МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

МЕТОДИЧНІ ВКАЗІВКИ

до виконання лабораторних робіт

з курсу «Сучасні технології розробки Інтернет-застосунків»
для студентів спеціальності
«Прикладна та комп'ютерна лінгвістика»

Харків 2019

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

МЕТОДИЧНІ ВКАЗІВКИ

до виконання лабораторних робіт

з курсу «Сучасні технології розробки Інтернет-застосунків»
для студентів спеціальності
«Прикладна та комп'ютерна лінгвістика»

Затверджено
редакційно-видавничою
радою університету,
протокол № 2 від 17.05.2019 р.

Харків
НТУ «ХПІ»
2019

Методичні вказівки до виконання лабораторних робіт з курсу «Сучасні технології розробки Інтернет-застосунків» для студентів спеціальності «Прикладна та комп'ютерна лінгвістика» / уклад. Хайрова Н. Ф., Петрасова С. В. – Харків : НТУ «ХПІ», 2019. – 53 с.

Укладачі: Н. Ф. Хайрова
С. В. Петрасова

Рецензент Н. В. Шаронова

Кафедра інтелектуальних комп'ютерних систем

ВСТУП

Методичні вказівки містять теоретичний матеріал та завдання до лабораторних робіт з курсу «Сучасні технології розробки Інтернет-застосунків» для студентів спеціальності «Прикладна та комп'ютерна лінгвістика».

Навчально-методичне видання призначено для набуття необхідної методичної допомоги при виконанні лабораторних робіт.

Методичний матеріал охоплює широке коло питань, пов'язаних з використанням серверної мови PHP та технологій MySQL для розробки web-додатків.

Лабораторні роботи присвячені вивченню основ синтаксису мови PHP, роботі з масивами, функціями та регулярними виразами PHP, створенню форм та способів їхньої обробки, використанню функцій для роботи з файлами. Окремі лабораторні роботи спрямовані на закріплення навичок роботи з MySQL, зокрема обробки запитів MySQL за допомогою PHP. Остання лабораторна робота пов'язана зі створенням та застосуванням класів в PHP. Кожна лабораторна робота супроводжується наочними прикладами.

Повністю оформлений звіт з лабораторної роботи повинен містити: титульний лист, мету, хід виконання роботи, програмний код та інтерфейс програми, а також висновки за результатами роботи.

Дане видання призначене для студентів старших курсів спеціальності «Прикладна та комп'ютерна лінгвістика», а також студентів технічних і гуманітарних спеціальностей, які вивчають технології розробки web-застосунків.

Лабораторна робота 1

ОСНОВИ МОВИ PHP

1.1. Логічні оператори PHP

Оператори порівняння використовуються для порівняння значень. У таблиці 1.1 подано оператори порівняння мови PHP.

Таблиця 1.1 – Оператори порівняння

Оператор	Опис
= =	дорівнює
! =	не дорівнює
>	більше
<	менше
> =	більше або дорівнює
<=	менше або дорівнює
= = =	еквівалентність, рівні значення і типи змінних ($a = = = $b)
! = =	нееквівалентність, змінні не еквівалентні ($a! = = $b)

Логічні оператори дозволяють визначати стан умов. Залежно від умови змінної в сценарії можуть відбуватися різні дії. Логічні оператори широко використовуються в керуючих структурах PHP. У таблиці 1.2 подано логічні оператори мови PHP.

Таблиця 1.2 – Логічні оператори

Оператор	Опис
&&	AND (ТА)
||	OR (АБО)
! NOT	NOT (НЕ)
xor	Вилучальна або одна зі змінних істинна. Випадок, коли обидві змінні істинні, виключається.

1.2. Умовні оператори PHP

1.2.1. Оператор if

if – оператор, що використовується для виконання блоку коду, коли виконується умова (true).

```
if (умова)
{
// код, що виконується
}
```

Рядки коду оператора if беруться у фігурні дужки ({}). Ці дужки визначають початок (дужка, що відкривається {) і кінець (дужка, що закривається }) оператора if. Наступний приклад демонструє використання оператора if.

Приклад 1.1

```
<?php
$number = 5;
if($number <= 10) {echo "Число менше або дорівнює 10.";}
?>
```

Число менше або дорівнює 10.

if ... else – оператор, що використовується для виконання блоку коду, коли умова виконується (true), або для виконання іншого блоку коду, коли умова не виконується (false).

Приклад 1.2

```
<?php
$number = 15;
if($number <= 10) {echo "Число менше або дорівнює 10.";}
else {echo "Число більше 10";}
?>
```

Число більше 10.

Третій тип умовного оператора є структурою *elseif*. Оператор elseif є комбінацією if та else. Подібно else він розширює оператор if, щоб виконати інший оператор, якщо умовний вираз вихідного if оцінюється як false. Однак на відміну від else він буде виконувати цей альтернативний вираз, тільки якщо умовний вираз в elseif оцінюється як true. В одному

операторі if може бути кілька структур elseif. Перший вираз elseif (якщо такий є), який оцінюється як true, буде виконано.

Приклад 1.3

```
<?php
$number = 15;
if($number <10) {echo "Число менше 10.";}
elseif($number == 10) {echo "Число дорівнює 10.";}
else {echo "Число більше 10.";}
?>
Число більше 10.
```

У цьому прикладі числове значення змінної $number порівнюється з 10. Спершу оператор if перевіряє, що $number менше 10. Якщо цей оператор виконується (true), виводиться повідомлення "Число менше 10". Потім оператор elseif використовується для перевірки, що $number дорівнює 10. Якщо цей оператор оцінюється як true, виводиться повідомлення "Число дорівнює 10". Оператор elseif виконується, тільки якщо оператор if повертає false. Якщо оператори if та elseif повертають false, виконується оператор else і виводиться повідомлення "Число більше 10". В той час як оператор elseif дозволяє перевірити тільки одну умову, структуру if можна застосувати для перевірки множини умов.

Оператори if можна використовувати "поодинці" або як частину оператора if ... else або if ... elseif ... else.

1.2.2. Оператор switch

На додаток до операторів if, PHP включає четвертий тип умовного оператора. Оператор *switch* дуже схожий або є альтернативою для команд if ... else. Оператор switch перевіряє умову. Результат цієї перевірки визначає, який *case* виконується. Switch використовується зазвичай, коли шукають точний (рівність) результат, замість умови більше або менше. При перевірці діапазону значень повинен застосовуватися оператор if.

```
<?php
switch(вираз)
{
case "значення1":
// код, який буде виконано, якщо вираз = значення1;
break;
```

case "значення2":

// код, який буде виконано, якщо вираз = значення2;

break;

default:

// код, який буде виконано, якщо вираз не дорівнює ні значення1, ні значення2;

}

?>

Подібно оператору if рядки коду в операторі switch беруться у фігурні дужки. Ці дужки визначають початок і кінець оператора switch. Наступний приклад демонструє використання оператора switch.

Приклад 1.4

```
<?php
$number = 25;
switch($number)
{
case 40:
echo "Значення \$number дорівнює 40";
break;
case 25:
echo "Значення \$number дорівнює 25";
break;
default:
echo "Значення \$number відмінно від 25 і 40";
}
?>
```

Оператор switch може включати множину операторів case. У попередньому прикладі показані два оператори case. Створюється змінна $number, якій присвоюється значення 25. Оператор switch використовується для порівняння значення $number з іншими значеннями. Порівняний вираз (в даному випадку $number) береться в дужки відразу після оператора switch. Потім викликається послідовність операторів case для порівняння виразу з іншими значеннями. Ці значення містяться відразу після оператора case. За значенням, яке порівнюється з виразом, слідує двокрапка (:). Оператори case аналогічні конструкціям if та elseif. Якщо

значення оператора case буде true, то виконується код, пов'язаний з цим оператором, і оператор break. Оператор break приводить до завершення оператора switch. Решта операторів case перевірятися не буде. В кінці оператор switch містить інструкцію default. Вона аналогічна оператору else. Якщо жоден з операторів case не буде виконано (не матиме значення true), виконується оператор default:

1) перевіряється умова switch і знаходиться значення ($number == 25);

2) значення умови передається по черзі операторам case;

3) якщо це значення збігається зі значенням case, виконується код цього блоку. Оператор break призводить до завершення оператора switch, решту операторів case не перевіряє;

4) якщо значення, яке перевіряється, не збігається ні з одним зі значень case, виконується розділ default;

5) в наведеному вище прикладі, оскільки вираз "значення $number" дорівнює 25, виконується другий оператор case, і у вікні браузера виводиться текст "значення $number дорівнює 25".

1.3. Оператори циклу while, do while і for, foreach

У програмуванні часто необхідно повторити один і той же блок коду кілька разів. Це можна реалізувати за допомогою операторів циклу. Мова PHP містить кілька типів операторів циклу.

Оператор *while* циклічно повторює блок коду, поки зазначена умова має значення true. Іншими словами, оператор while буде виконувати блок коду, якщо і поки умова буде істинною.

```
while(умова)
{
// код, що виконується;
}
```

Код в циклі while буде повторно виконуватися, поки умова на початку циклу має значення true. Блок коду, пов'язаний з оператором while, завжди береться в фігурні дужки.

Приклад 1.5

```
<?php
$number = 5;
while($number >= 2)
```

```
{
echo $number. "<br>";
$number -= 1;
}
?>
```

Висновок, який створюється прикладом циклу:

5
4
3
2

Оператор *do ... while* повторює циклічно блок коду, поки певна умова набуває значення true. Іншими словами, оператор do ... while буде виконувати блок коду, якщо і поки умова буде виконуватися (тобто оцінюватися як true).

Цикл do ... while аналогічний за своєю природою циклу while. Ключова відмінність полягає в тому, що тіло циклу do ... while буде обов'язково виконано як мінімум один раз. Це пов'язано з тим, що оператор умови оцінюється в кінці оператора циклу після виконання тіла циклу.

```
do
{
// код, що виконується;
}
while(умова);
```

Виконання коду всередині циклу do ... while буде повторюватися, поки умова в кінці циклу оцінюватиметься як true. Блок коду, пов'язаний з оператором do ... while, завжди розміщується всередині фігурних дужок.

Приклад 1.6

```
$number = 5;
do
{
echo $number. "<br>";
$number –= 1;
}
while($number >= 2);
```

Висновок, який створюється прикладом циклу:

5

4

3

2

Оператор циклу *for* використовується, коли відомо, скільки разів необхідно виконати оператор або послідовність операторів. У зв'язку з цим цикл for називають точним циклом.

for(ініціалізація; умова; крок циклу)

{

// код, що виконується;

}

Оператор циклу for має три параметри. Перший параметр використовується для ініціалізації змінних, другий містить умову, а третій включає в себе приріст, необхідний для реалізації циклу. Блок коду, пов'язаний з оператором for, завжди береться в фігурні дужки.

Приклад 1.7 демонструє цикл for для 4-кратного виведення повідомлення "Ласкаво просимо в світ PHP".

Приклад 1.7

```
<?php
for($counter = 1; $counter < 5; $counter ++)
{
echo "Ласкаво просимо в світ PHP! <br>";
}
?>
```

Цикли for використовуються також в якості зручного способу виконання ітерацій за значеннями масиву.

Конструкція *foreach* є варіацією циклу for і застосовується для ітерацій на масивах. Команда foreach сама виконує обхід і читання всіх елементів масиву. Вона дозволяє не тримати постійно в пам'яті той факт, що індексація масивів починається з нуля і ніколи не виходить за межі масиву, що дозволяє уникнути поширеної логічної помилки. Існують дві різні версії циклу foreach.

Перша версія foreach використовує такий синтаксис:

foreach($array as $value)

{

оператор

}

Перший тип циклу foreach використовується для ітерацій по масиву, який позначено як *$array*. Коли foreach починає виконання, внутрішній покажчик масиву автоматично встановлюється на перший елемент. Під час кожної ітерації циклу поточне значення масиву присвоюється змінній *$value*, і лічильник циклу збільшується на одиницю. Цикл продовжується, поки foreach не досягне останнього елемента або верхньої межі заданого масиву. Під час кожної ітерації значення змінної $value можна використовувати будь-яким способом, але початкове значення масиву не змінюється. Щоб змінити реальне значення масиву, необхідно додати символ "&". Будь-які зміни, зроблені в &value, будуть присвоєні елементу масиву з поточним індексом.

Приклад 1.8 демонструє, як цикл foreach застосовується для ітерацій за значеннями масиву.

Приклад 1.8

```
<?php
$my_array = array('red', 'green', 'blue');
echo "Різні кольори включають:";
foreach($my_array as $value)
{
$colours = $value. " ";
echo $colours;
}
?>
```

Друга форма циклу забезпечує такі ж функції, що і перша, але додатково присвоює на кожному кроці ітерації індекс поточного елемента масиву або ключ змінної *$key*, яку можна використовувати в блоці виконання. Аналогічно першій формі команди foreach на початку виконання внутрішній покажчик масиву автоматично встановлюється на перший елемент.

```
foreach($array as $key => $value)
{
оператор
}
```

Приклад 1.9

```
<?php
$names = array("Іван", "Петро", "Семен");
foreach($names as $val)
{
echo "Привіт, $val <br>";
// виводимо всім вітання
}
foreach($names as $k => $val)
{
// крім вітання, виводимо номера в списку, тобто ключі
echo "Привіт, $val! Ти в списку під номером $k <br> ";
}
?>
```

1.4. Операції з масивами

Масив – тип даних, з якими повинні бути визначені операції. Масиви можна складати і порівнювати.

Складають масиви за допомогою стандартного оператора "+", який об'єднує масиви. Якщо є два масиви, $a і $b, то результатом їх складання (об'єднання) буде масив $c, що складається з елементів $a, до яких праворуч дописані елементи масиву $b.

Якщо зустрічаються ключі, що збігаються, то в масив з результатом включається елемент ключів, що співпадають, з першого масиву, тобто з $a.

Приклад 1.10

```
<?php
$a = array("i" => "Інформатика", "м" => "Математика");
$b = array("i" => "Історія", "м" => "Біологія", "ф" => "Фізика");
$c = $a + $b;
$d = $b + $a;
```

```
print_r ($c);
/ * Отримаємо: array([і] => Інформатика [м] => Математика [ф] => Фізика) * /
print_r($d);
// Отримаємо: array([і] => Історія [м] => Біологія [ф] => Фізика)
?>
```

Порівнювати масиви можна, перевіряючи їхню рівність чи нерівність або еквівалентність чи нееквівалентність. Рівність масивів – це збіг всіх пар ключ-значення елементів масивів. Еквівалентність – це рівність значень і ключів елементів і запис елементів обох масивів в одному і тому ж порядку. Рівність значень в PHP позначається символом «= =», а еквівалентність – символом «= = =».

Приклад 1.11

```
<?php
$a = array("і" => "Інформатика", "м" => "Математика");
$b = array("м" => "Математика", "і" => "Інформатика");
if($a == $b) echo "Масиви рівні та";
else
echo "Масиви НЕ рівні та";
if($a === $b)
echo "еквівалентні";
else
echo "НЕ еквівалентні";
// отримаємо "Масиви рівні та НЕ еквівалентні"
?>
```

1.5. Використання функцій при роботі з масивами

Функція *count()* обчислює кількість елементів в змінній. Якщо застосувати її до будь-якої змінної, вона поверне 1. Виняток становить змінна типу NULL – count(NULL) є 0. Крім того, застосовуючи цю функцію до багатовимірного масиву, щоб отримати число його елементів, потрібно використовувати додатковий параметр COUNT_RECURSIVE.

Функція *in_array()* дозволяє встановити, чи міститься в заданому масиві шукане значення: *in_array("шукане значення", "масив"[, "Обмеження на тип"])*. Якщо третій аргумент заданий як true, то в масиві

потрібно знайти елемент, що співпадає з шуканим не тільки за значенням, а й за типом. Якщо шукане значення – рядок, то порівняння чутливе до регістру.

Приклад 1.12

```
<?php
$langs = array("Lisp", "Python", "Java", "PHP", "Perl");
if(in_array("PHP", $ langs, true))
echo "Треба б вивчити PHP <br>";
// виводимо повідомлення "Треба б вивчити PHP"
if(in_array("php", $langs))
echo "Треба б вивчити php <br>";
// нічого не виводимо, оскільки в масиві є рядок "PHP", а не "php"
?>
```

Функція *array_search()* – ще одна функція для пошуку значення в масиві. На відміну від in_array() в результаті роботи array_search() повертає значення ключа, якщо елемент знайдений, і хибність – в іншому випадку.

Синтаксис у цих функцій однаковий:

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

Порівняння рядків чутливе до регістру, а якщо вказано додатковий аргумент, то порівнюються ще й типи значень.

Приклад 1.13

```
<?php
$langs = array("", "Lisp", "Python", "Java", "PHP", "Perl");
if(! array_search("PHP", $langs))
echo "Треба б вивчити PHP <br>";
else
{
$k = array_search("PHP", $langs);
echo "PHP я вивчила $k-м";
}
?>
```

Якщо шуканих елементів в масиві декілька, в такому випадку функція array_search() поверне ключ першого зі знайдених елементів. Щоб отримати ключі всіх елементів, потрібно скористатися функцією array_keys().

Функція *array_keys()* вибирає всі ключі масиву. Але у неї є додатковий аргумент, за допомогою якого можна отримати список ключів елементів з конкретним значенням:

array_keys("масив"[, "значення для пошуку"]).

Функція array_keys() повертає як рядкові, так і числові ключі масиву, організовуючи всі значення у вигляді нового масиву з числовими індексами.

Приклад 1.14

```
<?php
$langs = array("Lisp", "Python", "Java", "PHP", "Perl", "Lisp");
$lisp_keys = array_keys($langs, "Lisp");
echo "Lisp входить в масив". count($lisp_keys). "рази: <br>";
foreach($lisp_keys as $val)
{
echo "під номером $val <br>";
}
?>
```

Для отримання всіх значень масиву існує функція *array_values(масив)*. Усі значення переданого їй масиву записуються в новий масив, проіндексований цілими числами, тобто всі ключі масиву губляться, залишаються лише значення.

Функція *array_unique(масив)* повертає новий масив, в якому елементи, що повторюються, фігурують в одному екземплярі. У такий спосіб замість кількох однакових значень та їхніх ключів буде отримано одне значення.

Який у нього буде ключ? Як з декількох ключів однакових елементів вибирається той, який буде збережено в новому масиві?

Відбувається наступне. Всі елементи масиву перетворюються в рядки і сортуються. Потім обробник запам’ятовує перший ключ для кожного значення, а інші ключі ігнорує.

Приклад 1.15

```
<?php
$langs = array("Lisp", "Java", "Python", "Java", "PHP", "Perl", "Lisp");
print_r(array_unique($langs));
?>
```

Функція *sort()* має наступний синтаксис: *sort(масив[, прапори])* і сортує масив за зростанням. Ця функція видаляє всі існуючі в масиві ключі, замінюючи їх числовими індексами, що відповідають новому порядку елементів. У разі успішного завершення роботи вона повертає true, інакше – false.

Як додатковий аргумент може використовуватися одна з наступних констант:

- SORT_REGULAR – порівнювати елементи масиву звичайним чином;
- SORT_NUMERIC – порівнювати елементи масиву як числа;
- SORT_STRING – порівнювати елементи масиву як рядки.

Якщо потрібно зберігати індекси елементів масиву після сортування, то потрібно використовувати функцію *asort(масив[, прапори])*.

Якщо необхідно впорядкувати масив в зворотному порядку, тобто від найбільшого значення до найменшого, то можна задіяти функцію *rsort(масив[, прапори])*.

Якщо при цьому потрібно ще й зберегти значення ключів, то слід використовувати функцію *arsort(масив[, прапори])*.

Синтаксис у цих функцій абсолютно такий самий, як у функції sort(). Відповідно і значення прапорів можуть бути такими самими, як у sort(): SORT_REGULAR, SORT_NUMERIC, SORT_STRING.

Приклад 1.16

```
<?php
$books = array("Пушкін" => "Руслан і Людмила", "Толстой" => "Війна і мир", "Лермонтов" => "Герой нашого часу");
asort($books);
// сортуємо масив, зберігаючи значення ключів
print_r($books);
echo "<br>";
```

```
rsort($books);
// сортуємо масив в зворотному порядку, ключі будуть замінені
print_r($books);
?>
```

Функції *ksort()* і *krsort()* сортують масив значень ключів. Синтаксис цих функцій аналогічний синтаксису функції sort().

Приклад 1.17

```
<?php
$books = array("Пушкін" => "Руслан і Людмила", "Толстой" => "Війна і мир", "Лермонтов" => "Герой нашого часу");
ksort($books);
// сортуємо масив, зберігаючи значення ключів
print_r($books);
?>
```

Функція *array_slice()* виділяє з масиву будь-який піднабір. Її синтаксис такий: *array_slice(масив, номер_елемента[, довжина])*.

Ця функція виділяє підмасив, починаючи з елемента, номер якого заданий параметром *номер_елемента*. Позитивний *номер_елемента* вказує на порядковий номер елемента щодо початку масиву, негативний – на номер елемента з кінця масиву.

Приклад 1.18

```
<?php
$arr = array(1,2,3,4,5);
$sub_arr = array_slice($arr, 2);
print_r($sub_arr);
/* виводимо array ([0] => 3 [1] => 4 [2] => 5), тобто підмасив, що складається з елементів 3, 4, 5 */
$sub_arr = array_slice($arr, –2);
print_r($sub_arr);
// виведе array([0] => 4 [1] => 5),  тобто підмасив з елементів 4, 5
?>
```

Якщо задати параметр *довжина* при використанні array_slice(), то буде виділено підмасив, що має рівно стільки елементів, скільки задано цим параметром. Довжину можна вказувати і негативну. В цьому випадку інтерпретатор видалить з кінця масиву число елементів, рівне модулю параметра *довжина*.

Приклад 1.19

```
<?php
$arr = array(1, 2, 3, 4, 5);
$sub_arr = array_slice($arr, 2, 2);
// містить масив з елементів 3, 4
$sub = array_slice($arr, –3, 2);
// теж містить масив з елементів 3, 4
$sub1 = array_slice($arr, 0, –1);
// містить масив з елементів 1, 2, 3, 4
$sub2 = array_slice($ arr, –4, –2);
// містить масив з елементів 2, 3
?>
```

Функція *array_chunk()* розбиває масив на декілька підмасивів заданої довжини: *array_chunk(масив, розмір[, зберігати_ключі])*.

В результаті роботи array_chunk() повертає багатовимірний масив, елементи якого являють собою отримані підмасиви. Якщо задати параметр *зберігати ключі* як true, то при розбитті будуть збережені ключі вихідного масиву. В іншому випадку ключі елементів замінюються на числові індекси, які починаються з нуля.

Приклад 1.20

```
<?php
$persons = array("Іванов", "Петров", "Сидорова", "Зайцева", "Волкова");
$triples = array_chunk($persons, 3);
// ділимо масив на підмасиви по три елементи
foreach($triples as $k => $table)
{
// виводимо отримані трійки
echo "За столиком номер $k сидять: <ul>";
```

```
foreach($table as $pers)
echo "<li> $pers";
echo "</ul>";
}
?>
```

Функція *array_sum(масив)* використовується для обчислення суми всіх елементів масиву.

Завдання до лабораторної роботи 1

1. Створити асоціативний масив, що складається не менше ніж з 10 елементів.

2. Для створеного масиву вивести на сторінку список елементів і ключів.

3. Виводити на сторінку результати всіх наступних перетворень:

- порахувати кількість елементів в масиві;
- відсортувати масив за ключами;
- відсортувати масив в зворотному порядку значень елементів;
- визначити кількість входжень в масив певного елемента;
- створити новий масив, що містить всі значення ключів;
- створити додатковий масив, що містить два елементи, подібних елементам первинного масиву, і скласти масиви;
- розбити отриманий масив на два масиви однакового розміру (або такі, що відрізняються на одиницю).

Лабораторна робота 2

РОБОТА З ФУНКЦІЯМИ В PHP

2.1. Робота з багатовимірними масивами

Масив, елементами якого є масиви, називається багатовимірним. Кожен набір ключів і значень являє собою вимір. В багатовимірних масивах для кожного виміру є свій набір ключів і значень.

Щоб отримати доступ до другого виміру використовується друга пара квадратних дужок, в яких вказується другий ключ. Якщо у масиві більше двох вимірів, вказуються ключі для кожного з них. Якщо звернутися до елемента $object['Телефонна книга'], буде отримано масив значень.

PHP допускає в асоціативних масивах скорочену форму запису, в якій елемент масиву присвоюється змінній, ім'я якої збігається зі значенням ключа. При цьому імена ключів повинні бути записані латинськими літерами, без пробілів, що відповідає вимогам, які пред'являються до імен змінних. Для цього використовується функція *extract()*, єдиним параметром якої є масив.

Якщо значення ключа збіглося з ім'ям якоїсь змінної, то значення цієї змінної буде заміщено значенням елемента масиву. Щоб цього не сталося, функція extract() може автоматично додавати символ підкреслення в початок імен змінних, запобігаючи можливому заміщенню змінної, що вже використовувалася. Символом підкреслення в створюваних іменах змінних автоматично відокремлюються ім'я ключа і префікс:

extract($ім'я_масиву, EXTR_PREFIX_ALL, "префікс").

Функція *compact()* діє протилежно функції extract(). Вона приймає у вигляді окремих параметрів змінні, масиви або їхню комбінацію і створює асоціативний масив, ключами якого слугують імена змінних, а значеннями елементів – значення змінних.

Для налагодження програми, яка працює з масивами, часто використовується вбудована функція *var_dump()*, яка дозволяє вивести весь масив за одне звернення: *var_dump($ім'я_масиву)*.

Дуже часто при роботі з масивами використовується функція *list()*, яка є не функцією, а мовною конструкцією. List() використовується для того, щоб присвоїти списку змінних значення за одну операцію.

Приклад 2.1

```
<?php
list($day, $month, $year) = full_age("07", "08", "1974");
echo "Вам $year років, $month місяців і $day днів";
?>
```

Для присвоєння змінним значень елементів масиву конструкція list() використовується в наступний спосіб.

Приклад 2.2

```
<?php
$arr = array("first", "second");
list($a, $b) = $arr;
// змінній $a присвоюється перше значення масиву, $b – друге
echo $a, " ", $b;
// виводимо рядок "first second"
?>
```

2.2. Математичні функції

У таблиці 2.1 подано прості математичні функції.

Таблиця 2.1 – Математичні функції

Функція	Опис
floor()	Приймає єдиний фактичний параметр (як правило, число з плаваючою точкою подвійної точності) і повертає найбільше ціле число, яке менше або дорівнює цьому фактичному параметру.
ceil()	Приймає єдиний фактичний параметр (як правило, число з плаваючою точкою подвійної точності) і повертає найближче ціле число, яке більше або дорівнює цьому фактичному параметру.
round()	Приймає єдиний фактичний параметр (як правило, число з плаваючою точкою подвійної точності) і повертає найближче ціле число. Якщо дрібна частина значення, позначеного фактичним параметром, точно дорівнює 0.5, то функція повертає найближче парне число.

Продовження таблиці 2.1

Функція	Опис
abs()	Якщо єдиний фактичний параметр має від'ємне значення, то функція повертає відповідне позитивне число; якщо фактичний параметр є позитивним, то функція повертає сам фактичний параметр.
min()	Приймає будь-яку кількість фактичних параметрів (але не менше одного) і повертає найменше з усіх значень фактичних параметрів.
max()	Приймає будь-яку кількість фактичних параметрів (але не менше одного) і повертає найбільше з усіх значень фактичних параметрів.
pow()	Приймає два числові параметри і повертає перший параметр, піднесений до степеня, рівний другому параметру. pow ($x, $y) = x^y
exp()	exp($x) = e^x
log10()	Якщо 10^y = x, то log10($x) = y
log()	Якщо e^y = x, то log($x) = y

Математичні константи. В іменуванні математичних констант в PHP використовується загальна схема *M_<constant-name>* (табл. 2.2).

Таблиця 2.2 – Математичні константи

Константа	Опис
M_PI	pi
M_PI_2	pi / 2
M_E	константа e
M_LOG2E	log2(e)

Незважаючи на те, що в мові PHP відсутній строгий контроль типів, в мові існує декілька функцій, призначених для правильності подання чисел (табл. 2.3).

Таблиця 2.3 – Функції для подання чисел

Функція	Опис	Приклади

is_numeric()	Повертає логічний результат, true, якщо переданий їй параметр являє собою числові дані будь-якого типу (зі знаком або без знака, цілочислові або з плаваючою точкою) або математичний вираз, який повертає допустиме числове значення.	is_numeric(4)
		// true
		is_numeric(4–4)
		// true
		is_numeric('1234')
		// true

is_int()	Повертає логічний результат, true, якщо переданий їй параметр являє собою цілочислове значення або математичний вираз, який повертає цілочислове значення.	is_int(4) // true
		is_int(4.2) // false
		is_int('4') // false

is_float()	Повертає логічний результат, true, якщо переданий їй параметр являє собою дійсне значення або математичний вираз, який повертає цілочислове значення.	is_float(4) // false
		is_float(4.2) // true
		is_float(4/3) // true
		is_float(M_PI) // true

2.3. Вироблення випадкових чисел

У мові PHP застосовуються два генератори випадкових чисел, що викликаються за допомогою функцій *rand()* і *mt_rand()*. З кожним з цих генераторів пов’язані три функції однакового призначення (табл. 2.4):

– функція задання початкового значення;

– функція отримання випадкового числа;

– функція, що здійснює вибірку найбільшого цілого числа, яке може бути повернуто генератором.

Таблиця 2.4 – Функції генерації випадкових чисел

Функція	Опис
srand()	Приймає єдиний позитивний цілочисловий фактичний параметр і задає значення цього параметра в якості початкового значення генератора випадкових чисел.
rand()	Якщо виклик цієї функції здійснюється без фактичних параметрів, то функція повертає "випадкове" число зі значенням від 0 до RAND_MAX (для визначення значення константи RAND_MAX може використовуватися функція *getrandmax()*). Виклик цієї функції може також здійснюватися з двома цілочисловими фактичними параметрами, що дозволяє обмежити границі значень чисел, що повертаються; перший фактичний параметр задає мінімальне значення, а другий – максимальне (включно).
getrandmax()	Повертає значення найбільшого числа, яке може бути повернуто функцією rand(). На платформі Windows це значення обмежено величиною 32 768.
mt_srand()	Ця функція аналогічна функції srand(), не зважаючи на те, що задає початкове значення для "кращого" генератора випадкових чисел.
mt_rand()	Ця функція аналогічна функції rand(), не зважаючи на те, що в ній задається "кращий" генератор випадкових чисел.

При використанні певних версій PHP для деяких платформ створюється враження, що функції rand() і mt_rand() виробляють випадкові числа, навіть без попереднього задання початкового значення. Насправді це не так.

Типовий спосіб задання початкового значення для будь-якого з генераторів випадкових чисел PHP (з використанням функції mt_srand() або srand()) полягає в наступному: mt_srand((double) microtime() * 1000000). В цьому операторі задається початкове значення генератора, що дорівнює кількості мікросекунд, що минули до даного часу з моменту відліку останньої цілої секунди. Даний оператор рекомендується поміщати на

кожну сторінку PHP перед використанням відповідної функції mt_rand() і rand().

2.4. Функції для роботи з датою і часом в PHP

Базова функція, пов'язана з датою і часом в PHP, – це функція *time()*, яка повертає кількість секунд, що пройшли з півночі 01.01.1970 (стандартний підхід Unix):

```
<?php
echo time();
?>
```

Запустивши даний скрипт, можна дізнатися, скільки секунд пройшло з півночі 01.01.1970 року до моменту виклику функції time() в скрипті.

Для того щоб, використовуючи функцію time(), отримати відомості про поточну дату і час, необхідно використовувати функцію *date()*, яка форматує системну дату/час. Функція має обов'язковий і необов'язковий аргументи і повертає дату/час, відформатовані відповідно до обов'язкового аргументу *format*, задані аргументом *timestamp* або поточний системний час, якщо *timestamp* не задано:

string date(string format[, int timestamp]).

Параметри формату format, які часто використовуються, наведені в таблиці 2.5; з іншими можна познайомитися в довіднику.

Таблиця 2.5 – Параметри формату format

Символ в рядку format	Опис	Приклад значення, що повертається
c	Дата в форматі ISO 8601	2004-02-12T15:19:21+00:00
d	День місяця, 2 цифри з нулями	від 01 до 31
D	Скорочене найменування дня тижня, 3 символи	від Mon до Sun
F	Повне найменування місяця	від January до December
g	Години в 12-годинному форматі без нулів	від 1 до 12

Продовження таблиці 2.5

Символ в рядку format	Опис	Приклад значення, що повертається
G	Години в 24-годинному форматі без нулів	від 0 до 23
h	Години в 12-годинному форматі з нулями	від 01 до 12
H	Години в 24-годинному форматі з нулями	від 00 до 23
i	Хвилини з нулями	від 00 до 59
j	День місяця без нулів	від 1 до 31
l	Повне найменування дня тижня	від Sunday до Saturday
m	Порядковий номер місяця з нулями	від 01 до 12
M	Скорочене найменування місяця, 3 символи	від Jan до Dec
n	Порядковий номер місяця без нулів	від 1 до 12
O	Різниця у часі за Гринвічем в годинах	наприклад: +0200
r	Дата в форматі RFC 2822	наприклад: Thu, 21 Dec 2000 16:01:07 +0200
s	Секунди з нулями	від 00 до 59
t	Кількість днів у місяці	від 28 до 31
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	наприклад: 1999, 2003
y	Номер року, 2 цифри	наприклад: 99, 03
z	Порядковий номер дня в році (нумерація з 0)	від 0 до 365

Будь-які інші символи, які зустрічаються в рядку *format*, будуть виведені в рядок з результатом без змін. Наприклад, наведений нижче скрипт:

```
<?php
echo date("Сьогодні d F (l) Y H:i:s Різниця у часі за Гринвічем O"). "<br>";
?>
```

виведе наступну інформацію:

"Сьогодні 15 March (Friday) 2019 08:10:11 Різниця у часі за Гринвічем +0100".

Також існує функція *gmdate()*, яка є повним аналогом функції date(), але функція gmdate() показує час не за часовим поясом сервера, на якому працює скрипт, а за Гринвічем. За допомогою цієї функції потрібно зберігати всю інформацію про дату і час, які бачить користувач, а при виведенні потрібно підлаштовуватися під часовий пояс кожного користувача.

Ще однією цікавою функцією дати і часу PHP є функція *getdate([ent timestamp]),* яка повертає асоціативний масив, що містить інформацію про дату і час, представлені міткою часу *timestamp* або поточним системним часом, якщо *timestamp* не передано. Масив містить елементи, які подано в таблиці 2.6.

Таблиця 2.6 – Можливі елементи масиву в результаті getdate()

Індекс	Опис	Приклад значення
"Seconds"	Секунди	від 0 до 59
"Minutes"	Хвилини	від 0 до 59
"Hours"	Години	від 0 до 23
"Mday"	Порядковий номер дня місяця	від 1 до 31
"Wday"	Порядковий номер дня	від 0 (неділя) до 6 (субота)
"Mon"	Порядковий номер місяця	від 1 до 12
"Year"	Порядковий номер року, 4 цифри	приклади: 1999, 2003
"Yday"	Порядковий номер дня в році (нумерація з 0)	від 0 до 365

Продовження таблиці 2.6

Індекс	Опис	Приклад значення
"Weekday"	Повне найменування дня тижня	від Sunday до Saturday
"Month"	Повне найменування місяця	від January до December
0	Кількість секунд, що пройшли з початку епохи Unix (The Unix Epoch), подібно значенням, що повертаються функцією time() та використовуються функцією date()	платформо-залежне, в більшості випадків від 2147483648 до 2147483647

Додати або відняти дні та години можна за рахунок додавання або віднімання секунд. Наприклад, щоб додати два дні, потрібно до часової позначки певною функцією time() додати 2*24*60*60 (2 дні * 24 години * * 60 хвилин * 60 секунд).

Коли сценарій отримує від користувача дату, в більшості випадків необхідно перевірити її на коректність, для чого використовується функція *bool checkdate(int month, int day, int year)*.

Функція повертає true, якщо дата, визначена аргументами, є правильною, інакше повертає false. Дата вважається правильною, якщо:

– рік в діапазоні від 1 до 32767 включно;

– місяць в діапазоні від 1 до 12 включно;

– *day* є допустимим номером дня для місяця, заданого аргументом *month*, враховуючи, що *year* може задавати високосний рік.

Поточні час і дату легко можна отримати за допомогою функції date(). Якщо необхідно створити дату і час таких компонентів, як день, місяць і рік, то потрібно скористатися функцією *mktime()*:

int mktime([int hour[, int minute[, int second[, int month[, int day[, int year[, int is_dst]]]]]]]).

Функція повертає мітку часу Unix, що відповідає даті і часу, заданими аргументами. Мітка часу – це ціле число, рівне різниці в секундах між заданою датою/часом і початком Епохи Unix (The Unix Epoch, 1 січня 1970 р).

Аргументи цієї функції можуть бути опущені в порядку справа наліво. Опущені аргументи вважаються рівними відповідним компонентам локальної дати/часу. Аргумент *is_dst* може бути встановлений в 1, якщо заданій даті відповідає літній час, 0 в іншому випадку, або -1 (початкове значення), якщо невідомо, чи діє літній час на задану дату. В останньому випадку PHP намагається визначити це самостійно. Останній день будь-якого місяця можна вичислити як "нульовий" день наступного місяця.

Завдання до лабораторної роботи 2

1. Створити двовимірний масив. Значення елементів масиву повинні бути числовими, заповнені з використанням функцій вироблення псевдовипадкових чисел.

2. Вивести на екран ключі й значення елементів масиву.

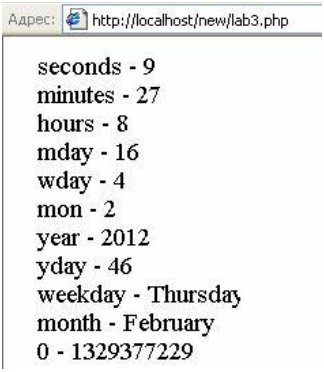
3. Використовуючи функцію compact(), створити масив з числовими індексами.

4. Використовуючи функцію extract(), створити змінні і вивести їх на екран.

5. Вивести у вікно браузера повідомлення про поточну дату і повідомлення про те, яка дата буде через два тижні.

6. Визначити кількість днів і годин, які пройшли між 09 годиною 19 червня 2018 року та 18 годиною 27 травня 2019 року.

7. Перевірити на коректність дві дати 29 лютого 2016 року і 29 лютого 2018 року. Якщо дата коректна, то вивести інформацію про неї на екран в наступному вигляді:



Якщо дата некоректна, то вивести повідомлення про те, що дата (місяць – ..., день – ..., рік – ...) є некоректною.

Лабораторна робота 3

РЕГУЛЯРНІ ВИРАЗИ PHP

3.1. Основні функції для роботи з регулярними виразами

Основні функції для роботи з Perl-сумісними регулярними виразами: preg_match(), preg_match_all(), preg_replace().

Функція *preg_match(pattern, string[, result, flags])* виконує перевірку на відповідність регулярному виразу.

Функція *preg_match_all(pattern, string, result[, flags])* шукає в рядку *string* збіг з шаблоном *pattern* і поміщає результат в масив *result*:

- pattern – шаблон регулярного виразу;
- string – рядок, в якому проводиться пошук;
- result – масив результатів (нульовий елемент масиву містить відповідність всьому шаблону, перший – першому підшаблону і т.д.);
- flags – необов'язковий параметр, що визначає те, як впорядковані результати пошуку.

Функція preg_match() здійснює пошук за шаблоном і повертає кількість знайдених відповідностей. Це може бути 0 (збіги не знайдені) та 1, оскільки preg_match() припиняє свою роботу після першого знайденого збігу. Функція preg_match() повертає false в разі, якщо під час виконання виникли будь-які помилки.

Якщо необхідно знайти або порахувати всі збіги, слід скористатися функцією preg_match_all().

Приклад 3.1

```
<?php
$str = "Доцент Смирнов зробив ". "відкриття. Його вчителем була ". "професор Іванова. ". "Цим відкриттям Смирнов ". "завоював собі ступінь ". "доктора. Раніше він був ". "тільки кандидат.";
$pattern = "/(професор | доцент)". "\s[А-Я][А-Я а-я] + (\s | \.) /i";
$n = preg_match_all($pattern, $str, $res);
for ($i = 0; $i <$n; $i ++)
echo htmlspecialchars ($res[0][$i]). "<br>";
?>
```

Функція *preg_replace()* виконує пошук і заміну за регулярним виразом: *mixed preg_replace(mixed pattern, mixed replacement, mixed subject[, int limit])*. Функція виконує пошук збігів в рядку *subject* з шаблоном *pattern* і замінює їх на *replacement*. У разі якщо параметр *limit* вказано, буде проведена заміна *limit* входжень шаблону; в разі якщо *limit* опущено або дорівнює –1, будуть замінені всі входження шаблону.

3.2. Метасимволи в регулярних виразах

Розрізняють дві множини метасимволів:

– ті, що розпізнаються всередині квадратних дужок (табл. 3.1);

– ті, що розпізнаються в будь-якому місці шаблону, за винятком квадратних дужок (табл. 3.2).

Таблиця 3.1 – Метасимволи, які розпізнаються всередині квадратних дужок

Метасимвол	Значення
\	Перехідний символ з різним призначенням.
^	Заперечення класу, але тільки якщо це перший символ (наприклад, "^ \d" задає все, крім цифр).
-	Задає діапазон символів (наприклад, "0-9" задає всі цифри, "A-Z" – всі латинські букви).
]	Вичислює символьний клас.

Метасимволи, які розпізнаються за межами квадратних дужок, можна розділити на групи в такий спосіб:

– ті, що визначають положення шуканого тексту в рядку, пов’язані з підвиразами, що обмежують символьний клас;

– квантифікатори;

– перерахування альтернатив.

Таблиця 3.2 – Метасимволи, які розпізнаються за межами квадратних дужок

Метасимвол	Значення
\	Перехідний символ з різним призначенням.
^	Оголошує початок об’єкта (або рядка в багаторядковому режимі), наприклад, ^abc – рядок починається з "abc".

Продовження таблиці 3.2

Метасимвол	Значення
$	Маркер кінця рядка (або рядка в багаторядковому режимі), наприклад, abc$ – рядок закінчується на "abc".
.	Відповідає будь-якому символу, крім символу переведення рядка (початкове значення "\n").
[	Починає визначення символьного класу.
]	Закінчує визначення символьного класу.
|	Розділяє перерахування альтернативних варіантів.
(	Починає підшаблон (регулярний підвираз).
)	Закінчує підшаблон.
?	Квантифікатор мінімізації: попередній символ зустрічається 0 або 1 раз.
*	Квантифікатор: попередній символ зустрічається 0 або більше разів.
+	Квантифікатор: попередній символ зустрічається 1 або більше разів.
{	Починає мінімальний/максимальний квантифікатор, вказує кількість повторень (інтервал повторень {min, max}, наприклад, \w{2, 3} – два або три словникові символи.
}	Закінчує мінімальний/максимальний квантифікатор.

Завдання до лабораторної роботи 3

1. Створити одновимірний масив, що включає десять дат, у форматі: день місяця, повна назва місяця, чотиризначний рік, трибуквене позначення дня тижня. У масив включити:

- "16 February 2019 Sut";
- "16 February 2020 Sun";
- поточну дату;
- дату, за рік до поточної;
- дату, за двадцять один день до поточної;
- дату, за десять днів до поточної;
- дату, за дев’ять днів до поточної;
- дату, за шість днів до поточної;
- дату, яка настане через вісім днів після поточної;

• дату, яка настане через сім днів після поточної.

2. Вивести у вікні браузера масив у вигляді:

3. Використовуючи регулярні вирази, вивести в браузері дати масиву, що відносяться до 2019 року.

4. Використовуючи регулярні вирази, вивести в браузері дати, що потрапляють в першу декаду будь-якого місяця.

5. Замінити в масиві всі «Sun» на «Неділя» і вивести результат в браузері.

Лабораторна робота 4

ОБРОБКА ФОРМ В PHP

4.1. Функції роботи з файловою системою

Основні функції роботи з файловою системою:

• *resource* ***fopen****(string filename, string mode[, bool use_include_path[, resource context]])* – відкриття файлу або URL;

• *string* ***fread****(resource handle, int length)* – бінарно-безпечне читання файлу;

• *int* ***filesize****(string filename)* – отримання розміру файлу;

• *string* ***filetype****(string filename)* – отримання типу файлу;

• *array* ***file****(string filename[, int use_include_path[, resource context]])* – читання файлу і поміщення його в масив;

• *int* ***fwrite****(resource handle, string string[, int length])* – бінарно-безпечний запис в файл;

• *int* ***file_put_contents****(string filename, mixed data[, int flags[, resource context]])* – запис рядка в файл;

• *string* ***file_get_contents****(string filename[, bool use_include_path[, resource context[, int offset[, ent maxlen]]]])* – отримання змісту файлу у вигляді одного рядка;

• *bool* ***is_readable****(string filename)* – визначення, чи доступний файл для читання;

• *bool* ***is_writable****(string filename)* – визначення, чи доступний файл для запису;

• *bool* ***unlink****(string filename[, resource context])* – видалення файлу.

4.2. Завантаження файлу на сервер

Для завантаження файлу на сервер необхідно створити html-форму:

– в тезі *form* повинен бути атрибут *enctype* зі значенням *multipart/ form-data*;

– елемент *input* повинен мати тип *type = file*;

– бажано використовувати в формі приховане поле, яке містить в собі максимально допустимий розмір файлу в байтах. При спробі завантажити файл, розмір якого більше зазначеного в цьому полі значення, буде зафіксована помилка.

Приклад 4.1

```
<form enctype = "multipart/form-data"
action = "parse.php"  method = "post">
<input type = "hidden"  name = "MAX_FILE_SIZE" value = "30000" />
Завантажити файл: <input  type = "file"  name = "myfile" /> <br>
<input  type = "submit"   value = "Відправити файл" />
</form>
```

Починаючи з версії PHP 4.1.0 вся інформація про завантажений на сервер файл міститься в глобальному масиві **$ _FILES**. Якщо включена директива *register_globals*, то значення переданих змінних доступні просто за їхніми іменами.

Масив $_FILES завжди має наступні елементи:

- $_FILES['і'мя_елемента_форми']['name'] – ім'я, яке мав файл на машині клієнта;
- $_FILES['і'мя_елемента_форми']['type'] – тип відправленого файлу;
- $_FILES['і'мя_елемента_форми']['size'] – розмір завантаженого файлу в байтах;
- $_FILES['імя_елемента_форми']['tmp_name'] – тимчасове ім'я файлу, під яким він був збережений на сервері;
- $_FILES['і'мя_елемента_форми']['error'] – код помилки, що з'явилася при завантаженні.

Виділяють п'ять типів помилок при завантаженні в PHP і відповідно $_FILES['myfile']['error'] може мати п'ять значень:

0 – помилка не відбулася, файл завантажено успішно;

1 – файл, що завантажується, перевищує розмір, встановлений директивою upload_max_filesize в файлі налаштувань php.ini;

2 – файл, що завантажується, перевищує розмір, встановлений елементом MAX_FILE_SIZE форми html;

3 – файл був завантажений частково;

4 – файл не завантажено.

На самому початку завантажені файли зберігаються в тимчасовій директорії сервера, якщо інша директорія не зазначена за допомогою опції upload_tmp_dir в файлі налаштувань php.ini.

Перемістити завантажений файл в потрібну директорію можна за допомогою функції *bool move_uploaded_file(тимчасове_ім'я_файлу, місце_призначення)*.

Функція перевіряє, чи дійсно файл, позначений рядком *тимчасове_і'мя_файлу*, був завантажений через механізм завантаження HTTP методом POST. Якщо це так, то він переміщується в файл, заданий параметром *місце_призначення* (цей параметр містить як шлях до нової директорії для зберігання, так і нове ім'я файлу).

Якщо *тимчасове_ім'я_файлу* задає неправильно завантажений файл, то ніяких дій проведено не буде, і move_uploaded_file() поверне false. Те ж саме станеться, якщо файл з яких-небудь причин не може бути переміщений. В цьому випадку інтерпретатор виведе відповідне попередження. Якщо файл, заданий параметром *місце_призначення*, існує, то функція move_uploaded_file() перезапише його.

Завдання до лабораторної роботи 4

1. Створити асоціативний масив, ключами в якому будуть можливі акаунти доступу на сторінку, а значення елементів являти собою паролі доступу.

2. Вивести на екран список.

3. На html-сторінці створити форму, що дозволяє вводити ім'я користувача і пароль (пароль не відображається у вікні).

4. Якщо введена комбінація логіна і пароля відсутня в масиві, на екран виводиться повідомлення: «На жаль, відсутня така комбінація імені користувача і пароля». Вивести на екран також IP-адресу та ім'я хоста, з якого надійшов запит.

5. Якщо комбінація логіна і пароля є в створеному масиві, вивести на екран повідомлення: «Користувач (введений логін) допущений до роботи на даному сайті».

6. Дописати введену комбінацію логіна і пароля в файл logins.txt.

7. Зчитати інформацію з файлу в масив і вивести інформацію з масиву на екран.

Лабораторна робота 5

ВЗАЄМОДІЯ PHP І MYSQL

5.1. Запис в базу даних за допомогою html-форми

Для того щоб за допомогою мови SQL записати значення в поле, використовується її команда:

INSERT INTO ім'я_таблиці SET ім'я_поля = "значення" – кожному полю, присутньому в таблиці, присвоюється значення у вигляді "ім'я_поля = 'значення'", поля перераховуються через кому. Для того щоб скористатися цією командою SQL, застосуємо функцію:

resource mysqli_query(string query[, resource link_identifier]), де покажчик *link_identifier* вказує активну базу даних (БД), до якої відправляється запит. Якщо параметр опущений, використовується останнє відкрите з'єднання.

Для запиту INSERT повертає true, якщо запит виконано успішно, і false – в разі помилки.

Приклад 5.1

Файл lect6_insert.php:

```
<?php
$conn = mysqli_connect("localhost", "root");
// встановлюємо з'єднання
$database = "Student";
$table_name = "Persons";
mysqli_select_db($database);
// вибираємо БД
$list_f = mysqli_list_fields($database, $table_name, $conn);
// отримуємо список полів в таблиці
$n = mysqli_num_fields($list_f);
// число полів
$squ = "INSERT INTO $table_name SET";
// формується текст запиту
for($i = 0; $i < $n; $i ++)
{
$name_f = mysqli_field_name($list_f, $i);
// визначаємо ім'я поля
```

```
$value = $_POST[$name_f];
// визначаємо значення поля $j = $i + 1;
$squ = $squ. $name_f. "= '$value'";
// формуємо текст запиту для одного поля
if ($j <> $n) $squ = $squ. ",";
// якщо поле не останнє, додається кома
}
echo $squ;
// перегляд правильності запиту
$result = mysqli_query($squ, $conn);
// запит до БД
if (! $result)
echo "<br> Can not add ($table_name)";
else echo "<br> Success!";
?>
```

5.2. Відображення даних, що зберігаються в MySQL

Для вибірки даних з БД використовується *SELECT * FROM table_name*, який позначає, що з таблиці table_name потрібно вибрати всі записи полів. При використанні функції PHP *mysqli_query()* з цим запитом, результат, що повертається, буде не false або true, а вказівником на таблицю значень. Для того щоб отримати конкретні значення, потрібні аналоги функції mysqli_field_name(), які повинні отримувати не ім'я, а значення поля:

1. Функція *mysqli_result(resource result, int row[, mixed field])* повертає значення однієї комірки результату запиту: *field* – може бути порядковим номером поля, ім'ям поля або ім'ям таблиці.

2. Функція *mysqli_num_rows(resource result)* повертає кількість рядків результату запиту SELECT.

3. Функція *mysqli_fetch_array(resource result[, int result_type])* обробляє ряд результатів запиту, повертає масив (асоціативний, чисельний або обидва) з обробленим рядком результату запиту, або false, якщо рядків більше немає.

Параметр *result_type* може бути однією з констант:

- MYSQLI_ASSOC – повернути тільки асоціативний масив;
- MYSQLI_NUM – повернути тільки чисельний масив;

• MYSQLI_BOTH – повернути масив, що складається як з асоціативних індексів, так і з чисельних.

Приклад 5.2

```
<?php
$conn = mysqli_connect("localhost", "root");
$database = "Student";
$table_name = "Persons";
mysqli_select_db($database);
$list_f = mysqli_list_fields($database, $table_name, $conn);
$n = mysqli_num_fields($list_f);
for($j = 0; $j < $n; $j ++)
{
$names[] = mysqli_field_name($list_f, $j);
}
$sq1 = "SELECT * FROM $table_name";
$q = mysqli_query($sq1, $conn) or die("помилка запиту");
$n1 = mysqli_num_rows($q);
echo "<table width = 90% align = center>
<tr>
<td bgcolor = '#005533' align = center>
<font color = '# FFFFFF'>
<b> $table_name </b>
</font >
</td>
 </tr>
 </table>";
echo "<table border = 1 width = 90% align = center>";
echo "<tr>";
foreach ($names as $val)
{
echo "<th align = center bgcolor = '# C2E3B6'>
<font size = 2> $val </font>
</th>";
}
echo "</tr>";
```

```
for ($i = 0; $i < $n1; $i ++)
{
echo "<tr>";
foreach ($names as $val)
{
$value = mysqli_result($q, $i, $val);
echo "<td> <font size = 2> $value </font> </td>";
}
echo "</tr>";
}
echo "</table>";
?>
```

Завдання до лабораторної роботи 5

1. За допомогою скрипта створити таблицю бази даних MySQL.

2. Створити скрипт, який дозволяє за допомогою форми заповнювати таблицю значеннями, що вводяться користувачем.

3. Створити скрипт, що виводить результат запиту до БД в таблицю на екран.

Лабораторна робота 6

ОБРОБКА СКЛАДНИХ ЗАПИТІВ MYSQL ЗА ДОПОМОГОЮ PHP

6.1. Способи обробки складних запитів

Структура виведення може не збігатися зі структурою результатів запиту. Наприклад, запит, отриманий за результатами видачі записів з двох пов'язаних таблиць:

Приклад 6.1

```
SELECT country.continent, country.countryname, city.cityname
FROM country, city
WHERE city.countryID = country.ID
ORDER BY continent, countrytame, cityname.
```

В результаті цього запиту для кожного міста буде показана країна і континент, де воно знаходиться, що і повинно відобразитися окремими рядками HTML. При цьому, якщо міста знаходяться в одній країні, то кожен раз країна і континент будуть повторюватися. Але в правильному відображенні спочатку повинен бути вказаний континент, потім країна, а потім всі міста, які знаходяться в цій країні. Тобто структура виводу не буде збігатися зі структурою найбільш зручної форми запиту.

Для таких структур, що не збігаються, зазвичай використовуються два підходи:

- створення декількох спеціалізованих запитів до БД;
- використання більш складного коду відображення.

6.1.1. Використання декількох запитів

Перший підхід є простим, але не досить ефективним, тому що збільшується число окремих запитів до БД.

Приклад 6.2

```
<?php
$glodal_dbh = msqli_connect("localhost", "root")
or die("Could not connect to database");
mysqli_select_db("my_database", $glodal_dbh)
or die("Could not select database");
```

```
function display_cities($db_connection)
// функція виводить таблицю з назвами міст та країн
{
$country_query = "SELECT id, continent, countryname
FROM country
ORDER BY continent, countryname";
$country_result = mysqli_query($country_query, $db_connection);
// виведення шапки таблиці
print("<table border = 1> \n");
print("<tr> <th> Continent </th>
<th> Country </th>
<th> City </th> </tr>");
// обробка в циклі країни
while($country_row = mysqli_fetch_row($country_result))
{
$country_id = $country_row[0];
$continent = $country_row[1];
$country_name = $country_row[2];
print("<tr aling = left valing = top>");
print("<td> $continent </td>");
print("<td> $country_name </td>");
// виведення списку міст
print("<td>");
$city_query = "SELECT cityname from city
WHERE countryID = $country_id
ORDER BY cityname";
$city_result = mysqli_query($city_query, $db_connection)
or die(mysqli_error());
// обробка в циклі даних про міста
while($city_row = mysqli_fetch_row($city_result))
{
$city_name = $city_row[0];
print("$city_name <br>");
}
// закінчення комірки зі списком міст
print("</td> </tr>");
}
```

```
print("</table> \n");
}
?>
<html>
<head><title> Cities by countries </title></head>
<body>
<?php
display_cities($global_dbh);
?>
</body>
</html>
```

6.1.2. Використання складних операторів друку

При другому підході виконуємо один запит, інформація з якого виводиться вибірково, коли кожен рядок HTML буде відповідати більше ніж одному рядку БД.

Приклад 6.3

```
<?php
$glodal_dbh = msqli_connect("localhost", "root")
or die("Could not connect to database");
mysqli_select_db("my_database", $glodal_dbh)
or die("Could not select database");
function display_cities($db_connection)
/* виведення таблиці з назвою країн, вибірково формуючи тільки по одному рядку таблиці HTML для кожної країни */
{
$query = "SELECT country.id, country.continent, country.countryname, city.cityname
FROM country, city
WHERE city.countryID = country.id
ORDER BY country.continent, country.countryname, city.cityname";
$result_id = mysqli_query($query, $db_connection)
or die(mysqli_error($query));
// виведення шапки таблиці
print("<table border = 1> \n");
print("<tr> <th> Continent </th>
```

```
<th> Country </th>
<th> City </th> </tr>");
/* ініціалізація значення для "попередньої" країни, але для країни з id=1 попередня країна буде id = 0 */
$old_country_id = 0;
// обробка в циклі рядків результату (по одному на кожне місто)
while($row_array = mysqli_fetch_row($result_id))
{
$country_id = $country_row[0];
if(country_id! = $old_country_id)
{
$continent = $country_row[1];
$country_name = $country_row[2];
/* якщо обробка даних про попередню країну на цьому закінчена, виведення кінцевих комірок з даними про місто і країну */
if($old_country_id! = 0)
print("<tr aling = left valing = top>");
print("<td> $continent </td>");
print("<td> $country_name </td> <td>");
// з початком обробки нова країна вже не є новою
$old_country_id = $country_id;
}
$city_name = $row_array[3];
print("$city_name <br>");
}
print("</td> </tr> </table>");
}
?>
<html>
<head> <title> Cities by countries </title> </head>
<body>
<?php
display_cities($global_dbh);
?>
</body>
</html>
```

Другий спосіб складніший, ніж перший. Дані видачі запиту обробляються послідовно. При цьому назва країни виводиться в браузері тільки спочатку списку міст, що знаходяться на території цієї країни.

Завдання до лабораторної роботи 6

1. Використовуючи код PHP/MySQL, створити і заповнити значеннями дві пов’язані таблиці.

2. Створити запит, що вимагає відобразити пов’язані записи двох таблиць.

3. Вивести результати створеного запиту в браузері:

– із застосуванням декількох запитів;

– з використанням складних операторів виведення.

Лабораторна робота 7

ОБ'ЄКТНА МОДЕЛЬ PHP

7.1. Класи в PHP

Мова PHP підтримує об'єктну модель. У PHP клас визначається за допомогою синтаксису, в якому використовується ключове слово *class* (імена класів і функцій не можна починати з символу підкреслення), імена властивостей класу визначаються ключовим словом *var*, а методи, що застосовуються до об'єктів класу, описуються функціями.

```
class ім'я_класу
{
  var $ім'я_властивості;
  / * Список властивостей * /
  function ім'я_методу()
  {
    / * Визначення методу * /
  }
  / * Список методів * /
}
```

Приклад 7.1

1. Створюються дві форми, одну з яких користувач вибирає в залежності від того, що він хоче створити, – опис статті або особи:

```
<form action = "task1.php">
Створити опис статті: <input type = submit
name = art_create
value = "Create Article">
</form>
<form action = "task1.php">
Створити опис особи: <input type = submit
name = pers_create
value = "Create Person">
</form>
```

2. У файлі task1.php створюються два класи – статті та особи. У кожного класу є метод для ініціалізації його змінних і метод для відображення об'єктів цього класу. При вирішенні задачі використо-

вуються дві функції, вбудовані в PHP для роботи з класами і об'єктами. Це функція *get_class(об'єкт)*, яка повертає ім'я класу, екземпляром якого є об'єкт, переданий їй як параметр. І функція *get_class_vars(ім'я_класу)*, яка повертає масив всіх властивостей класу та їхніх значень.

```
<?php
// Клас статей
class Article
{
var $title;
var $author;
var $description;
function Article($t = "Назва відсутня", $a = "Автор відсутній", $d = "Опис відсутній")
{
$this-> title = $t;
$this-> author = $a;
$this-> description = $d;
}
function show()
{
$art = "<h2> $ this-> title </h2>
<font size = -1> $this-> description </font>
<p> Автор: $this-> author </p>";
echo $art;
}
}
// Клас особ
class Person
{
var $first_name;
var $last_name;
var $email;
function Person($t = "Ім'я не введено", $a = "Прізвище не введено", $d = "Email невідомий")
{
$this-> first_name = $t;
$this-> last_name = $a;
```

```
$this-> email = $d;
}
function show()
{
$art = "<h2> $this-> first_name </h2>
<font size = -1> $this-> last_name </font>
<p> email: $ this-> email </p>";
echo $art;
}
}
if (isset($_GET["art_create"]))
{
$art = new Article;
$art_vars = get_class_vars(get_class($art));
make_form($art, $art_vars, "art_create");
if (isset($_GET["create_real"]))
{
show_($art_vars);
}
}
if (isset($_GET["pers_create"]))
{
$art = new Person;
$art_vars = get_class_vars(get_class($art));
make_form($art, $art_vars, "pers_create");
if (isset($_GET["create_real"]))
{
show_($art_vars);
}
}
function make_form($art, $art_vars, $glob)
{
$str = "<form>";
foreach ($art_vars as $var_name => $var_value)
{
$str. = "$var_name <input type = text name = $var_name> <br>";
}
```

```
$str. = "<input type = hidden name = $glob>";
$str. = "<input type = submit name = create_real
value = 'Create and Show'> </form>";
echo "$str";
}
function show_($art_vars)
{
global $art;
$k = count($art_vars);
$p = 0;
foreach($art_vars as $name => $value)
{
$p ++;
if($_GET["$ name"] == " ") $val = $art -> $name;
else $val = $_GET["$name"];
if($p <> $k) $par. = ' " ' . $val. ' ", ';
else $par. = ' " ' . $val. ' " ';
}
$const = get_class($art);
$par = '$art ->' . $const . "(" . $par. ");";
eval($par);
$art-> show();
}
?>
```

Функція *bool isset(mixed var[, mixed var [...]])* повертає true, якщо *var* існує; інакше false.

Функція *mixed eval(string code_str)* обчислює рядок, що задається в *code_str*, як код PHP. Переданий рядок повинен бути правильним кодом PHP, включаючи закінчення операторів крапкою з комою.

Завдання до лабораторної роботи 7

Використовуючи описаний приклад і підходи об'єктно-орієнтованого програмування, вибрати одну з двох можливих форм, що генеруються, і відобразити її в залежності від вибору користувача.

СПИСОК ЛІТЕРАТУРИ

1. Никсон Р. Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5 / Р. Никсон. – Санкт-Петербург : Питер, 2016. – 768 с.

2. Прохоренок Н. А. HTML, JavaScript, PHP и MySQL. Джентльменский набор Web-мастера / Н. А. Прохоренок. – Санкт-Петербург : БХВ-Петербург, 2010. – 912 с.

3. Котеров Д. В. PHP 7 / Д. В. Котеров, И. В. Симдянов. – Санкт-Петербург : БХВ-Петербург, 2016. – 1088 с.

4. PHP: Hypertext Preprocessor [Електронний ресурс]. – Режим доступу : https://www.php.net/

5. PHP.SU: Портал PHP, MySQL [Електронний ресурс]. – Режим доступу : http://www.php.su/

ЗМІСТ

Навчальне видання

МЕТОДИЧНІ ВКАЗІВКИ
до виконання лабораторних робіт з курсу
«Сучасні технології розробки Інтернет-застосунків»
для студентів спеціальності
«Прикладна та комп'ютерна лінгвістика»

Укладачі ХАЙРОВА Ніна Феліксівна
ПЕТРАСОВА Світлана Валентинівна

Відповідальний за випуск *проф. Н. В. Шаронова*

Роботу до видання рекомендував *проф. В. Д. Дмитрієнко*

В авторській редакції

План 2019 р., поз. 233

Підп. до друку 18.06.2019. Формат 60×84 1/16. Папір офсетний.
Riso-друк. Гарнітура Times New Roman. Ум. друк. арк. 3,31.
Наклад 50 прим. Зам. № 846094. Ціна договірна.

Видавець Видавничий центр НТУ «ХПІ».
Свідоцтво про державну реєстрацію ДК № 5478 від 21.08.2017 р.
61002, Харків, вул. Кирпичова, 2

Надруковано у ФЛ-П Черняк Л.О.
61002, м. Харків, вул. Багалія, 16
Свідоцтво № 24800000000079553, від 16.05.2007 р.