривають файл за допомогою функції fclose().

Запис у файл в PHP виконується порівняно просто. Для цього використовують функції fwrite() чи fputs(). fputs() – псевдонім fwrite().

```
fwrite($fp, $outputstring [, int length]);
```

Довжину рядка для запису length (необов'язковий параметр) у PHP можна отримати за допомогою функції strlen($string).

Альтернатива fwrite() – функція file_put_contents(), її прототип з обов'язковими параметрами:

```
int file_put_contents(string filename, string data).
```

Ця функція записує рядок, що передається у параметрі data, у файл з іменем filename без необхідності відкривання та закривання файлу функціями fopen() та fclose().

Функція доступна у PHP 5. Її парна функція – file_get_contents().

Для перевірки ознаки досягнення кінця файлу можна скористатися функцією feof().

```
<?php
     $docroot= $_SERVER['DOCUMENT_ROOT'];
     @$fp=fopen($docroot."/test.txt","r");
     if(!$fp)
     {
          echo "помилка відкривання файлу";
          exit;
     }
     while(!feof($fp))
     {
          $str=fgets($fp);
          echo $str.'<br />';
     }
     fclose($fp);
```

У даному прикладі для читання з файлу використовується функція `fgets()`. Необов'язковий параметр – кількість символів, які читаються з файлу. Причому Максимальна довжина рядка, що обробляється, дорівнює значенню параметру – 1 байт.

Різновид функції `fgets()` –

```
array fgetcvs(resource fp, int length [,string delimiter
[,string enclosure]]).
```

Ця функція розбиває рядки файлів за умови використання деякого символа-роздільника (наприклад, табуляції чи коми). Отримані при цьому рядки додаються до масиву.

Параметр `length` повинен бути більшим за довжину у символах найдовшого рядка файлу, з якого відбувається читання.

Параметр `enclosure` використовується для опису символів, між якими поміщається кожне поле у рядку. По замовчування це – подвійні лапки ".

Функція `readfile()` відкриває файл, відображає його вміст у вікні браузера і закриває файл. Функція повертає кількість байт, прочитаних з файлу

```
int readfile(string filename[, int use_include_path[,
resource context]]).
```

Аналогічний результат можна отримати за допомогою функції `fpassthru()`. Для цього необхідно відкрити файл (`fopen()`) і передати вказівник на файл у `fpassthru()`:

```
<?php
     $docroot= $_SERVER['DOCUMENT_ROOT'];
     $fp=fopen($docroot."/test.txt","rb");
     fpassthru($fp);
?>
```

Існує функція, котра дозволяє перетворити рядки файлу у масив – `file()`.

```
<?php
     $docroot= $_SERVER['DOCUMENT_ROOT'];
     $filearray=file($docroot."/test.txt","r");
     foreach ($filearray as $current)
          echo $current.'<br />';
```

Функція не є безпечною для бінарних файлів.

Починаючи з PHP 4.3.0 можна використати функцію `file_get_contents()`. Дія функції аналогічна до `readfile()` за винятком того, що замість виведення у вікно браузера функція повертає вміст файлу у вигляді рядка. `file_get_contents()` безпечна для бінарних файлів

```
<?php
     $docroot = $_SERVER['DOCUMENT_ROOT'];
     $str = file_get_contents($docroot."/test.txt");
     echo $str;
```

Можливість по символьного читання файлу забезпечується функцією fgetc():

```
<?php
     $docroot = $_SERVER['DOCUMENT_ROOT'];
     $fp = fopen($docroot."/test.txt","r");
     while(!feof($fp))
     {
          $char = fgetc($fp);
          if(!feof($fp))
          echo ($char=="\n" ? '<br />' : $char);
     }
```

Побічний ефект використання fgetc() замість fgets() полягає у тому, що fgetc() повертає символ EOF, fgets() – ні. Після читання символа доводиться знову перевіряти feof(), оскільки EOF відображатися у вікні браузера не повинен.

Функція fread() надає можливість зчитування з файлу довільної кількості символів:

```
string fread(resource fp, int length).
```

Функція прочитає з файлу length байт або усі байти до кінця файлу залежно від того, яка подія відбудеться раніше.

Інші корисні функції:

– file_exists() – перевіряє, чи існує файл:

```
<?php
     $docroot = $_SERVER['DOCUMENT_ROOT'];
     if(file_exists($docroot."/test.txt"))
          echo 'файл існує';
     else
          echo 'файл не існує';
```

– filesize() –визначає розмір файлу у байтах:

```
<?php
     $docroot = $_SERVER['DOCUMENT_ROOT'];
     echo filesize($docroot."/test.txt");
```

– unlink() – видалення файлу:

```
<?php
     $docroot = $_SERVER['DOCUMENT_ROOT'];
     unlink($docroot."/test.txt");
```

– rewind() – переміщує вказівник на початок файлу;

– ftell() – повідомляє у байтах позицію вказівника відносно початку файлу;

– int fseek(resource fp, int offset [, int whence]) – використовується для встановлення вказівника файла у деяку конкретну точку всередині файлу. Необов'язковий параметр whence (звідки) по замовчуванню набуває значення SEEK_SET, котре фактично означає початок файлу. Інші можливі його значення: SEEK_CUR (поточне значення вказівника файлу) та SEEK_END (кінець файлу):

```
<?php
     $docroot = $_SERVER['DOCUMENT_ROOT'];
     $fp = fopen($docroot."/test.txt","r");
     echo 'поточне значення вказівника: '.ftell($fp).'<br
/>';
     fseek($fp, 2);
     echo 'поточне значення вказівника: '.ftell($fp).'<br
/>';
     rewind($fp);
     echo 'поточне значення вказівника: '.ftell($fp);
```

3.2 Хід роботи

1. Ознайомитися із теоретичними відомостями (виконується до початку лабораторного заняття).
2. Дослідити функціонування операторів, дія яких розглядається у теоретичних відомостях.
3. Виконати практичне завдання, поставлене викладачем (зазвичай суть практичного завдання полягає у розробці невеликого сценарію, функціональність якого визначається темою, що розглядається, ти стислими термінами розробки під час лабораторного заняття).
4. Зробити висновки.
5. Захистити роботу та відповісти на контрольні запитання.

Контрольні запитання

1. Яким чинок здійснюють відкривання файлу? Режими відкривання.
2. Як організовують запис до файлу?
3. Яким чином організовують зчитування з файла?
4. Назвіть оператори для читання символів та рядків з файлу. Як їх використовують?
5. Яким чином отримують доступ до основи дерева документів Web-сервера і для чого здійснюється ця операція?

Лабораторна робота №4
Використання бази даних MySQL з Web за допомогою PHP
Тривалість: 2 акад. години

Мета: набути практичних навичок роботи з базою даних MySQL з Web за допомогою PHP.

Завдання: дослідити використання операторів для роботи з базою даних.

Обладнання: ЕОМ, програмне забезпечення з відкритим кодом (opensource): PHP8, Apache 2.4.

4.1 Теоретичні відомості

Архітектура баз даних для Web визначає такі кроки програмного забезпечення:

– Web-браузер користувача надсилає HTTP-запит певній Web-сторінці;

– Web-сервер приймає запит на PHP-файл, отримує файл і передає його на обробку механізмові PHP;

– механізм PHP проводить синтаксичний аналіз сценарію. Сценарій містить команду під'єднання до бази та виконання запиту. PHP відкриває з'єднання з MySQL-сервером і надсилає йому відповідний запит;

– сервер MySQL приймає запит бази даних, обробляє його і надсилає результат назад механізмові PHP;

– механізм PHP завершує виконання сценарію, що зазвичай пов'язане з форматуванням результатів запиту у вигляді HTML, після чого повертає результати у HTML-форматі Web-серверові;

– Web-сервер надсилає браузерові користувача HTML-сторінку з результатами виконання його запиту.

Отже, довільний сценарій, який забезпечує доступ до бази даних з Web, повинен виконати ряд базових кроків:

1. Перевірити і відфільтрувати запити, що надходять від користувача.
2. Встановити з'єднання з потрібною базою даних.
3. Реалізувати запит до бази даних.
4. Отримати результати.
5. Представити результати користувачеві.

Нехай таблиця створена за допомогою запиту:

```
CREATE TABLE `tab` (
`id` INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY ,
`name` VARCHAR( 255 ) NOT NULL ,
`power` INT NOT NULL
) ENGINE = innodb CHARACTER SET cp1251 COLLATE
cp1251_ukrainian_ci;
```

4.1.1 Перевірка і фільтрація вхідних даних

Перша за все, слід видалити зайві пробіли, котрі користувач міг випадково ввести до- чи після критерію пошуку. Це здійснюється за допомогою функції `trim()`, що застосовується до `$searchitem`:

```
$searchtherm=trim($searchtherm);
```

Наступний етап – перевірка того, чи користувач вказав критерій пошуку. Вона виконується тільки після видалення зайвих пробілів.

```
if(!$searchtherm)
     {
          echo "Ви не вказали критерій пошуку";
          exit;
     }
```

Перед збереженням у базу слід виключити керуючі символі (літералізувати їх) у даних, введених користувачем.

```
if(!get_magic_quotes_gpc())
{
     $searchtherm=addslashes($searchtherm);
}
```

Функція `get_magic_quotes_gpc()` перевіряє, чи здійснюється автоматичне взяття керуючих символів у лапки.

Функція `htmlspecialchars()` використовується для кодування символів, які мають спеціальне значення у HTML (`&`, <, >, ").

```
<?php
     $searchtherm="AT&T";
     $searchtherm=htmlspecialchars($searchtherm);
     echo $searchtherm;
?>
```

Результат виконання скрипта: AT&T.

4.1.2 Встановлення з'єднання

```
$user="root";
$pass=="";
$database="test";
$host=localhost;

@$db =mysql_connect($host,$user,$pass);

if (!$db) {
    echo 'Не вдалося встановити з'єднання з  '.$host;
    exit;
}
```

4.1.4 Вибір бази даних

```
mysql_query ("SET NAMES `cp1251`");
@mysql_select_db($database) or die( "Не вдалося вибрати базу даних");
```

4.1.5 Виконання запиту до бази даних

Для здійснення запиту необхідно скористатися функцією `mysql_query()`.

$result = mysql_query("SELECT * FROM tab WHERE id>0 && id<3").

У цьому випадкові здійснюється пошук елементів таблиці tab бази даних test, що відповідають критерію: `id` запису знаходиться у межах (`id>0 && id<3`).

Запит можна передати і таким чином:

```
$query = ("SELECT * FROM tab WHERE id>0 && id<3";
$result = mysql_query($query).
```

4.1.6 Отримання результатів запиту

Існує група функцій, що дозволяють отримати потрібні фрагменти з ідентифікатора результату запиту:

```
$num_results = mysql_num_rows($result);
echo $num_results;
```

За допомогою функції `mysql_num_rows($result)` отримують кількість записів, які відповідають критерієві пошуку.

Функція `mysql_fetch_row($result)` забезпечує послідовний доступ до результатів пошуку. При виконанні функції дістається рядок із результуючого набору і повертається у вигляді масиву, у якому кожний ключ є іменем атрибуту, а кожне значення – відповідним значенням запису:

```
$row = mysql_fetch_row($result);
echo $row[0];
```

Для видалення результатів екранування керуючих символів використовують функцію `stripslashes()`, наприклад:

```
$res = stripslashes($row[2]).
```

Очистка масиву результатів виконання запиту реалізується за допомогою функції

```
mysql_free_result($result).
```

Функція `mysql_close($db)` закриває з'єднання з базою.

4.1.7 Додавання інформації до бази

Додавання нових елементів до бази схожа на отримання елементів з бази. Необхідно виконати ті ж самі дії, як і у випадку пошуку результатів запиту, при цьому замість оператора `SELECT` використовують оператор `INSERT`.

Наведений нижче скрипт додає до таблиці запис, потім знаходить кількість записів у таблиці і виводить це значення у браузер:

```
<?php
     error_reporting(E_ALL^E_NOTICE);

     $user="root";
     $pass=="";
     $database="test";
     $host=localhost;

     $db =mysql_connect($host,$user,$pass);

     @mysql_query ("SET NAMES `cp1251`");

     if (!$db) {
         echo 'Could not connect to '.$host;
         exit;
     }

     @mysql_select_db($database) or die( "Unable to select
database");
     $name="qwerty";
     $mark=5;
     $query = "INSERT INTO tab VALUES ('','$name',$mark)";
     mysql_query($query);
     $result = mysql_query("SELECT * FROM tab");
     $num_results = mysql_num_rows($result);
     echo $num_results;
     mysql_free_result($result);
     mysql_close(($db));
```

4.2 Хід роботи

1. Ознайомитися із теоретичними відомостями (виконується до початку лабораторного заняття).
2. Дослідити функціонування операторів, дія яких розглядається у теоретичних відомостях.
3. Виконати практичне завдання, поставлене викладачем (зазвичай суть практичного завдання полягає у розробці невеликого сценарію, функціональність якого визначається темою, що розглядається, ти стислими термінами розробки під час лабораторного заняття).
4. Зробити висновки.
5. Захистити роботу та відповісти на контрольні запитання.

Контрольні запитання

1. Які кроки програмного забезпечення передбачає архітектура баз даних для Web-сервера?
2. Яким чином здійснюють перевірку і фільтрацію вхідних даних при роботі з базами даних?
3. Встановлення з’єднання з базою даних.
4. Яким чином здійснюється вибір бази даних?
5. Виконання запиту до бази даних.
6. Отримання результатів виконання запиту та їхня обробка.

Лабораторна робота №5
Об’єктно-орієнтоване програмування у PHP
Тривалість: 2 акад. години

Мета: набути практичних навичок використання об’єктно-орієнтованого підходу до програмування.

Завдання: вивчити та дослідити функціонування класів, використання модифікаторів доступу, застосування інтерфейсів.

Обладнання: ЕОМ, програмне забезпечення з відкритим кодом (opensource): PHP8, Apache 2.4.

5.1 Теоретичні відомості

В контексті об’єктно-орієнтованого програмного забезпечення об’єктом може бути довільний елемент або концепція.

Основна перевага – здатність підтримання і стимулювання інкапсуляції (encapsulation). По суті, доступ до даних всередині об’єкта можливий через операції об’єкта, які називаються інтерфейсом об’єкта.

5.1.1 Структура класу

Мінімальна варіант визначення класу має вигляд:

```
class classname
{
}
```

Класи повинні мати атрибути і операції (поля та функції-члени класу). Атрибути задають за допомогою ключового слова `var`:

```
class classname
{
     var $attribute1;
     var $attribute2;
}
```

Операції створюють за рахунок оголошення функцій всередині визначення класу:

```
class classname
{
     function operation1();
     {
     }
     function operation2($param1, param2)
     {
     }
}
```

5.1.2 Конструктори

Більшість класів володіє спеціальним типом операції, що отримала назву конструктора. Конструктор викликається у тих випадках, коли потрібно створити об’єкт;

зазвичай він виконує так задачі ініціалізації, як встановлення дефолтних значень атрибутів чи створення інших об'єктів, потрібних для даного об'єкта.

Конструктор оголошується, як і інші операції, але має спеціальне ім'я `__construct()` (важливо: PHP8).

```
class classname
{
      function  __construct($param)
      {
      echo "Конструктор викликано з параметром $param <br />";
      }
}
```

5.1.3 Деструктори

Деструктор – функція, протилежна до конструктора. Механізм деструктора з'явився у PHP8. Деструктор дозволяє виконати певні дії перед знищенням класу. Деструктор може мати ім'я `__destruct()`.

5.1.4 Створення екземпляру класу

Екземпляр класу створюється за допомогою ключового слова `new`:

```
<?php
class classname
{
      function  __construct($param)
      {
            echo "Конструктор викликано з параметром: $param
<br />";
      }
}

$a = new classname("перший");
```

5.1.5 Використання атрибутів класу

Всередині класу програміст отримує доступ до спеціального вказівника з іменем `$this`, який дозволяє здійснити звертання до змінної із операції всередині класу:

```
class classname
{
      var $attribute;
      function  operation($param)
      {
            $this->attribute = $param;
            echo operation($param);
      }
}
```

У наступному прикладі продемонстровано зовнішнє звертання звертання до змінної класу:

```
<?php
class classname
{
     var $attribute;
     }
$a = new classname();
$a->attribute = "value1";
echo $a->attribute;
```

За допомогою функцій `__get()` та `__set` уникають прямого доступу до змінних класу:

```
<?php
class classname
{
     var $attribute;
     function __set($name,$value)
     {
          $this->$name=$value;
     }
     function __get($name)
     {
          return $this->$name;
     }
}
$a = new classname();
$a->__set("attribute",10);
echo $a->__get("attribute");
```

5.1.6 Керування доступом за допомогою модифікаторів `private` та `public`

Модифікатор доступу `public` (загальнодоступний) встановлюється по замовчуванню і означає відкритий доступ до атрибуту чи методу, що еквівалентно відсутності модифікатора доступу. Такі елементи доступні як зсередини, так і зовні класу.

Модифікатор доступу `private` означає, що позначений ним елемент доступний тільки зсередини класу. Щодо методів – модифікатор вказує, які з них призначені для службового використання класу. Приватні методи не успадковуються.

Модифікатор `protected` аналогічний за призначенням модифікаторові `private`, але атрибути та методи, позначені ним, успадковуються.

При цьому не використовується ключове слово `var`.

5.1.7 Реалізація успадкування у PHP

Якщо клас повинен бути субкласом іншого класу – використовують ключове слово `extends`.

Оскільки клас class2 є розширенням класу class1, то змінній класу class2 доступні методи і атрибути обох класів.

Успадкування працює тільки в одному напрямкові. Субклас отримує батьківські властивості, але батьківський клас не отримує властивостей свого дочірнього класу. Це значить що рядки `$a = new class1; $a->operation2()` у наступному прикладі були би помилковими.

```
<?php
class class1
{
     var $attribute1;
     function operation1()
     {
          echo $this->attribute1.'<br />';
     }
}
class class2 extends class1
{
     var $attribute2;
     function operation2()
     {
          echo $this->attribute2;
     }
}
$b = new class2;
$b->attribute1=1;
$b->attribute2=2;
$b->operation1();
$b->operation2();
```

Керуванням видимістю при успадкуванні здійснюється за допомогою операторів `protected` та `private`.

5.1.8 Перекриття

Допускається, а часом корисно здійснювати повторне оголошення раніше оголошеного атрибуту чи методу іншого класу у класі, що успадковує даний.

Уникнути повторного оголошення можна за допомогою ключового слова `final` при оголошенні методу чи атрибуту.

Можна також заборонити створення субкласів на основі заданого класу, помістивши ключове слово `final` перед його оголошенням.

5.1.9 Множинне успадкування

PHP не підтримує множинного успадкування. Це означає, що кожний клас може успадковувати характеристики тільки одного батьківського класу. Кількість дочірніх класів, що мають спільний батьківський, не обмежується.

5.2 Хід роботи

1. Ознайомитися із теоретичними відомостями (виконується до початку лабораторного заняття).
2. Дослідити функціонування операторів, дія яких розглядається у теоретичних відомостях.
3. Виконати практичне завдання, поставлене викладачем (зазвичай суть практичного завдання полягає у розробці невеликого сценарію, функціональність якого визначається темою, що розглядається, ти стислими термінами розробки під час лабораторного заняття).
4. Зробити висновки.
5. Захистити роботу та відповісти на контрольні запитання.

Контрольні запитання

1. Призначення об’єктно-орієнтованого стилю програмування.
2. Мінімальна варіант визначення класу. Атрибути та методи класу.
3. Конструктор класу та деструктор.
4. Створення екземпляру класу.
5. Використання атрибутів класу.
6. Розкрийте суть та призначення методу керування доступом за допомогою модифікаторів `private` та `public`
7. Поясніть суть поняття перекриття та множинного успадкування. Чи підтримує PHP множинне успадкування.

Лабораторна робота №6
Задача аутентифікації. Керування сеансами
Тривалість: 2 акад. години

Мета: набути практичних навичок щодо вирішення задачі аутентифікації та керування сеансами.

Завдання: дослідити використання операторів для організації аутентифікації користувача та керування сеансами.

Обладнання: ЕОМ, програмне забезпечення з відкритим кодом (opensource): PHP8, Apache 2.4.

6.1 Теоретичні відомості

6.1.1 Механізм доступу зі збереженням даних аутентифікації у сценарії

Реалізувати найпростіший контроль доступу нескладно. Наступний приклад реалізує простий механізм аутентифікації, однак він містить кілька суттєвих проблем. Назвемо даний файл `secret.php`.

```
<?php
@$user = $_POST['user'];
@$password = $_POST['password'];
if(empty($user) || empty($password))
{
?>
     <h1>Ви не вказали свої логін та пароль</h1>
     <form action="secret.php" method="POST">
     <input type="text" name="user" value="login">
     <input type="text" name="password" value="password">
     <input type="submit" value="Увійти">
     </form>
<?php
}
else if($user=='user' && $password=='pass')
echo 'Комбінація вірна';
?>
```

Наведений сценарій:
– підтримує тільки одне жорстко закодоване ім’я користувача та пароль;
– зберігає пароль у вигляді відкритого тексту;
– захищає тільки одну сторінку;
– передає пароль у вигляді відкритого тексту.

Зберігання паролів на сервері в окремому файлі дозволить легко написати програму для додавання та видалення користувачів, а також для зміни паролів.

6.1.2 Використання баз даних для вирішення задачі аутентифікації

Всередині сценарію та іншого файлу існує обмеження на кількість користувачів, яких можна обслужити без серйозного зниження загальної продуктивності сценарію. Якщо планується зберігати та проводить пошук поміж більш як 100 користувачами, слід надати перевагу базі даних.

Використання бази даних для збереження імен та паролів не надто ускладнить сценарій, натомість дозволить швидко проводити аутентифікацію багатьох користувачів.

6.1.3 Шифрування паролів

Незалежно від того, де зберігаються паролі – у базі даних чи у файлі – зберігання їх у відкритому вигляді викликає невиправданий ризик.

Односторонній алгоритм хешування забезпечує додатковий захист при незначних затратах.

Таким чином, замість коду

```
if($user=='user' && $password=='pass')
{
     // пароль співпадає
}
```

можна скористатися кодом

```
if($user=='user' &&
     ф-я хешування($password)
=='5baa61e4c9b93f3f0682250b6cf8331b7ee68fd8')
{
     // пароль співпадає
}.
```

Достатньо знати, чи співпадає введений пароль з тим, для якого застосовувалася `sha1()`.

Якщо для збереження даних аутентифікації вибрано базу MySQL, можна скористатися PHP чи функцією MySQL .

Слід пам’ятати, що хеш-функції повертають дані фіксованого розміру.

6.1.4 Захист кількох сторінок

Захист більш ніж одної сторінки за допомогою сценаріїв складніший, оскільки е HTTP відсутній механізм підтримки станів, тобто не існує автоматичного зв’язку чи асоціації між послідовними запитами, що надходять від одного і того ж користувача.

Найпростіший спосіб захисту кількох сторінок – використання механізму контролю доступу, що надає Web-сервер.

Коли користувач відкриває кілька сторінок сайту, запитувати його кожного разу логін та пароль не допускається. Інформацію про введені одного разу дані можна було би передавати як гіперлінк, проте такі дані керуються, відповідно кожен зможе їх продивитися.

Розв’язати проблему можна за допомогою двох механізмів – базової HTTP-аутентифікації та підтримки сеансів. Базова аутентифікація дозволяє розв’язати проблему кешування, проте сервер надсилає пароль Web-браузеру у кожному запиті. Керування сеансами вирішує обидві проблеми.

Розглянемо приклад.

```
<?php
   if (!isset($_SERVER['PHP_AUTH_USER'] ))
   {
     header('WWW-Authenticate: Basic realm="My Realm"');
     header('HTTP/1.0 401 Unauthorized');
     echo 'Ви не зареєструвалися';
     exit;
   }
   else
   {
     echo "<p>Вітаємо, {$_SERVER['PHP_AUTH_USER']}.</p>";
     echo "<p>Ви ввели пароль
{$_SERVER['PHP_AUTH_PW']}.</p>";
     if($_SERVER['PHP_AUTH_USER']=='login' &&
$_SERVER['PHP_AUTH_PW']=='pass')
     echo 'все гаразд';
     else echo 'Пароль або логін не співпадає';
   }
?>
```

Якщо користувач не передав дані аутентифікації, він отримує повідомлення про те, що він не зареєструвався. Також отримає повідомлення про те, чи були передані коректні параметри аутентифікації.

6.2 Хід роботи

1. Ознайомитися із теоретичними відомостями (виконується до початку лабораторного заняття).
2. Дослідити функціонування операторів, дія яких розглядається у теоретичних відомостях.
3. Виконати практичне завдання, поставлене викладачем (зазвичай суть практичного завдання полягає у розробці невеликого сценарію, функціональність якого визначається темою, що розглядається, ти стислими термінами розробки під час лабораторного заняття).
4. Зробити висновки.
5. Захистити роботу та відповісти на контрольні запитання.

Контрольні запитання

1. Розкрийте, яким чином функціонує механізм доступу зі збереженням даних аутентифікації у сценарії.
2. Яким чином та у яких випадках слід використовувати бази даних для вирішення задачі аутентифікації?
3. Яким чином здійснюється шифрування паролів
4. Яким чином здійснюється захист кількох сторінок на PHP?

Лабораторна робота №7
Маніпулювання рядками та регулярні вирази
Тривалість: 2 акад. години

Мета: набути практичних навичок роботи регулярними виразами та використання рядкових операцій.

Завдання: дослідити використання операторів для роботи з регулярними виразами та рядкових операторів.

Обладнання: ЕОМ, програмне забезпечення з відкритим кодом (opensource): PHP8, Apache 2.4.

7.1 Теоретичні відомості

Основні операції, що здійснюються над рядками, наступні:
– форматування рядків;
– об'єднання та розділення рядків;
– порівняння;
– співставлення і заміна підрядків за допомогою функцій обробки рядків;
– використання регулярних виразів.

7.1.1 Форматування рядків

Доволі часто рядки, що вводить користувач, необхідно очищати від службових символів перед тим, як їх можна буде використати за призначенням.

Перший крок – видалення зайвих пробілів. Для цього використовують три функції

$name = trim($name);

Функція видаляє зайві пробіли на початку та у кінці рядка і повертає рядок як результат. Видаляються: символи нового рядка (\n), повернення каретки (\r), символи горизонтальної (\t) та вертикальної (\0B) табуляції, кінця рядка (\0) та звичайні пробіли.

Необов'язковий параметр дозволяє вказати інші символи, що видалятимуться.

```
<?php
     $name = " Hello,  Jack, text   ";
     echo strlen($name).'<br />';
     $name = trim($name);
     echo strlen($name).'<br />';
     $name = trim($name, "Jack, Hello");
     echo strlen($name).'<br />';
```

Функції `rtrim()` `ltrim()` аналогічній розглянутій за виключенням того, що видаляють символи справа та зліва у рядку відповідно.

Функція `nl2br()` приймає рядок як параметр і замінює у ньому усі символи нового рядка XHTML-дескриптором `<br />`:

```
<?php
     echo nl2br("довільний\nнабір\n слів");
```

Для виведення інформації у вікно браузера використовують також функцію `print()`, котра виконує ту ж саму операцію, що і `echo`, однак крім того повертає ще результат виконання `true` або `false`.

Функції `printf()` та `sprintf()` виконують форматоване виведення . `Printf()` виводить інформацію у вікно браузера, а `sprintf()` – повертає сформований рядок як результат.

Існує можливість змінювати регістр рядків. Для цього використовують наступні функції:

– `strtoupper($string)` – перетворює символи рядка у символи верхнього регістру;

– `strtolower($string)` – перетворює символи рядка у символи нижнього регістру;

– `ucfirst($strinf)` – перетворює перший символ рядка у символ з верхнім регістром, якщо він був буквеним знаком;

– `ucwords($string)` – перетворює у символ верхнього регістру перший символ кожного слова у рядкові, який починаться з буквеного знаку.

З метою зберігання у базі даних рядків, що містять символи лапок (одинарних та подвійних – "", '), зворотної косої риски – \ та символ NULL слід відмінити інтерпретацію цих символів як керуючих. Цього досягають шляхом екранування зворотною косою рискою, яка ставиться перед екранованим символом. Наприклад: \\, \", \'.

У PHP існують дві функції, призначені для відміни дії керуючих символів. Перед записом до бази даних рядок слід від форматувати за допомогою функції `addslashes()`, котра додає косі риски. Як і інші рядкові функції, дана – приймає рядок як параметр і повертає рядок як результат.

По замовчуванню у PHP увімкнута директива `magic_quotes_gpc` (`GET`, `POST`, `cookie`), яка автоматично додає так звані "магічні лапки". Перевірити, чи увімкнена ця директива, можна за допомогою функції `get_magic_quotes()`. Вона повертає значення `true` або `false`.

Функція `stripslashes()` видаляє косі риски з даних користувача перед їхнім відображенням у браузері.

7.1.2 Об'єднання та роз'єднання рядків

Розділити рядок на окремі його складові (слова) можна за допомогою функції

```
array explode(string розділювач, string рядок [, int limit].
```

Тут `limit` – необов'язковий параметр, який обмежує число результатів.

Обернена функція – `implode(string розділювач, array результати).`

```
<?php
     $res = explode(',','test,for,test');
     foreach ($res as $current) echo $current.'<br />';
```

Функція `strtok()` на відміну від `explode()` дозволяє отримувати лексеми рядка, по одній за один виклик. Прототип функції:

```
string strtok(string рядок, string розділювач).
```

Для отримання першої лексеми функції передають рядок та розділювач, для отримання наступних лексем достатньо передавати тільки розділювач.

Наприклад:

```
<?php
     $text = "1 2 3 4 5   6 7 0 9";
     echo $token=strtok($text," ").'<br />';
     while($token!=' ')
     {
          $token=strtok(" ");
          echo $token.'<br />';
     }
```

Функція `substr()` дає можливість отримати доступ до підрядка між заданими позиціями у вихідному рядкові. Прототип функції:

```
string substr(string рядок, int початок [, int довжина])
```

7.1.3 Порівняння рядків

```
int strcmp(string str1, string str2)
```

У випадку, якщо рядки однакові, функція, повертає значення 0. Якщо у лексикографічному порядкові рядок `str1` більше за `str2` (тобто `str1` йде за `str2`), функція повертає значення, більше за 0 і, якщо навпаки – менше за 0. Ця функція чутлива до регістру.

Функція `strcasecmp()` відрізняється від `strcmp()` тим, що вона нечутлива до регістру.

Функція `strnatcmp()` (natural string comparing – природнє порівняння рядків) і її нечутливий до регістру аналог порівнюють рядки у “природному порядкові”. Порівняйте два приклади:

```
<?php
     $str1='2';
     $str2='10';
     echo strcmp($str1,$str2);
```

```
<?php
     $str1='2';
     $str2='10';
     echo strnatcmp($str1,$str2);
```

7.1.4 Перевірка довжини рядка за допомогою функції `strlen()`

Передавши функції параметр-рядок, отримуємо довжину рядка у символах.

7.1.5 Пошук та заміна підрядків

```
string strstr(string haystack, string needle)
```

Якщо функція у рядкові `haystack` знаходить `needle`, вона повертає частину рядка `haystack`, починаючи з `needle`.

Варіанти функції: `stristr()` – нечутливий до регістру; `strrchr()` – у випадку кілької входжень `needle` повертає частину рядка, починаючи з останнього входження.

Функції `strops()` та `strrpos()` діють аналогічно до `strstr()` і відрізняються від неї тим, що повертають числову позицію рядка `needle` всередині `haystack`:

```
string strops(string haystack, string needle
[,int offset ])
```

Необов'язковий параметр `offset` використовується для вказання позиції всередині рядка `haystack`, з якої повинен починатися пошук.

Функціональні можливості пошуку та заміни винятково корисні при роботі з рядками.

Для заміни найчастіше використовують функцію `str_replace()`, її прототип:

```
mixed str_replace(mixed needle,
mixed new_needle, mixed haystack, [,int &count]);
```

Ця функція замінює всі екземпляри рядка `needle` на `new_needle` у рядкові `haystack` . Необов'язковий параметр `count` містить кількість виконаних замін.

Функція substr_replace() використовується для пошуку та заміни конкретного підрядка на основі його позиції у рядкові. Її прототип:

```
string substr_replace(string needle,
string replacement, int start [, int length]);
```

7.1.5 Регулярні вирази

Регулярні вирази надають спосіб опису шаблону з використанням текстових фрагментів.

На додачу до точного співставлення шаблонів можна використовувати спеціальні символи для вказування мета значень.

Пошук підрядків – основне застовування регулярних виразів:

```
<?php
     $text="{рядок у фігурних дужках}";
     echo $text.'<br />';
     preg_match_all("|{(.+)}|U",$text, $out,
PREG_PATTERN_ORDER);
     $res=$out[0][0];
     $pattern="|{|U";
     $repl='';
     $res= preg_replace($pattern, $repl, $res);
     $pattern="|}|U";
     $res= preg_replace($pattern, $repl, $res);
     echo $res;
```

У даному прикладі регулярний вираз знаходить підрядок між фігурними дужками і видаляє дужки.

7.2 Хід роботи

1. Ознайомитися із теоретичними відомостями (виконується до початку лабораторного заняття).
2. Дослідити функціонування операторів, дія яких розглядається у теоретичних відомостях.
3. Виконати практичне завдання, поставлене викладачем (зазвичай суть практичного завдання полягає у розробці невеликого сценарію, функціональність якого визначається темою, що розглядається, ти стислими термінами розробки під час лабораторного заняття).
4. Зробити висновки.
5. Захистити роботу та відповісти на контрольні запитання.

Контрольні запитання

1. Поясніть суть поняття форматування рядків.
2. Яким чином здійснюють форматоване виведення?
3. Як реалізувати об’єднання та роз’єднання рядків?
4. Які оператори використовують для порівняння рядків?
5. Поясніть, яким чином реалізують пошук та заміну підрядків.
6. Поясніть суть поняття “регулярний вираз”. Використання регулярних виразів.

Лабораторна робота №8
Використання функцій для роботи з мережею та протоколами
Тривалість: 2 акад. години

Мета: набути практичних навичок використання функцій для роботи з мережею та протоколами.

Завдання: освоїти технологію “вирізання з екрану”, використання FTP-сервера.

Обладнання: ЕОМ, програмне забезпечення з відкритим кодом (opensource): PHP8, Apache 2.4.

8.1 Теоретичні відомості

8.1.1 Технологія “вирізання з екрану”

Дана технологія названа так, оскільки користувач отримує інформацію, першочергово призначену для відображення на екрані, і дістає з неї частини, призначені для представлення за допомогою нового інтерфейсу.

Нехай існує файл http://localhost/test.html:

```
<html>
<body>
{1234}
</body>
</html>
```

та скрипт

```
<?php
     $theurl = "http://localhost/test.html";
     if(!($text=file_get_contents($theurl)))
     {
          echo "неможливо під'єднатися";
          exit;
     }
     preg_match_all("|{(.+)}|U",$text, $out,
PREG_PATTERN_ORDER);
     $res=$out[0][0];
     $pattern="|{|U";
     $repl='';
     $res= preg_replace($pattern, $repl, $res);
     $pattern="|}|U";
     $res= preg_replace($pattern, $repl, $res);
     echo $res;
```

8.1.2 Під’єднання до віддаленого FTP-сервера

```
$conn = fpt_connect(“host”);
if(!conn)
{
     echo “помилка з’єднання”;
     exit;
```

```
}
```

8.1.3 Реєстрація на FTP-сервері

Реєстрація на FTP-сервері реалізується за допомогою функції `ftp_login()`:

```
@result = ftp_login($conn, $user, $password);
if(!result)
{
     echo "Помилка: користувач $user не зареєстрований<br
/>";
     ftp_quit($conn);
     exit;
}
```

8.1.4 Завантаження файлу із сервера

```
echo 'читання файлу з сервера…<br />';
$fp = fopen($localfile, 'w');
if(!success = ftp_fget($conn, $fp, $remotefile,
FTP_BINARY))
{
     echo 'Помилка: файл не може бути завантажений';
     ftp_quit($conn);
     exit;
}
fclose($fp);
echo 'Файл успішно завантажено';
```

8.1.5 Закриття з'єднання

```
ftp_quit($conn);
```

8.1.6 Завантаження файлів на сервер

```
int ftp_fput (int ftp_connection, string remotefile_path,
int fp, int mode)

int ftp_put (int ftp_connection, string remotefile_path,
string localfile_path, int mode)
```

8.1.7 Уникнення таймаутів

```
set_time_limit($seconds);
```

8.2 Хід роботи

1. Ознайомитися із теоретичними відомостями (виконується до початку лабораторного заняття).
2. Дослідити функціонування операторів, дія яких розглядається у теоретичних відомостях.
3. Виконати практичне завдання, поставлене викладачем (зазвичай суть практичного завдання полягає у розробці невеликого сценарію, функціональність якого визначається темою, що розглядається, ти стислими термінами розробки під час лабораторного заняття).
4. Зробити висновки.
5. Захистити роботу та відповісти на контрольні запитання.

Контрольні запитання

1. Яким чином реалізують технологію “вирізання з екрану”?
2. Як здійснити під’єднання до віддаленого FTP-сервера? Яке його призначенн?
3. Покажіть на прикладі, як провести реєстрація на FTP-сервері.
4. Розкрийте механізм завантаження файлу із сервера.
5. Розкрийте механізм завантаження файлів на сервер.

Лабораторна робота №9
Робота з датою та часом
Тривалість: 2 акад. години

Мета: набути практичних навичок роботи даними часу.

Завдання: дослідити використання операторів для роботи з даними часу.

Обладнання: ЕОМ, програмне забезпечення з відкритим кодом (opensource): PHP8, Apache 2.4.

9.1 Теоретичні відомості

Функція `date()` розпізнає два параметри, один із яких є обов'язковим, другий – ні. Перший параметр задає рядок формату (таблиця 9.1), інший – мітку часу Unix.

Таблиця 9.1 – Коди форматування PHP-функції `date()`

Символ у рядкові *format*	Опис	Приклад значення, що повертається
`a`	Ante meridiem або Post meridiem у нижньому регістрі	*am* або *pm*
`A`	Ante meridiem або Post meridiem у верхньому регістрі	*AM* або *PM*
`B`	Час у стандарті Swatch Internet	Від *000* до *999*
`c`	Дата у форматі ISO 8601 (додано у PHP 5)	2004-02-12T15:19:21+00:00
`d`	День місяця, 2 цифри с ведучими нулями	Від *01* до *31*
`D`	Скорочена назва дня тижня, 3 символа	Від *Mon* до *Sun*
`F`	Повна назва месяця, наприклад January або March	Від *January* до *December*
`g`	Години у 12-годинному форматі без ведучих нулів	Від *1* до *12*
`G`	Години в 24- годинному форматі без ведущих нулей	Від *0* до *23*
`h`	Години в 12- годинному форматі з ведучими нулями	Від *01* до *12*
`H`	Години в 24- годинному форматі з ведучими нулями	Від *00* до *23*
`i`	Хвилини с ведучими нулями	Від *00* до *59*
`I`	Ознака літнього часу	*1*, якщо дата відповідає літньому часові, інакше *0*
`j`	День місяця без ведучих нулів	Від *1* до *31*
`l` (прописна 'L')	Повна назва дня тижня	Від *Sunday* до *Saturday*
`L`	Ознака високосного року	*1*, если рік високосний, інакшее *0*.
`m`	Порядковий номер місяця с ведучими нулями	Від *01* до *12*
`M`	Скорочена назва місяця, 3 символи	Від *Jan* до *Dec*
`n`	Порядковий номер місяця без ведучих нулів	Від *1* до *12*
`O`	Різниця з датою за Грінвічем у годинах	Наприклад: *+0200*

Символ у рядкові *format*	Опис	Приклад значення, що повертається
`r`	Дата у форматі RFC 2822	Наприклад: *Thu, 21 Dec 2000 16:01:07 +0200*
`s`	Секунди з ведучими нулями	Від *00* до *59*
`S`	Англійський суфікс порядкового числівника дня месяця, 2 символи	*st*, *nd*, *rd* или *th*. Застосовується разом з *j*
`t`	Кількість днів у місяці	Від *28* до *31*
`T`	Часова зона на сервері	Приклади: *EST*, *MDT* ...
`U`	Кількість секунд, що пройшли від початку Епохи Unix (The Unix Epoch, 1 січня 1970, 00:00:00 GMT)	
`w`	Порядковий номер дня тижня	Від *0* (неділя) до *6* (субота)
`W`	Порядковий номер тижня року за ISO-8601, перший день тижня - понеділок	Наприклад: *42* (42-й тиждень року)
`Y`	Порядковий номер року, 4 цифри	Приклади: *1999*, *2003*
`y`	Номер року, 2 цифри	Приклади: *99*, *03*
`z`	Порядковий номер дня в році (нумерація з 0)	Від *0* до *365*
`Z`	Зміщення часової зони у секундах. Для часових зон західніше UTC це від'ємне число число, східніше UTC - додатнє	Від *-43200* до *43200*

Інша корисна функція – `getdate()`:

```
array getdate([int timestamp]).
```

Вона отримує як необов'язковий параметр мітку часу `timestamp` і повертає асоціативний масив, що містить компоненти цієї дати та часу.

```
<?php
     $date = getdate();
     foreach ($date as $key => $value)
     echo '$arr['.$key.']='.$value.'<br />';
```

Результат виконання скрипта (залежиться від поточного часу), наприклад, може бути такий:

arr[seconds]=47
$arr[minutes]=52
$arr[hours]=9
$arr[mday]=15
$arr[wday]=2
$arr[mon]=4
$arr[year]=2008
$arr[yday]=105
$arr[weekday]=Tuesday
$arr[month]=April
$arr[0]=1208242367.

Для перевірки правильності дат можна скористатися функцією

```
int checkdate(int month, int day, int year).
```

9.2 Хід роботи

1. Ознайомитися із теоретичними відомостями (виконується до початку лабораторного заняття).
2. Дослідити функціонування операторів, дія яких розглядається у теоретичних відомостях.
3. Виконати практичне завдання, поставлене викладачем (зазвичай суть практичного завдання полягає у розробці невеликого сценарію, функціональність якого визначається темою, що розглядається, ти стислими термінами розробки під час лабораторного заняття).
4. Зробити висновки.
5. Захистити роботу та відповісти на контрольні запитання.

Контрольні запитання

1. Яке призначення функції `date()`?
2. Назвіть деякі коди форматування PHP-функції `date()`. Яке їхнє призначення?
3. Поясніть суть поняття Unix-мітки.
4. Яким чином можна перевірити валідність дати?
5. Як організовують порівняння дат?