МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ ІМЕНІ СЕМЕНА КУЗНЕЦЯ

Методичні рекомендації
до самостійної роботи
з навчальної дисципліни

"ТЕХНОЛОГІЇ РОЗРОБКИ WEB-РЕСУРСІВ"

для студентів напряму підготовки
6.051501 "Видавничо-поліграфічна справа"
всіх форм навчання

Харків
ХНЕУ ім. С. Кузнеця
2016

Затверджено на засіданні кафедри комп'ютерних систем і технологій. Протокол № 12 від 07.04.2016 р.

Укладач В. П. Молчанов

М 54 **Методичні** рекомендації до самостійної роботи з навчальної дисципліни "Технології розробки WEB-ресурсів" для студентів напряму підготовки 6.051501 "Видавничо-поліграфічна справа" всіх форм навчання / уклад. В. П. Молчанов. – Харків : ХНЕУ ім. С. Кузнеця, 2016. – 64 с.

Наведено методи, засоби та завдання для організації самостійного вивчення та поглиблення отриманих у рамках лекційного курсу і аудиторних лабораторних робіт компетентностей, знань, вмінь та навичок. Подано перелік необхідної для виконання завдань літератури.

Рекомендовано для студентів напряму підготовки 6.051501 "Видавничо-поліграфічна справа" всіх форм навчання.

Вступ

Навчальна дисципліна "Технологія розробки WEB-ресурсів" належить до професіонального циклу базових навчальних дисциплін. Вона вивчається студентами напряму підготовки "Видавничо-поліграфічна справа" всіх форм навчання протягом шостого семестру. Навчальна дисципліна потребує від студентів інтенсивної самостійної роботи зі спеціальною літературою та програмним забезпеченням у час, вільний від обов'язкових навчальних занять. До методичних рекомендацій включені теми, які не розглядалися раніше у методичних рекомендаціях до лабораторних робіт.

Метою самостійної роботи є поглиблення знань, які були отримані на лекційних заняттях, а також підтвердження і реалізація навичок, що були сформовані на лабораторних заняттях. Важливим завданням самостійної роботи є формування компетентностей, що дозволяють студенту реалізовувати на практиці отримані знання.

Студенти повинні розширити свої знання про технологічні засоби створення серверних компонентів WEB-ресурсів, вивчити матеріали, наявні у мережі Інтернет за досліджуваним питанням, і виконати запропоновані завдання.

Основні види самостійної роботи, які запропоновані студентам із навчальної дисципліни:

вивчення лекційного матеріалу;

робота з вивчення рекомендованої літератури;

вивчення основних термінів та понять за темами дисципліни;

підготовка до лабораторних занять;

перевірка особистих знань за запитаннями для самостійного контролю та виконання контрольних завдань;

робота над індивідуальним завданням.

Для закріплення і перевірки набутих компетентностей під час самостійної роботи до кожної роботи пропонуються довідкові матеріали, практичні завдання і запитання для самодіагностики.

Самостійна робота 1. Налаштування сервера IIS

Мета роботи: отримання знань та навичок зі створення середовища для розроблення WEB-ресурсів із серверними компонентами.

У результаті виконання самостійної роботи у студента формуються такі **компетентності:** здатність самостійно обирати технологію створення WEB-додатка та засоби створення і відлагодження WEB-додатків.

Результатом виконання самостійної роботи є налаштоване середовище та демонстрація його працездатності.

Завдання для самостійної роботи

Виконувати дану самостійну роботу доцільно перед виконанням першої лабораторної роботи "Дослідження взаємодії програми з сервером на основі CGI".

У ході виконання роботи необхідно:

1. Обґрунтувати вибір типу сервера.
2. Встановити сервер IIS.
3. Налаштувати сервер IIS.
4. Вивчити управління роботою сервера.
5. Налагодити і виконати CGI-програму з використанням IIS.

Контрольні запитання для самодіагностики

1. Опишіть процес взаємодії сервера і браузера.
2. Дайте характеристику серверів, які використовуються для відлагодження WEB-додатків. Обґрунтуйте ваш вибір.
3. Опишіть процес установки сервера IIS.
4. Як розподілені налаштування за різними рівнями сервера?
5. Як змінити порт для сайта?
6. Як розмістити на сервері ще один сайт?
7. Для чого може знадобитися зупинка або перезапуск сервера?
8. Дайте загальну характеристику інтерфейсу CGI.
9. Який заголовок має видавати у вихідний потік CGI-програма?
10. Як задати для сервера дозвіл на виконання CGI-програми?
11. Як передаються дані у програму CGI?
12. Як використовуються вхідні та вихідний потоки у програмі CGI?

Методичні рекомендації до теми

1. Обґрунтування вибору типу сервера.

Для налагодження WEB-додатків на локальній машині необхідно встановити WWW-сервер (іноді його називають персональним WWW-сервером). Серед безлічі різних розробок, найбільшого поширення набули дві: *Microsoft IIS* і вільно поширюваний *Apache*.

Перший функціонує тільки на платформі *Windows*, але забезпечує функціонування ресурсів, створених із використанням як технології .NET, так і PHP (після підключення відповідного модуля). Другий – багатоплатформовий, але технологія .NET не підтримується.

IIS поставляється разом із операційною системою, але для установки вимагає додаткових дій. Налаштування IIS здійснюється за допомогою спеціальної графічної консолі, що є звичним для користувачів *Windows*.

Після установки настройки *Apache* виконуються за допомогою конфігураційних файлів, і це вимагає додаткового вивчення. Для спрощення роботи з цим сервером випускаються різні збірки (набір дистрибутивних матеріалів і скрипт для розгортання), під час їх використання потрібно тільки запустити скрипт, усі інші дії з розгортання та налаштування виконуються автоматично. Як правило, у цих збірках разом з сервером WEB установлюється модуль PHP і сервер баз даних MySQL, а також ряд допоміжних засобів. Найбільш популярними засобами цього типу є Денвер і XAMPP. У мережі Інтернет міститься величезна кількість матеріалів, присвячених цим засобам, їх легко можна знайти самостійно.

2. Установлення сервера IIS.

Сервер IIS різних версій (5, 6, 7.0, 7.5) поставляється разом з операційною системою *Windows,* але перед використанням потрібно його встановити. Дії з установки можуть відрізнятися для різних версій операційної системи, але у цілому схожі. Слід розглянути установку сервера у середовищі *Windows 7* як наймасовішої операційної системи.

Для установки необхідно зробити таке:

Пуск -> **Панель управления** -> **Программы** -> **Включение или отключение компонентов Windows**.

Після відкриття вікна необхідно вибрати розділ Служби IIS, розкрити і вибрати потрібні для роботи компоненти.

Обов'язково повинні бути включені використовувані компоненти розробки додатків (у даному випадку, можна включити все), консоль і служба управління IIS. Після вибору необхідних компонентів натиснути кнопку Ok.

Після установки компонентів доцільно перезавантажити комп'ютер.

3. Налаштування сервера IIS.

Після установки налаштування і управління сервером може здійснюватися за допомогою графічного інтерфейсу диспетчера служб IIS.

Щоб відкрити диспетчер IIS із меню "Пуск" потрібно натиснути кнопку **Пуск** й обрати **Панель управления**. Потім виконати одну з таких дій:

під час роботи у середовищі *Windows Vista*® вибрати **Система и ее обслуживание** та **Администрирование**;

Під час роботи у середовищі *Windows 7* або *Windows Server 2008* вибрати **Система и безопасность** і **Администрирование**.

У вікні **Администрирование** двічі клацніть пункт **Диспетчер служб IIS**.

У *Windows 10* вікно управління сервером можна відкрити двома способами:

1. **Пуск** -> **Все приложения** -> **Средства администрирования** -> **Диспетчер служб IIS**.

2. Контекстне меню на значку **Компьютер** -> **Управление** -> **Службы и приложения** -> **Диспетчер служб IIS**.

У *Windows* всіх версій диспетчер можливо відкрити через функцію пошуку: у полі **Поиск** набрати **inetmgr**, натиснути **Ввод**.

За частого використання зручно створити ярлик у швидко доступному місці (наприклад, на робочому столі).

У вікні диспетчера IIS три панелі: дві бічних (**Подключения** і **Действия**) і центральна. У панелі **Подключения** відображається структура об'єктів сервера (сайти і пули додатків). У панелі **Действия** містяться команди для виконання основних дій з управління об'єктами. У центральній панелі відображаються дані, що належать до налаштувань об'єктів управління, у двох режимах: **Просмотр возможностей** і **Просмотр содержимого**.

Для захисту від взаємного впливу і раціонального використання системних ресурсів програми і сайти згруповані у пули, кожен з яких має конкретні параметри, велика частина параметрів пов'язана з настройками середовища виконання *.NET*. Використання декількох пулів доцільно за умови більшої кількості додатків, до того ж використовують різні версії, тому у даному випадку (технологія PHP, локальний комп'ютер, навчальні завдання) можна видалити всі пули, залишивши тільки *DefaultAppPool*.

За замовчуванням на сервері розташований один сайт із ім'ям *Default Web Site*, якому відповідає папка *C:/inetpub/wwwroot*. Ім'я хоста для цього сайта *localhost*. Це означає, що під час набору у адресному рядку браузера *http://localhost* будуть відображатися файли цього сайта.

Слід мати на увазі, що для нормального функціонування сервера до фізичних папок сайтів повинен бути відкритий загальний доступ (**контекстное меню** -> **Свойства** -> **Доступ** -> **Общий доступ**).

4. Управління роботою сервера.

Дії з управління виконуються з диспетчера служб IIS.

Для управління потрібним об'єктом (сервер, пул, сайт) необхідно вибрати його у лівій панелі диспетчера (Підключення). У правій панелі (Дії) з'явиться список доступних дій. Об'єкту кожного типу буде відповідати свій список дій. Для сервера – це запуск, зупинка, перезапуск і ряд додаткових налаштувань.

Наприклад, у разі необхідності змінити порт, на якому функціонує сайт, слід у лівій панелі виділити назву сайта, а у правій вибрати дію **Привязки…**

Для налагодження WEB-додатків, створених із використанням технології PHP, до сервера повинен бути підключений відповідний модуль для інтерпретації програмного коду.

Модуль PHP можна скачати з сайта розробника *http://windows.php.net /download/*. Там пропонується кілька варіантів. Доцільно вибрати варіант *VC9 x86 Non Thread Safe*. Для роботи у режимі *FastCGI* це найбільш швидка і стабільна реалізація. Після запуску установки, вибирається місце для файлів, потім режим роботи PHP (вибрати *IISFastCGI*). у інших вікнах залишити значення за замовчуванням.

Для перевірки роботи модуля PHP необхідно помістити у кореневу папку WEB-сайта (*c:\inetpub\wwwroot*) файл *index.php* з таким змістом:

```
<html>
<head>
<title>Test PHP</title>
</head>
<body>
<?php phpinfo()?>
</body>
</html>
```

Якщо все працює правильно, то після відкриття сайта у браузері (*http://localhost*) відобразиться сторінка з інформацією про встановлення PHP.

5. Налагодження та виконання CGI-програм з використанням IIS.

Найпростішим і історично найпершим способом створення динамічних ресурсів для мережі Інтернет є використання інтерфейсу *Common Gateway Interface* (*CGI*).

CGI є стандартом інтерфейсу зовнішньої прикладної програми з WEB-сервером.

Клієнтська частина програми є HTML-документом, у якому реалізований інтерфейс з користувачем. Реалізується у основному за допомогою форм, хоча можуть використовуватися й інші елементи, у тому числі скрипти й аплети.

Серверна частина складається з виконуваного модуля, що вирішує основні завдання оброблення даних, що надходять від клієнтської частини, і формування відповіді у форматі HTML. Такий модуль називається cgi-модулем або розширенням сервера.

Сам інтерфейс CGI є правилами взаємодії сервера з зовнішнім щодо до нього модулем (cgi-модулем). Він визначає чотири інформаційних потоки:

змінні оточення;

стандартний вхідний потік;

стандартний вихідний потік;

командний рядок.

Змінні оточення. У середовищі будь-якої ОС вони слугують для зв'язку між процесами, і становлять набір спеціальних змінних, значення яких установлюються перед запуском програми і доступні програмі у ході виконання. Склад змінних і їх імена залежать від процесу, що запускає цю програму.

Цих змінних досить багато, вони містять різні дані, зокрема:

SERVER_SOFTWARE – містить інформацію про WEB-сервер;

GATEWAY_INTERFACE – містить інформацію про версію CGI;

CONTENT_LENGTH – значення цієї змінної відповідає довжині стандартного вхідного потоку у символах;

CONTENT_TYPE – ця змінна містить додаткову інформацію про зміст HTTP-запиту;

REQUEST_METHOD – метод запиту, який був використаний "POST", "GET", "HEAD" і т. д.;

PATH_INFO – значення змінної містить отриманий від клієнта віртуальний шлях до cgi-модуля;

PATH_TRANSLATED – значення змінної містить фізичний шлях до cgi-модуля, перетворений з значення PATH_INFO;

QUERY_STRING – значення цієї змінної відповідає рядку символів, наступному за знаком "?" у URL відповідного даному запиту;

HTTP_USER_AGENT – назва програми перегляду, яку використовує клієнт під час відправлення запиту.

Аргументи командного рядка. Крім змінних оточення програмі передається список параметрів, які розміщені у командному рядку під час запуску. CGI-модуль у командному рядку від сервера отримує:

залишок URL після імені cgi-модуля у якості першого параметра (перший параметр буде порожній, якщо було присутнє тільки ім'я cgi-модуля);

список ключових слів як залишок командного рядка для скрипта пошуку або імена полів форми з доданим знаком рівності і відповідних значень змінних.

Стандартний вхідний потік. Склад даних, що відправляються сервером на вхідний потік CGI-модуля, залежить від використаного клієнтом методу запиту.

У разі методу запиту POST дані передаються як вміст HTTP запиту у такій формі:

name = value & name1 = value1 & ... & nameN = valueN,

де *name* – ім'я змінної,

value – значення змінної,

N – кількість змінних.

Установлюються значення змінних оточення *CONTENT_LENGTH* і *CONTENT_TYPE*.

Таким чином, якщо у результаті використання форми з аргументом тега *FORM METHOD = "POST",* сформований рядок даних *firm = MMM & price = 100023*, то сервер встановить значення *CONTENT_LENGTH* рівним 21 і *CONTENT_TYPE* у *application/x-www-form-urlencoded*, а у стандартний потік введення посилається блок даних.

У разі запиту методом *GET*, рядок даних передається як частина URL. Наприклад, *http:/ host/bin/script?Name1=value1&name2=value2.* У цьому випадку змінна оточення *QUERY_STRING* набуває значення *name1=value1& name2=value2*, а у вхідний потік нічого не надсилається.

Стандартний вихідний потік. CGI-модуль виводить інформацію у стандартний вихідний потік. Це може бути або документ, згенерований cgi-модулем, або інструкція сервера, де отримати необхідний документ.

Вихід cgi-модуля повинен починатися з заголовка, що містить певні рядки і завершуватися двома символами *CR* ("\n", код 0x10). Ці строки

сприймаються як директиви сервера. Рядки, які не є директивами сервера, посилаються безпосередньо клієнтові. CGI специфікація визначає три директиви сервера:

Content-type – MIME або тип повертається документа;

Location – вказує серверу, що повертається не сам документ, а посилання на нього;

Status – задає рядок-статус, у якому кодується різноманітна інформація (наприклад, про успішне завершення операції).

Під час відправки клієнту HTML-документа директива має вигляд *Content-Type:text/html \n \n.*

Якщо буде переданий URL, то директива може виглядати, наприклад, так *Location:http://host/file.htm*, і сервер поверне клієнтові заданий цим шляхом документ, якби клієнт запитував цей документ безпосередньо.

Зазвичай CGI-програми розміщуються у каталозі зі стандартним ім'ям (найчастіше *cgi-bin*). У цьому випадку для ідентифікації досить імені файла (без розширення). Посилання на цей ресурс може виглядати, наприклад, так: *http://www.orp.com/cgi-bin/myscript*. Іноді сервер не містить спеціального каталогу. У цьому випадку файли програм можуть розташовуватися у довільному місці, шлях до якого включено у URL. Але необхідно вказувати і розширення файла: *http://www.orp.com/mucat/myscript.exe*. Для деяких серверів (наприклад, IIS 7) може знадобитися задавати дозвіл на виконання із зазначенням імені програми.

Використовуватися такі посилання можуть у будь-якому атрибуті, якій допускає URL (*href, src*). Після самої адреси можуть розміщуватися дані, які будуть передані серверу разом із URL. Дані починаються з символів за знаком "?", Наприклад,

** виклик CGI **

У цьому випадку серверу буде відправлено рядок */mycgi/cd2.exe?date*. Сервер передає цей рядок CGI-програмі через змінні оточення (в *PATH_INFO* буде URL, а у *QUERY_STRING* – рядок "date").

Після прийняття та розшифровки запиту виконується динамічне формування cgi-модулем HTML-документа, наприклад, таблиці вибірки з бази даних.

Далі наводиться приклад програми на мові C #, яка виводить клієнту вітання у відповідь на запит.

Код програми:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace cgi1
{
class Program
{
  static void Main(string[] args)
  {
        Console.Write("Content-Type:text/html\n\n");
        Console.WriteLine("<html>");
        Console.WriteLine("<head>");
        Console.WriteLine("<title>Простейшая CGI-программа</title>");
        Console.WriteLine("</head>");
        Console.WriteLine("<body>");
        Console.WriteLine("<h1>Вас приветствует CGI-программа!</h1>");
        Console.WriteLine("</body>");
        Console.WriteLine("</html>");
  }
 }
}
```

Програма повинна бути відкомпільована, а отриманий виконуваний файл (наприклад, proba.exe) поміщений у відповідний каталог сервера.

Конфігурація сервера повинна дозволяти запуск CGI-програм. Для IIS необхідно вибрати сайт, який містить виконуваний файл (у лівій панелі), двічі клацнути на значку **Сопоставление обработчиков…** У вікні, знайти і виділити рядок, що містить у поле Ім'я *CGE-exe*. у панелі **Действия** вибрати **Смена разрешений** і задати відповідні дозволи. Після цього необхідно вказати, яку програму дозволяється виконувати. Вибрати дію **Изменить**, і у діалоговому вікні у полі **Файл программы** вибрати потрібний файл.

Після виконання всіх зазначених налаштувань перевіряється робота ресурсу з використанням браузера. Набравши у рядку адреси браузера URL (що-небудь на зразок *http://localhost/cgi-bin/proba.exe*), який вказує шлях до створеного файла, побачимо у вікні згенеровану сторінку.

Програма, яка повинна отримати від клієнта дані для оброблення, буде виглядати дещо складніше за рахунок того, що потрібно виконувати розбір одержуваних рядків.

Література: [6; 9].

Самостійна робота 2. Створення та оброблення XML-документів

Мета роботи: отримання знань та навичок у створенні та використанні XML-документів.

У результаті виконання самостійної роботи у студента формуються такі **компетентності:** здатність створювати, аналізувати та використовувати XML-документи, та XML-додатки, використовувати мову XML для зберігання даних додатків

Результатом виконання самостійної роботи є файлі зі створеними документами для кожного пункту завдання та індивідуального завдання "Створення та обробка XML-документів". Тема індивідуального завдання вибирається студентом самостійно за узгодженням із викладачем, орієнтуючись на типові зразки.

Завдання для самостійної роботи

Виконувати дану самостійну роботу доцільно під час вивчення теми "Мова розмітки XML".

У ході виконання роботи необхідно.

1. Створити три XML-документа різної структури за власним задумом.
2. Створити DTD для перевірки валідністі документів.
3. Виконати перетворення XML-документів за допомогою XSL до HTML.

Контрольні запитання для самодіагностики

1. Сформулюйте правила, що визначають коректність XML-документа.
2. Дайте визначення поняття валідний документ
3. Якими засобами можна перевірити валідність і коректність документа?
4. Що таке DTD, з чого вони складаються?
5. Як оголошуються типи елементів у DTD?
6. Як оголошуються припустимі атрибути елементів у DTD?
7. Що задає таке оголошення <!ATTLIST term tC CDATA #REQUIRED>?
8. Для чого слугує мова XSL?
9. Сформулюйте загальні правила завдання XSL-перетворень.

10. Що таке шаблон у XSLT?
11. Які перетворення задає наступна група елементів?

```
<xsl:for-each select="список/источник">
<LI><SPAN STYLE="font-style:italic">
<xsl:value-of select="автор"/>
<xsl:value-of select="название"/>
<xsl:value-of select="издательство"/>
<xsl:value-of select="год"/>
</SPAN>
<xsl:value-of select="кол_страниц"/></LI>
</xsl:for-each>
```

Методичні рекомендації до теми

Призначення мов розмітки полягає у описі структурованих документів. Головним недоліком мов, що існували до появи XML, була їх негнучкість. Наприклад, під час спроби використовувати HTML для опису довільних документів (яких-небудь банківських або бухгалтерських) існуючих тегів явно не вистачить.

Взагалі, кількість і різноманітність необхідних мов розмітки визначається кількістю методів оброблення даних. Саме тому і була запропонована XML – мова, використовуючи яку можна створювати свою власну мову розмітки для конкретних застосувань.

Правила розмітки XML-документів. XML-документ формується за допомогою розмітки, що виділяє окремі елементи. Елемент XML складається з початкового і кінцевого тегів, між якими розміщується довільний текст. Причому, такі символи, як пропуск, повернення каретки і табуляція, сприймаються як пропуск зовні тегів, але у даних (усередині тегів) ураховуються як спеціальні символи.

Поняття тега і загальні правила їх вживання дуже схожі на аналогічні у HTML. Основна відмінність у тому, що імена тегів не фіксовані, а вибираються розробником відповідно до предметної області.

Теги виділяються символами "<" і ">". Кінцеві теги починаються символами "</". Одиночні (непарні) теги містять так званий порожній елемент і закінчуються символами "/>". Наприклад, <Greeting ></Greeting> або <problem />.

Теги можуть вкладатися один у одного без "перетину" (якщо елемент починається усередині іншого елемента, він повинен і закінчуватися усередині цього елемента).

Імена (назви) тегів повинні починатися з букви, символу підкреслення або двокрапки і містити букви, цифри, символи підкреслення, дефіси, крапки точки і двокрапки (але не пропуски). Хоча у рекомендаціях XML1.0 явно це не вказується, слід уникати використання двокрапки у іменах тегів, оскільки двокрапка використовується у XML під час вказівки просторів імен. Імена задаються з урахуванням регістра. <DOCUMENT> і <document> це різні теги.

Слід особливо підкреслити, що не забороняється використання кирилиці та інших національних алфавітів.

Початкові і порожні теги можуть містити атрибути, що дозволяють визначити додаткові дані (ім'я_атр="значення_атр ").

У XML не допускаються атрибути, яким не привласнені які-небудь значення.

<circle origin_x="10.0" origin_y " 20.0" radius="10.0" />

Лапки застосовуються подвійні або одинарні:

<quotation text='He said . "Not that !"1 />

У разі необхідності задати текст зі складнішим поєднанням спеціальних символів можна використовувати підстановки (як і у HTML), тут вони називаються об'єктними посиланнями (' – символ **'**, " – символ **"**, & – символ **&**, < – символ **<**, > – символ **>**, $ &#x1a і тому подібне).

Для виключення з процесу синтаксичного аналізу великих фрагментів коду використовуються спеціальні секції <![CDATA [....]]> Усе, що знаходиться усередині секції не аналізується. Наприклад, це може бути код скрипта (а у нього свої правила коректності) і тому подібне.

Оскільки користувач сам визначає імена тегів і зміст елементів, то одні і ті ж дані можна розмістити або у тегах, або у атрибутах. Твердо сталого правила немає, проте занадто велика кількість атрибутів призводить до того, що документ важко читати.

Структура XML-документа. Документ складається з трьох частин: прологу (може бути порожнім), кореневого елемента і необов'язкової загальної частини частки.

Пролог може містити XML-оголошення (наприклад <?xml version = "1.0"?>), оголошення типу документа або DTD (<!DOCTYPE .>), інструкції з оброблення, коментарі і пропуски. Рекомендується включати в пролог, як мінімум, XML-оголошення, у якому указується вживана версія мови XML.

Кореневий елемент містить вкладені елементи. XML-документ повинен містити тільки один кореневий елемент.

Необов'язкова загальна частина може складатися з XML-коментарів, інструкцій з оброблення і пропусків (пропуски, табуляції і т. д.).

Оголошення знаходиться у першому рядку у елементі <?xml...?>, наприклад,

<?xml version = "1.0" standalone="yes" encoding="UTF-8"?>

У XML-оголошенні допускається вказівка таких атрибутів:

version – версія XML-оголошення;

encoding – кодування символів документа (за замовчуванням передбачається UTF-8);

standalone – "yes ", якщо цей документ не посилається на зовнішні об'єкти,"no " – інакше, (за замовчуванням – "yes ").

Коментарі схожі на HTML-коментарі. Їх можна використовувати для включення у документ пояснень, які ігноруються під час синтаксичного розбору. Коментар поміщається між символами <!-- ... -->.

Коментарі не повинні поміщатися перед XML-оголошенням. Не можна поміщати коментар усередині символів розмітки (тегів). Неприпустимо використовувати символ "--" усередині коментаря.

Інструкції з оброблення управляють функціонуванням XML-процесора. Вони починаються послідовністю символів "<?", і завершуються символами "?>". Не можна використовувати тег <?xml … ?> (це оголошення). Інструкції з оброблення залежать від використовуваного процесора, а не є специфікаціями мови XML. У загальному випадку, інструкції можуть бути поміщені у будь-яке місце документів поза елементами.

Далі наведено приклад XML-документа.

```
<?xml version="1.0" ?>
<?xml-stylesheet type="text/css" href="my .css" ?>
<!DOCTYPE DOCUMENT [
<!ELEMENT DOCUMENT (ZAG , STUDENT* )>
<!ELEMENT ZAG (#PCDATA )>
<!ELEMENT STUDENT (NAME , photo,BOOK* )>
<!ELEMENT NAME (#PCDATA )>
<!ELEMENT photo (#PCDATA )>
<!ELEMENT BOOK (AUTOR,TITLE )>
<!ELEMENT AUTOR (#PCDATA )>
<!ELEMENT TITLE (#PCDATA )>
]>
<DOCUMENT >
<ZAG>Список студентів, що не здали книги</ZAG>
<STUDENT >
```

```
 <NAME>Іванов</NAME>
 <photo>im1.gif</photo>
 <BOOK >
  <AUTOR>Пушкар </AUTOR>
  <TITLE > Основи наукових досліджень</TITLE>
 </BOOK >
 <BOOK >
   <AUTOR>Молчанов </AUTOR>
   <TITLE > Технології WEB-дизайну</TITLE>
  </BOOK >
</STUDENT >
</DOCUMENT >
```

Відображення XML-документів у браузері. Можна відкрити XML-документ безпосередньо у *Internet Explorer* або у іншому браузері. Останні версії цих програм підтримують XML і мають вбудовані засоби контролю правильності документа. Проте якщо XML-документ не містить зв'язку з таблицею стилів, то браузер лише позначає різні складові частини документа різним кольором, а також представляє кореневий елемент у вигляді ієрархічного дерева з можливістю згортання і розгортання структури і перегляду з меншою або більшою мірою деталізації (символи знаку мінус "-" або плюс "+" зліва від початкового тега).

Використання таблиць стилів для відображення XML-документів. Оскільки у XML розробник створює свої власні елементи, браузер не має вбудованих засобів, що дозволяють правильно відобразити їх. Найбільш простим способом описати, як повинні відображатися елементи є створення таблиці стилів і скріплення її з XML-документом.

Варто нагадати, що таблиця стилів є текстовим файлом, зазвичай з розширенням .css, який містить набір правил, що складаються з селектора (у простому випадку це ім'я тега) і декларацій (набір пар *ім'я_властивості:значення* у фігурних дужках), що повідомляють браузеру, яким чином форматувати і відображати елементи. Створюватися він може найпростішим текстовим редактором (наприклад, Блокнот). Більшість властивостей, які розглядалися під час форматування HTML, можуть бути застосовні і до XML-документів.

Специфікація CSS містить велику кількість різних селекторів. Зокрема, імена тегів (p,h1), класові (.cl) і id-селектори (#id). Проте, під час роботі з xml-документами класові та id-селектори у такому вигляді не працюють.

Для зв'язку правил з елементами більш за все підходить селектор атрибутів. Він конструюється з імені елемент та імені атрибута у квадратних дужках:

p[class] – усі теги **p**, що мають атрибут class ;

p[class="c1"] – усі теги **p**, що мають атрибут class із значенням 1;

*[id="id1"] – усі теги, що мають атрибут id із значенням id1.

Слід підкреслити, що використовувати можна будь-які атрибути, а не лише class та id.

Таблиці стилів не дають можливості модифікувати або реорганізовувати вміст документа.

Вони не дозволяє здійснювати доступ до атрибутів, інструкцій з оброблення та іншим компонентам XML, а також не дають можливості обробляти інформацію, яку ці компоненти містять. Тому, наприклад, неможливо відобразити малюнок.

Частіше за все таблиці стилів використовуються для відображення текстового змісту документа.

Створення DDT та перевірка валідності. Важливою можливістю технології XML є можливість перевірки документа на відповідність формальним критеріям відсутності помилок. Існують два критерії для перевірки XML-документа: валідність (дієвість) і коректності (правильність).

Документ є формально коректним, якщо він задовольняє загальні вимоги до XML-документів.

Під валідністю розуміється відповідність документа синтаксису, який визначає розробник.

Перевірка коректності повинна забезпечити відповідність всього документа основним вимогам синтаксису XML. Перевіряється, чи немає незакритих тегів, чи всі атрибути поміщені у лапки, чи немає тегів із порушенням вкладеності, чи має документ єдиний кореневий елемент і тому подібне. Дана перевірка є обов'язковою для всіх XML-документів.

Якщо, наприклад, початковий тег не має відповідного йому кінцевого тега, то це *неправильно побудований* документ XML. Документ, який неправильно побудований, не може вважатися за документ XML, XML-процесор (парсер) не повинен обробляти його звичайним способом, він зобов'язаний класифікувати ситуацію як фатальну помилку, подальше оброблення припиняється. Так поводяться більшість браузерів.

Вимоги валідності є додатковим набором правил у специфікації XML, яким необхідно слідувати, щоб уникнути помилок при використанні

документа. Оскільки валідність є не обов'язковою для XML-документа, відхилення від вимог вважається лише за помилку (*error*), обробка не припиняється. Якщо XML-процесор зустрічає помилку, він може просто видати повідомлення про неї і продовжити виконання обробки.

Для перевірки валідності документ повинен містити формальний опис синтаксису, визначеного розробником. Перевірка валідності можлива лише для тих документів, для яких заданий такий формальний опис.

Для опису синтаксису XML-документів нині створено дві мови: DTD (*Document Type Definition*) і XSD (*XML Schema Definition*). Обидві мови стандартизовано на рівні *World Wide Web Consortium* (W3C). Фактично, XSD і DTD – це "граматика для певного класу документів".

Якщо порівнювати XSD і DTD, то мова XSD дозволяє задавати більш суворі обмеження на типи даних XML-документа, але є складнішою. Не дивлячись на те, що схеми даних XSD власне є документами XML, редагувати їх за допомогою звичайного XML-редактора виявляється незручним через їх достатню складність. Тому, як правило, для їх створення використовуються спеціальні редактори, що відображають їх у наглядному графічному вигляді. Тому буде розглянуто саме DTD.

Оголошення типу документа (DTD) визначає структуру документа та зміст його елементів. Це дозволяє перевіряти документ на валідність (на додаток до перевірки на коректність). Кожен елемент і атрибут, який використовується, повинні відповідати специфікації DTD, записаній у відповідному оголошенні.

Оголошенням типу документа (DTD) є блок XML-розмітки, який під час додавання у пролог, може розташовуватися у будь-якому місці прологу – поза іншою розміткою, після XML-оголошення.

DTD може міститися у прологу XML-документа (внутрішнє) або у зовнішньому файлі, ім'я якого вказується у документі (зовнішнє). Виглядає це так для внутрішнього оголошення:

```
...
<! DOCTYPE rootname [
<!ELEMENT rootname (contacts, issues, authors )>
...
]>
...
```

Або так для зовнішнього:

```
<?xml version=1.0 standalone="no " ?>
<! DOCTYPE rootname SYSTEM "fileDTD.dtd">
...
```

Можна одночасно використовувати внутрішні і зовнішні визначення DTD за допомогою елемента <!DOCTYPE> такого вигляду:

```
!DOCTYPE rootname SYSTEM "URL"
 […
]>
```

Зазвичай внутрішні визначення доповнюють зовнішні.

Частіше за все DTD містить оголошення типів елементів (вони визначають типи елементів, які може містити документ, а також вміст і порядок проходження елементів) та оголошення списків атрибутів (кожне оголошення задає імена атрибутів, які можуть бути використані з певним типом елемента, а також типи даних і встановлювані за замовчуванням значення цих атрибутів).

У валідному XML-документі необхідно повністю оголосити тип кожного елемента, який використовується. Оголошення типу елемента указує на ім'я типу елемента, допустимий вміст елемента і порядок розміщення дочірніх елементів. Як єдине ціле, оголошення типів елементів у DTD задає повну логічну структуру документа.

Оголошення типу елемента має таку узагальнену форму:

```
<!ELEMENT Ім'я опис_вмісту>
```

Тут "Ім'я" є ім'я оголошуваного типу елемента. "опис_вмісту" визначає, що може містити елемент. Оголошувати певний тип елемента у даному документі можна тільки один раз.

Наприклад, оголошення типу елемента з ім'ям TITLE, для вмісту якого можуть використовуватися тільки символьні дані (дочірні елементи не допускаються):

```
<!ELEMENT TITLE (#PCDATA )>
```

#PCDATA (parseable character data) – будь-яка інформація, з якою може працювати програма-аналізатор. Це текст, що не є розміткою. Він називається розібраними символьними даними.

Крім #PCDATA "опис_вмісту" може містить один з чотирьох типів вмісту: EMPTY (елемент має бути порожнім – тобто не може мати вмісту), ANY (будь-який, елемент цього типу може містити або не містити дочірні елементи, мати або не мати символьних даних), дочірній і змішаний вміст.

Приклад XML-документа, що описує одну книгу:

```
<?xml version="1.0" encoding="windows-1251 " ?>
<!DOCTYPE BOOK
   [
```

```
   <!ELEMENT BOOK (TITLE, AUTHOR )>
   <!ELEMENT TITLE (#PCDATA )>
   <!ELEMENT AUTHOR (#PCDATA )>
   ]
>
<BOOK >
   <TITLE>The Scarlet Letter</TITLE>
   <AUTHOR>Nathaniel Hawthorne</AUTHOR>
</BOOK >
```

У цьому документі тип елемента BOOK оголошений таким, що має дочірній вміст елементи TITLE і AUTHOR. У даному прикладі елемент BOOK повинен мати рівно один дочірній елемент TITLE, за яким слідує рівно один дочірній елемент AUTHOR. Пропуск дочірнього елемента або використання одного і того ж типу дочірнього елемента більше одного разу також неприпустимо.

У разі необхідності можна вказати, що елемент може мати будь-який з серії допустимих дочірніх елементів, елементи розділяються символом "|". Наприклад, наступне DTD указує, що елемент FILM може складатися з одного дочірнього елемента STAR, або одного дочірнього елемента NARRATOR, або одного дочірнього елемента INSTRUCTOR:

```
<!DOCTYPE FILM
   [
   <!ELEMENT FILM (STAR | NARRATOR | INSTRUCTOR)>
   <!ELEMENT STAR (#PCDATA )>
   <!ELEMENT NARRATOR (#PCDATA )>
   <!ELEMENT INSTRUCTOR (#PCDATA )>
   ]
>
```

Отже, такий елемент буде валідним:

```
<FILM >
   <STAR>Robert Redford</STAR>
</FILM >
```

А цей – ні (оскільки можна включити лише один із дочірніх елементів):

```
<FILM >
   <NARRATOR>Sir Gregory Parsloe</NARRATOR>
   <INSTRUCTOR>Galahad Threepwood</INSTRUCTOR>
</FILM >
```

Будь-яку з цих форм вмісту можна змінювати, використовуючи символи: **?** (жоден або один з попередніх елементів), **+** (один або більш із попередніх елементів), ***** (жоден або будь-яка кількість із попередніх елементів.

Наприклад, наступне оголошення означає, що можна включити один або більш дочірніх елементів NAME, і що дочірній елемент HEIGHT є не обов'язковим:

```
<!ELEMENT MOUNTAIN (NAME+, HEIGHT ?, STATE )>
```

Таким чином, такий елемент буде валідним:

```
<MOUNTAIN >
   <NAME>Pueblo Peak</NAME>
   <NAME>Taos Mountain</NAME>
   <STATE>New Mexico</STATE>
</MOUNTAIN >
```

Символи модифікації (?, +, *) можна помістити після дужок. Наприклад, наступне оголошення допускає включення одного або декількох дочірніх елементів будь-якого з цих трьох типів у будь-якому порядку:

```
<!ELEMENT FILM (STAR | NARRATOR | INSTRUCTOR )+>
```

Таке оголошення робить валідним всі такі елементи:

```
<FILM >
   <NARRATOR>Bertram Wooster</NARRATOR>
   <STAR>Sean Connery</STAR>
   <NARRATOR>Plug Basham</NARRATOR>
</FILM >
<FILM >
   <STAR>Sean Connery</STAR>
   <STAR>Meg Ryan</STAR>
</FILM >
<FILM >
   <INSTRUCTOR>Stinker Pike</INSTRUCTOR>
</FILM >
```

Для формування більш складнішого вмісту можна вкладати одну модель у іншу. Наприклад, наступне DTD задає, що кожен елемент FILM повинен мати один дочірній елемент TITLE, за ним повинен слідувати один дочірній елемент CLASS, після нього повинні йти один із дочірніх елементів STAR, NARRATOR або INSTRUCTOR.

```
<!DOCTYPE FILM
   [
   <!ELEMENT FILM (TITLE, CLASS, (STAR | NARRATOR |      INSTRUCTOR ) )>
   <!ELEMENT TITLE (#PCDATA )>
   <!ELEMENT CLASS (#PCDATA )>
   <!ELEMENT STAR (#PCDATA )>
   <!ELEMENT NARRATOR (#PCDATA )>
   <!ELEMENT INSTRUCTOR (#PCDATA )>
   ]
>
```

Якщо елемент має змішаний вміст, він може включати і дочірні елементи, і символьні дані. За умови змішаного вмісту можна задавати типи дочірніх елементів, але не порядок або кількість входжень.

Щоб оголосити тип елемента, який може містити символьні дані і на додаток жодного або декілька дочірніх елементів, кожен тип дочірнього елемента перераховується після ключового слова #PCDATA і розділяється символами "|". У кінці поміщається *. Кожне ім'я елемента може з'являтися у оголошенні тільки один раз.

Наприклад, наступне оголошення указує, що елемент TITLE може містити символьні дані плюс жодного або декілька дочірніх елементів SUBTITLE:

```
<!ELEMENT TITLE (#PCDATA | SUBTITLE )*>
```

Відповідно до цього оголошення такі елементи TITLE є допустимими:

```
<TITLE >
   Moby-Dick
   <SUBTITLE>Or, the Whale</SUBTITLE>
</TITLE >
<TITLE >
   <SUBTITLE>Or, the Whale</SUBTITLE>
   Moby-Dick
</TITLE >
<TITLE >
   Moby-Dick
</TITLE >
<TITLE >
   <SUBTITLE>Or, the Whale</SUBTITLE>
   <SUBTITLE>Another Subtitle</SUBTITLE>
</TITLE >
<TITLE></TITLE>
```

У валідному XML-документі необхідно оголосити всі атрибути, які передбачається використовувати для елементів. Атрибути, що асоціюються з певним елементом, визначаються за допомогою спеціального типу DTD-розмітки, званого оголошенням списку атрибутів.

Воно:

визначає імена атрибутів, що асоціюються з елементом;

встановлює тип даних кожного атрибута;

задає обов'язковість для кожного атрибута.

Якщо атрибут необов'язковий, у оголошенні списку атрибутів указується, що повинен чинити процесор, якщо атрибут опущений. У оголошенні

повинно, наприклад, міститися значення атрибута за замовчуванням, яке використовуватиме процесор.

Оголошення списку атрибутів має таку загальну форму:

<!ATTLIST Ім'я ВизАт>

Тут "Ім'я" є ім'я елемента, що асоціюється з атрибутом або атрибутами. "ВизАт" – це одне або декілька значень атрибутів, кожне з яких визначає один атрибут.

Визначення атрибута має таку форму запису:

Ім'я ТипАтр ОгУмов

Тут "Ім'я" є ім'ям атрибута.

"ТипАтр" є типом атрибута, тобто види значень, які можуть бути привласнені атрибута.

"ОгУмов" – це оголошення за замовчуванням, яке указує на обов'язковість атрибута і містить іншу інформацію.

Припустимо, оголошений тип елемента з ім'ям FILM таким чином:

```
<!ELEMENT FILM (TITLE (STAR NARRATOR INSTRUCTOR ) )>
```

Ось приклад оголошення списку атрибутів, яке описує два атрибути – Class і Year – для елемента FILM:

```
<!ATTLIST FILM Class CDATA "fictional " Year CDATA #REQUIRED >
```

Атрибуту Class можна привласнити будь-який рядок у лапках (ключове слово CDATA). Якщо атрибут для певного елемента буде опущений, йому буде автоматично привласнено значення за замовчуванням "fictional". Атрибута Year можна привласнити будь-який рядок у лапках; однак, це повинно бути зроблено обов'язково для кожного елемента FILM (ключове слово #REQUIRED), тому він не має значення за замовчуванням.

Наступний повний XML-документ містить оголошення списку атрибутів, а також елемент FILM:

```
<?xml version="1.0" encoding="windows-1251 " ?>
<!DOCTYPE FILM
   [
   <!ELEMENT FILM (TITLE, (STAR | NARRATOR | INSTRUCTOR ) )>
   <!ATTLIST FILM Class CDATA "fictional " Year CDATA #REQUIRED >
   <!ELEMENT TITLE (#PCDATA )>
   <!ELEMENT STAR (#PCDATA )>
   <!ELEMENT NARRATOR (#PCDATA )>
   <!ELEMENT INSTRUCTOR (#PCDATA )>
   ]
>
<FILM Year="1948 ">
```

<TITLE>The Morning After</TITLE>
<STAR>Morgan Attenbury</STAR>
</FILM >

Якщо використовується для одного елемента більш за одне оголошення списку атрибутів, зміст оголошень об'єднується. Якщо атрибут із заданим ім'ям оголошений для одного і того ж елемента кілька разів, перше оголошення використовується, а подальші – ігноруються.

Усього існує шість можливих типів значень атрибута:

CDATA – вмістом можуть бути будь-які символьні дані;

ID – визначає унікальний ідентифікатор елемента у документі;

IDREF (IDREFS) – указує, що значенням атрибута повинна виступати назва (або декілька таких назв, розділених пропусками) унікального ідентифікатора визначеного у цьому документі елемента;

ENTITY (ENTITIES) – значення атрибута має бути назвою (або списком назв, якщо використовується ENTITIES) компоненту (макровизначення), визначеного у документі;

NMTOKEN (NMTOKENS) – вмістом елемента може бути тільки одне окреме слово (тобто цей параметр є обмеженим варіантом CDATA);

список допустимих значень – визначається список значень, які може мати даний атрибут.

Також у оголошенні атрибута можна використовувати такі параметри:

#REQUIRED – визначає обов'язковий атрибут, який має бути заданий у всіх елементах даного типу;

#IMPLIED – атрибут не є обов'язковим;

#FIXED "значення" – указує, що атрибут повинен мати лише вказане значення, однак само визначення атрибута не є обов'язковим, але у процесі розбору його значення у будь-якому випадку буде передано програмі-аналізатору;

значення – задає значення атрибута за замовчуванням.

Розглянуті вищі визначення (правила) DTD задають структуру і дані XML-документів. Якщо формально вони не задані, то для аналізу можна застосовувати лише спеціально розроблене програмне забезпечення, у якому враховані ці (фактичні) правила, тобто документи прив'язані до конкретних програм.

За наявності DTD для перевірки можуть використовуватися універсальні програми-аналізатори. Крім того, різні застосування можуть спільно використовувати DTD для забезпечення єдиного погляду на XML-документ. У цьому випадку визначення повинні зберігатися окремо і бути доступними.

Аналізатори вбудовані у більшість браузерів. Проте браузери, подібно до *Internet Explorer*, вимагають, щоб XML-документи були формально коректними, але не обов'язково дійсними (валідними). Вони не потребують обов'язкової наявності визначень DTD, але якщо ті присутні, то можуть бути використані у процесі перевірки XML-документа. Існують і спеціальні модулі перевірки валідності документів.

Модулі перевірки валідності – це пакети, які перевіряють XML-код і виводять інформацію про результати перевірки. Перелік таких програм можна знайти у Інтернеті.

У XML користувач сам створює свої елементи. Це означає, що для відображення цих елементів слід вказати браузеру, як це зробити. Для цього запропоновано використовувати специфікації каскадних таблиць стилів (*Cascading Style Sheet*, CSS). Однак CSS лише встановлює формат і розміщення елементів. Тому у рамках розвитку технологій на основі XML були запропоновані рішення, які істотно розширяють можливості з оброблення і перетворення документів. Це розширювана мова таблиць стилів (*Extensible Styleheet Language*, XSL) і спеціальний додаток XSLT (*XSL Transformations*).

XSLT – це XML-додаток (мова розмітки), що визначає правила, відповідно до яких один XML-документ перетвориться у інший. XSLT-документ, тобто трансформуюча таблиця стилів XSLT, містить шаблони (опис форматування). XSLT-процесор порівнює елементи вхідного XML-документа із шаблонами у таблиці стилів. Коли відповідний шаблон знайдений, процесор записує вміст шаблону у вихідне дерево. Після закінчення цього етапу вихідне дерево записується або у XML-документ, або у інший формат, такий як звичайний текст або HTML.

Таким чином, XSLT застосовується для оброблення документів (фільтрація даних, внесення додаткової розмітки і тому подібне). Ця мова забезпечує доступ до вмісту XML-документів, а також позволяє створювати на їх основі нові документи. Одне дерево перетвориться у інше, одна розмітка – у іншу. Документ може бути перетворений у інший формат (HTML, спеціальна розмітка, звичайний текст).

Після того, як таблиця XSLT задана, можна перетворити початковий документ за допомогою XSLT-процесора. XSLT-процесор – це програма, яка читає таблицю стилів XSLT, читає вхідний XML-документ і перетворить вхідний документ у вихідний відповідно до інструкцій, даних у таблиці стилів.

XSLT-процесор може бути вбудованим у браузер, у WEB-сервер або у сервер додатків, а може бути окремою програмою.

Таблиця XSLT є коректно сформованим XML-документом, має XML-оголошення, може містити оголошення типу документа (DTD), хоча у більшості таблиць стилів його немає. Кореневим елементом цього документа є або *stylesheet*, або *transform*. Це синоніми, між ними немає жодної різниці, окрім імені. Вони обидва мають однаковий набір можливих дочірніх елементів і атрибутів, і обидва означають для XSLT-процесора одне і те ж. XSLT-документ може виглядати так:

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<!-- один або декілька елементів шаблону...-->
</xsl:stylesheet >
```

Елемент **xsl:stylesheet** ідентифікує документ як XSL-таблицю стилів і слугує контейнером для інших елементів. У ньому також визначається простір імен xsl. Усі імена, що мають відношення до таблиці повинні мати префікс "xsl:".

Файли, що містять XSLT, мають розширення xsl. XML-документи, що підлягають обробленні з використанням таблиць стилів, повинні мати у пролозі інструкцію оброблення *xml-stylesheet*, яка указує, де знаходиться відповідна документу таблиця стилів. Якщо це таблиця стилів XSLT, то атрибут *type* повинен мати значення text/xml (на відміну від text/css для CSS). Наприклад,

```
<?xml-stylesheet type="text/xml" href="http://www.ore.com/styles/people.xsl"?>
```

Ця інструкція говорить про те, що слід застосовувати таблицю стилів, що знаходиться за абсолютною URL-адресою. Значенням атрибута href може бути і відносна адреса.

Для управління вихідними даними, що формуються на основі початкового XML-документа, у таблицю стилів XSLT включають шаблони. Кожен шаблон представлений елементом *xsl:template*. Цей елемент має атрибут *match*, який вказує на певну гілку дерева структури документа (атрибут *match* аналогічний селектору у правилі CSS). Значення атрибута *match* має назву зразка (*pattern*). Наприклад, шаблон, який містить інструкції для відображення всього XML-документа:

```
<xsl:template match="/">
<!-- дочірні елементи... -->
</xsl:template >
```

Зразок у даному шаблоні (символ "/") представляє весь XML-документ.

Інший простий зразок – це ім'я елемента, наприклад, <xsl:template match="person">Людина</xsl:template>. Даний шаблон свідчіть про те, що

кожного разу, коли зустрічається елемент *person*, процесор таблиць стилів повинен генерувати текст "Людина".

Складніші конструкції для зв'язування шаблону з гілкою дерева записуються за допомогою спеціальної мови XPath.

Кожна XSL-таблиця стилів повинна містити тільки один шаблон із атрибутом *match*, який має значення "/". Крім того, можна також включити один або декілька додаткових шаблонів з інструкціями для відображення певних підлеглих гілок у структурі XML-документа.

За замовчуванням XSLT процесор читає вхідний XML-документ від початку до кінця, починаючи із кореня документа і рухаючись вниз деревом. Для кожного рівня перевіряється наявність шаблону, якщо він виявлений, то активізується для кожного знайденого вузла і на цьому робота закінчується. Шаблон, відповідний батьківському елементу, активізується раніше шаблонів, відповідних дочірнім елементам. Тому за наявності шаблону батьківського документа обхід закінчується без активації шаблонів дочірніх елементів. Таким чином, за наявності декількох шаблонів активізується лише шаблон самого старшого батька. Для активізації останніх слід використовувати спеціальні елементи.

Шаблон може містити послідовність елементів двох видів:

XML-елементи, що представляють незмінну коректну розмітку (наприклад, HTML-розмітку, а може бути яку-небудь іншу, у яку виконується перетворення);

XSL-елементи, які задають порядок використання даних із початкового документа (інструкції за вибором і модифікацією даних XML).

XSL-таблиця може виконувати на основі шаблонів перетворення у розмітку (XML-документ) будь-якого вигляду, у тому числі і у HTML. Для XML-документів, що перетворюються у HTML-розмітку, XSL-таблиця стилів повідомляє браузеру (або іншій програмі, що виконує перетворення), як відобразити XML-документ шляхом його вибіркового перетворення у блоки HTML-розмітки.

Приклад вставки незмінної розмітки:

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:template match="/">
      <HTML >
      <HEAD >
      <TITLE>pr10</TITLE>
      </HEAD >
```

```
      <BODY >
      <H2>Результат перетворення</H2>
      </BODY >
      </HTML >
 </xsl:template >
</xsl:stylesheet >
```

Якщо відкрити у браузері XML-документ із посиланням на таку XSLT, то буде відображений HTML-документ, що міститься у шаблоні.

Для зміни порядку використання даних і оброблення існують спеціальні елементи (елементи трансформації). Програми обробки відрізняють ці елементи за префіксом xsl. Основні елементи для перетворення (всього їх близько 30) наведені у табл. 1.

Вказівка на певний XML-елемент, з яким виконуватимуться перетворення, проводиться завданням значення атрибута *select.* Це може бути шлях деревом або вираз XPath. Під час задавання шляху проводиться орієнтація на уявлення структури документа у вигляді дерева ("/" – кореневий елемент), з урахуванням поточного місцеположення, що задається атрибутом match у xsl:template або select у xsl:for-each. Можна використовувати спеціальні символи ("**.**" і " ***** "). Крапка – поточний елемент, зірочка – будь-який елемент.

Таблиця 1

Елементи трансформації

Елемент	Призначення	Атрибути
<xsl:template>	опис шаблону	match
<xsl:apply-templates>	вставка даних за шаблоном	select
<xsl:value-of>	вставка значень з елемента	select
<xsl:for-each>	повторення обробки	select, order-by
<xsl:sort>	сортування	select, order
<xsl:attribute>	установка значень атрибутів	name

Якщо дані знаходяться у атрибутах, то посилання на них виглядає так *путь_к_элементу/@имя_атрибута*.

Розглянемо вживання конструкцій XSLT для перетворення XML-документа, якій містить дані про студентів та книги, у HTML.

Спочатку задамо шаблони й елементи apply-templates, які забезпечать обхід всіх вузлів STUDENT і BOOK:

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:template match="/">
        <HTML>
        <HEAD>
        <TITLE>pr1</TITLE>
        </HEAD>
        <BODY>
        <H2>Список студентів, які не здали книги</H2>
        <xsl:apply-templates select="DOCUMENT/STUDENT"/>
        </BODY>
        </HTML>
 </xsl:template>
 <xsl:template match="STUDENT">
        <xsl:apply-templates select="BOOK"/>
 </xsl:template>
 <xsl:template match="BOOK">
        </xsl:template>
</xsl:stylesheet>
```

Незмінна розмітка, яка відповідає початку HTML-документа (до тега BODY), поміщена у шаблон з атрибутом match="/". Далі розміщуються шаблони для відображення даних зі всіх вузлів документа STUDENT і BOOK. Шаблон з match="/" активізується автоматично. Для активізації інших у шаблони батьківських елементів включений елемент *apply-templates*. Він забезпечує активізацію і вставку даних відповідно до вказаного шаблону.

Ця таблиця стилів активізує шаблони для всіх вузлів документа. Проте браузер відобразить лише незмінну розмітку, інших даних у шаблонах немає.

Для вставки текстового вмісту тегів із початкового документа використовується елемент *value-of*. Атрибут *select* задає елемент початкового документа, з якого береться значення. Наступний фрагмент забезпечить відображення текстових даних:

```
 <xsl:template match="STUDENT ">
        <SPAN><xsl:value-of select="NAME"/></SPAN>
        <BR/>
        <xsl:apply-templates select="BOOK"/>
 </xsl:template >
 <xsl:template match="BOOK ">
        <SPAN><xsl:value-of select="AUTOR"/></SPAN>
        <SPAN><xsl:value-of select="TITLE"/></SPAN><BR/>
 </xsl:template >
</xsl:stylesheet >
```

Процесор вставляє текстову частину кожного тега, включаючи вміст вкладених (дочірніх) тегів, на місце елемента value-of.

Для перебору всіх вузлів заданого рівня можна використовувати не тільки активізацію шаблонів, а і спеціальний елемент for-each. Він повторює оброблення, яке визначене всередині для кожного вузла обраного рівня одного батька <xsl:for-each select= " ">... </xsl:for-each>.

Це аналог оператора циклу у мовах програмування:

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:template match="/">
 <HTML>
 <HEAD>
 <TITLE>pr1</TITLE>
 </HEAD>
 <BODY>
 <H2>Список студентів, які не здали книги</H2>
<xsl:for-each select="DOCUMENT/STUDENT">
 <H3><xsl:value-of select="NAME"/></H3>
</xsl:for-each>
 </BODY>
</HTML>
 </xsl:template>
</xsl:stylesheet>
```

За допомогою цієї таблиці буде виведено вміст тегів NAME зі всіх тегів STUDENT.

Елементи xsl:for-each можуть бути вкладеними (вкладені цикли):

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:template match="/">
 <HTML>
 <HEAD>
 <TITLE>pr1</TITLE>
 </HEAD>
 <BODY>
 <H2>Список студентів, які не здали книги</H2>
<xsl:for-each select="DOCUMENT/STUDENT">
  <xsl:for-each select="BOOK">
   <H3><xsl:value-of select="TITLE"/></H3>
  </xsl:for-each>
</xsl:for-each>
 </BODY>
</HTML>
```

```
 </xsl:template>
</xsl:stylesheet>
```

Будуть виведені тексти зі всіх тегів TITLE.

Елемент xsl:attribute дозволяє додати атрибут до елемента результуючого дерева. Має обов'язковий атрибут *name*, який задає ім'я атрибута, що створюється цією інструкцією. Вміст цього елемента є шаблоном (або елементом шаблону). Результатом виклику цього шаблону має бути простий текст. За допомогою цього елемента можна сформувати елемент із заданими атрибутами, наприклад, тег IMG:

```
<IMG >
<xsl:attribute name="src"><xsl:value-of select="photo"/></xsl:attribute>
</IMG >
```

Розглянуті елементи дозволяють перетворювати XML-документи у HTML для подальшого відображення у браузері.

Розглянуто найпростіше перетворення. Окрім цього можна сортувати і фільтрувати дані для виводу, використовувати спеціальні функції і тому подібне.

Література: [1; 4; 7].

Самостійна робота 3. Використання мови PHP

Мета роботи: отримання знань та навичок у створенні додатків із використанням мови PHP.

У результаті виконання самостійної роботи у студента формуються такі **компетентності:** здатність розробляти програми мовою PHP.

Результатом виконання роботи є розроблені і відлагоджені програми для кожного пункту та індивідуального завдання "Створення WEB-додатка за власним задумом". Тема індивідуального завдання обирається студентом самостійно за узгодженням з викладачем, орієнтуючись на типові завдання.

Завдання для самостійної роботи

Виконувати дану самостійну роботу доцільно під час вивчення теми "Мова програмування PHP". У ході роботи слід обрати одне зі списку завдань, узгодити його з викладачем та виконати таке.

1. Створення та оброблення асоціативних масивів.
2. Оброблення рядків.
3. Оброблення даних із форм.

Список типових завдань.

1. Створити додаток для заповнення та збереження на сервері анкети нового користувача

2. Створити додаток для входу на сайт із перевіркою імені та пароля.

3. Створити додаток для збирання статистики за віком відвідувачів.

4. Створити додаток для збирання відгуків користувачів про сторінку.

5. Створити додаток для збирання замовлень на товар.

6. Створити додаток для проведення опитування у мережі з обчисленням усереднених оцінок.

7. Створити додаток за власним задумом.

Контрольні запитання для самодіагностики

1. Сформулюйте відмінності у синтаксисі PHP і знайомих вам мов C і JavaScript.

2. Що таке асоціативні масиви, як вони використовуються?

3. Назвіть способи отримання доступу до даних із форм у PHP.

4. Сформулюйте правила оформлення функцій у мові PHP. .

5. У чому різниця у використанні одинарних і подвійних лапок?

6. Як можна задати у PHP текст великого розміру?

7. Які засоби можна використовувати для оброблення асоціативних масивів?

8. У чому різниця між відправкою даних методами GET і POST?

Методичні рекомендації до теми

1. Робота з асоціативними масивами

Традиційно під масивом розуміють упорядковану сукупність однотипних об'єктів, доступ до яких здійснюється на ім'я масиву із зазначенням порядкового номера (індексу).

```
$p=$ar[3];
$ar[3]=23;
```

У PHP масиви можуть створюватися з використанням спеціальних функцій, шляхом послідовного присвоєння значень елементів або перетворення даних з інших типів. Нумерація елементів починається з 0, індекси або задаються, або формуються автоматично.

Найчастіше використовується функція *array ().* Вона отримує у якості параметрів список значень елементів і повертає масив, що складається із зазначених елементів.

```
$lang = array ("English", "Gaelic", "Spanish");
// $lang[0] = "English"
// $lang[1] = "Gaelic"
// $lang[2] = "Spanish"
```

Масив можна сформувати, присвоювання значення послідовним елементам:

```
$lang[ ] = "Spanish";  // $lang[0] = "Spanish"
$lang[ ] = "English";   // $lang [l] = "English";
$lang[ ] = "Gaelic";     // $lang[2] = "Gaelic";
```

У цьому випадку нові елементи додаються без явної вказівки індексу. Кожен новий елемент додається у позицію, рівну довжині масиву плюс 1. Нові елементи можна додавати і у конкретну позицію масиву. Для цього вказується індекс нового елемента:

```
$languages[15] = "Italian";
$languages[22] = "French";
```

Масив можна створити шляхом перетворення рядків із використанням регулярних виразів (функції *split(), preg_split()*) або простого розбиття рядка на частини (функції *explode(), str_split*(), і ін.). Наприклад, так буде сформований масив із трьох елементів.

```
$st="English Gaelic Spanish";
$lang=explode(" ", $st);
```

Асоціативний масив – це масив, у якому у якості індексу використовується рядок символів, так званий ключ. Під час формування таких масивів необхідно ставити пари "ключ" => "значення"

```
$lang=array("Spain"=>"Spanish","France"=>"French");
```

або присвоювати значення послідовно

```
$lang["Spain"] = "Spanish";
$lang["France"] = "French";
```

Розглянемо дії, що виконуються з асоціативними масивами.

Додавання елементів масиву. Розмірність масивів у PHP змінюється динамічно. Щоб додати елемент досить виконати привласнення:

```
$lang["England"] = "English";
```

Елемент буде додано у кінець масиву. Слід зазначити, що елементи асоціативних масивів упорядковані відповідно до надходження значень, саме у цьому порядку вони будуть проглядатися у ході оброблення.

Видалення елементів масиву. Для видалення елемента слугує функція *unset ():*

```
unset($lang["England"]);
```

Заміна ключів значеннями. У парі "ключ" => "значення" для масиву, ключ і значення можна міняти місцями. Наприклад, якщо необхідно

створити новий масив, у якому індексами будуть назви мов, а значеннями – країни, у яких вони використовуються, то застосовується функція *array_flip()*:

```
$stat = array_flip($lang);
```

Злиття масивів. Кілька асоціативних масивів можна об'єднати у один за допомогою функції *array_merge()*. наприклад,

```
$stat_lang = array_merge($stat, $lang);
```

Якщо у вихідних масивах є елементи, які мають однакові рядкові ключі, тоді кожне наступне значення буде замінювати попереднє.

Сортування масивів. Сортувати асоціативні масиви можна як за ключами, так і за значеннями. Для цього використовуються різні функції.

Функція *sort()* сортує масив по зростанню значень (*rsort()* – за спаданням, *shuffle()* – розташує елементи у довільному порядку). Однак після сортування асоціативні ключі будуть замінені числовими індексами. Для збереження ключів необхідно використовувати функцію *assort().* Функція *array_reverse()* змінює порядок проходження елементів на зворотний.

Для сортування елементів асоціативних масивів за ключами використовуються функції *ksort()* і *krsort()* (за зростанням і зменшенням, відповідно).

Для вивчення і розуміння результатів дії з масивами зручно користуватися таким фрагментом програми.

```
echo "<pre>";
print_r($stat_lang);
echo "</pre>";
```

Елементи масиву виводяться у такому вигляді.

```
Array
(
   [Spanish] => Spain
   [Spain] => Spanish
   [French] => France
   [France] => French
   [English] => England
   [England] => English
)
```

Перевірка елементів у масиві. За допомогою функції *in_array()* можна визначити, чи є у масиві елемент із заданим значенням. Наприклад, in_array("Spain", $ stat_lang) поверне *true*, якщо елемент є, і *false* – у протилежному випадку:

```
if (in_array("Spain", $stat_lang))
{
  echo "Exists!";
```

```
} else {
  echo "Not exist!";
}
```

За допомогою функції *array_key_exists()* перевіряється наявність елемента із заданим ключем:

```
if (array_key_exists("England", $stat_lang))
{
  echo "Key exists!";
} else {
  echo "Key does not exist!";
}
```

За допомогою функції *array_search()* можна знайти ключ елемента із заданим значенням

```
$state = array_search("England", $stat_lang);
```

Перебір елементів масиву. Для роботи з масивами зручно використовувати спеціальні функції, такі *як list(), each(), foreach()*.

Функція *list()* забезпечує одночасне присвоювання значень, витягнутих з масиву, відразу декільком змінним, які включені до списку параметрів. Працює тільки для елементів з цілочисельними індексами. наприклад,

```
$lang=array("Spain","Spanish","France","French");
list($a1,$a2,$a3,$a4)=$lang;
echo 'a1 – ',$a1;
echo ' a2 – ',$a2;
echo ' a3 – ',$a3;
echo ' a4 – ',$a4;
```

Результат буде такий:

```
a1 – Spain a2 – Spanish a3 – France a4 – French
```

Функцію *list()* можна використовувати і у ході читання даних з текстового файла. Наприклад, для читання і виведення рядків з розбивкою їх на структурні частини по символу-роздільника.

```
while ($line = fgets ($user_file. 4096));
list ($name, $occupation, $color) = split( "|", $line); // розділити рядок на частини
print "Name: Sname< br>";
print "Occupation: Soccupation <br>";
print "Favorite color: Scolor <br>";
endwhile;
```

Кожен рядок файла читається, форматується і виводиться у вихідний потік.

Функція *each()* отримує у якості параметра ім'я масиву, а повертає масив, що містить ключ і значення поточного елемента, покажчик поточного

елемента зсувається на наступний елемент. Повертається масив містить чотири елементи: два з цілочисельним ключем (0 і 1) і два з асоціативним (*key* і *value*). наприклад,

```
$lang=array("Spain"=>"Spanish","France"=>"French");
$a=each($lang);
echo "<pre>";
print_r($a);
echo "</pre>";
```

У результаті буде сформований масив такого вигляду:

```
Array
(
   [1] => Spanish
   [value] => Spanish
   [0] => Spain
   [key] => Spain
)
```

За допомогою циклу *while.* Це може бути корисним для перебору елементів масиву. Наступний фрагмент забезпечить висновок елементів у тому ж форматі, що і *print_r().*

```
$lang=array("Spain"=>"Spanish","France"=>"French");
while (list($k, $v) = each ($lang)) {
   echo "[$k] => $v <br>";
}
```

Результат:

```
[Spain] => Spanish
[France] => French
```

Можна використовувати й асоціативні ключі.

```
$lang=array("Spain"=>"Spanish","France"=>"French");
while ($a = each ($lang)) {
   echo "[$a[key]] => $a[value] <br>";

}
```

Результат буде той же.

Конструкція *foreach()* спеціально призначена для перебору елементів масивів і властивостей об'єктів. Її можна використовувати у двох варіантах: *foreach ($arr as $value)* і *foreach ($arr as $key => $value)*. У першому випадку на кожному кроці витягується і присвоюється змінній *$value* значення чергового елемента масиву *$arr*. У другому – змінним *$value* і *$key* присвоюються значення і ключ чергового елемента масиву *$arr* відповідно.

Наприклад, друк ключа і значення для кожного елемента асоціативного масиву.

```
$lang=array("Spain"=>"Spanish","France"=>"French");
foreach ($lang as $k =>$v) {
   print "[$k] =>  $v <br>\n";
}
```

2. Оброблення рядків.

Рядок – це набір символів, який розглядається як один об'єкт і може присвоюватися змінним, передаватися у якості параметрів у функції, повертатися з функцій або відправлятися у вихідний потік.

Рядки можуть задаватися декількома способами, кожен із яких має ряд особливостей.

Рядок можна задати з використанням подвійних лапок, наприклад:

```
$my_string = "строка с \"переменными\" $count";
```

За умови такого задавання імена змінних усередині рядка замінюються значеннями. Для відображення спеціальних символів (наприклад, лапок) необхідно перед ними записувати символ /.

Під час задавання рядків із використанням одинарних лапок імена змінних залишаються просто послідовністю символів, без оброблення. Наприклад, у результаті виконання такого фрагмента:

```
$count = 13;
$string_1 = "Встроке \"Hello, world!\" $countсимволов.<br>";
$string_2 = 'Встроке \'Hello, world!\' $count символов.<br>';
echo $string_1;
echo $string_2;
```

побачимо у вікні браузера два рядки:

```
В строке "Hello, world!" 13 символов
В строке 'Hello, world!' $count символов
```

У деяких ситуаціях може виявитися, що інтерпретатор не здатний коректно обробити рядок. Наприклад, якщо ім'я змінної не відокремлено від продовження рядка або необхідно підставити у рядку не значення простої змінної, а значення виразу. У цих випадках необхідно ім'я або вираз, що підлягає обробленню, укладати у фігурні дужки ({}).

```
$sport_1 = 'волей';
$sport_2 = 'фут';
// Неправильные конструкции
$play_1 = "Я люблю играть у $sport_1бол.";
$play_2 = "Я люблю играть у $sport_2бол.";
// Правильные конструкции
$play_1 = "Я люблю играть у {$sport_1}бол.";
$play_2 = "Я люблю играть у {$sport_2}бол.";
```

Ця запис слугує для інтерпретатора PHP вказівкою на те, що перед підстановкою значень необхідно обчислити тільки значення виразу з ім'ям змінної, укладеними у фігурні дужки.

На відміну від рядків у одинарних і подвійних лапках, у мові PHP передбачений ще один спосіб задавання рядка, так звана синтаксична структура вкладеного документа (*Heredoc*). Подібна синтаксична конструкція є зручним засобом задавання великих фрагментів тексту.

Знаком операції, що застосовується у синтаксичній структурі вкладеного документа, є (<<<). За цим знаком повинна безпосередньо слідувати мітка (без лапків), яка позначає початок багаторядкового тексту. Мітка може мати будь-яке ім'я, відповідне звичайними правилами іменування змінних у мові PHP. Інтерпретатор PHP продовжує включати до складу значення змінної такі рядки до тих пір, поки знову не з'явиться та ж мітка, що й на початку рядка. За заключною міткою може слідувати необов'язкова крапка з комою.

Наприклад,

```
$string = <<<EOT
<form method="post" action="{$_SERVER['PHP_SELF']}">
<input type="text" name="login" placeholder="Введитеимя..."><br>
   <input type="password" name="password" placeholder="Введите
пароль..."><br>
   <input type="submit" value="Отправить">
</form>
EOT;
echo $string;
```

Цей фрагмент програми (виводить найпростішу HTML-форму), забезпечує присвоєння змінної *$string* рядка, що містить HTML-код. Зручною особливістю є те, що у цьому тексті можуть бути присутніми знаки лапок (й інші спеціальні символи).

Символи та індекси символів у рядках. На відміну від деяких інших мов програмування у мові PHP немає окремого символьного типу, яка не збігається з строковим типом. У мові PHP символ – це рядок із довжиною 1. Вибірка окремих символів із рядка може здійснюватися шляхом вказівки порядкового номера символу, який починається з нуля, і який повинен бути зазначений у фігурних дужках безпосередньо за ім'ям строкової змінної. Наприклад, фрагмент програми, наведений нижче, робить вибір з рядка і виведення у вихідний потік по одному символу тексту.

```
$my_string = "Doubled";
for ($index = 0; $index <strlen($my_string); $index++)
```

```
{
   $char = $my_string{$index};
        print("$char");
}
```

Функція *strlen()* повертає довжину рядка.

Операції з рядками. У мові PHP передбачені дві рядкові операції: операція конкатенації (зчеплення рядків, позначається крапкою) і операція конкатенації з привласненням (позначається . =). Наприклад,

```
$string_1 = "Це частина";
$string_2 = "строки";
// Конкатенація строк
echo $string_1." простий ".$string_2."<br>";     // "Це частина простої строки"
// Конкатенація з привласненням
$string_1 .= " простий ";      // Еквівалентно $string_1 = $string_1." простий ";
$string_1 .= $string_2;
echo $string_1;              // " Це частина простої строки "
```

Слід зазначити, що у наведеному далі прикладі першому оператору *echo* не передається кілька строкових фактичних параметрів; передається тільки один строковий фактичний параметр, створений у результаті конкатенації чотирьох рядків. Перший і третій рядки задані за допомогою змінних, а другий і четвертий рядки є рядками, укладеними у подвійні лапки.

Строкові функції. У мові PHP передбачена велика кількість різноманітних функцій для оброблення і перетворення рядків. Розглянемо кілька найбільш часто використовуваних функцій.

Довжина рядка. Для визначення довжини рядка використовується функція *strlen()*. Однак необхідно враховувати, що ця функція повертає довжину рядка у байтах, а не у числі символів. Для визначення числа символів необхідно враховувати кодування. Наприклад, за допомогою функції *mb_strlen()*

```
$rus_str="строка";
echo $rus_str." – ".mb_strlen($rus_str, 'UTF8')." символов";
```

Аналіз вмісту рядка. Часто виникає задача знаходження символу або підрядка у рядку. Для цього є група функцій, наприклад, *strpos()* і *mb_strpos()*. Перша дозволяє знайти номер позиції конкретного символу або початку підрядка у рядку у байтах, друга – у символах. Наприклад, фрагмент програми

```
$str = "Привет world!";
echo "Символ 'l': ".strpos($str, 'ld');
echo "<br>";
echo "Символ 'l': ".mb_strpos($str, 'ld');
```

визначити початок підрядка у байтах і у символах

Символ 'l': 16

Символ 'l': 10

Порівняння рядків. Під час порівняння рядків можна використовувати операції порівняння на рівність (== або ===) з урахуванням їх особливостей (перша перед порівнянням призводить типи, друга порівнює без перетворення). Однак більш надійним є порівняння рядків за допомогою функції *strcmp().* Рядки порівнюються побайтно. Якщо перший рядок менше другий, повертається негативне число, якщо другий рядок менше першої – позитивне, якщо рядки ідентичні – нуль.

Функція *strcasecmp()* виробляє таке ж порівняння, але без урахування регістру.

Вибірка підрядка. Функції *substr()* і *mb_substr()* повертають рядок, що містить частину послідовності символів зі старої рядка. Наприклад,

mb_substr($str, 7, 3, 'UTF8');

У рядку $str виділяється підрядок з трьох символів із початковим символом на 7 позиції, і повертається у якості результату.

Функція *str_replace()* дозволяє замінити всі входження заданого підрядка інший рядком. Функція приймає три параметри: рядок, у якому повинен бути виконаний пошук, підрядок, що підлягає заміні, і рядок, який повинен застосовуватися для заміни. Заміна виконується стосовно всіх входженням підрядка, знайденим у рядку пошуку.

Функція *substr_replace()* вибирає частину, що підлягає заміні, за її абсолютною позицією. Ця функція приймає до чотирьох параметрів: рядок, у якому повинна бути виконана заміна, рядок, що застосовується у якості заміни, початкова позиція заміни і довжина замінної частини рядка.

Функція *strstr()* витягує частину рядка від збігається позиції до кінця рядка. Ця функція створена для зручності, щоб замінювати комбінацію *strpos()* і *substr()*. Два наступних вираження повністю еквівалентні:

```
$domain = strstr($email, "@");
$domain = substr($email, strpos($email, "@"));
```

Функції виводу на зовнішній пристрій і у рядок. Основними конструкціями, що застосовуються для виведення, є *print* і *echo*, які докладно розглядалися раніше. Стандартний спосіб виведення значень змінних на зовнішній пристрій складається у тому, щоб включити імена цих змінних у рядок з подвійними лапками (при обробці якої інтерпретатором відбувається підстановка значень змінних), а потім передати цей рядок у конструкцію *print* або *echo*.

Якщо потрібно ще точніше відформатований вивод, то можна скористатися наданими мовою PHP функціями *printf()* і *sprintf().* Ці дві функції приймають однакові параметри: спеціальну рядок формату, за якою слід довільну кількість інших параметрів, підставляється у потрібні місця у рядку формату для отримання результату. Єдина відмінність між функціями *printf()* і *sprintf()* полягає у тому, що перша відправляє результуючий рядок безпосередньо на зовнішній пристрій, що застосовується для виведення, а друга повертає результуючий рядок у якості результату свого виконання.

3. Оброблення даних з форм

Основне призначення форм полягає у передачі серверу інформації від користувача. Наприклад, для реєстрації чи збирання особистих даних, даних різних опитувань тощо. У них можна вводити текст або вибирати відповідні варіанти зі списку, а також встановлювати значення різних перемикачів. Дані, записані у форму, відправляються для оброблення на сервері. Залежно від введених значень сервер може формувати різні WEB-сторінки, відправляти запити до бази даних, запускати різні додатки і т. д.

Найбільш важлива особливість функціонування будь-якої технології, що забезпечує зв'язок клієнта з сервером, полягає у тому, що сам протокол HTTP характеризується відсутністю підтримки станів. Це означає, що кожен запит HTTP (який у більшості випадків зводиться до вимоги на отримання і доставку окремого ресурсу, такого як HTML-сторінка, таблиця стилів тощо) є незалежним від усіх інших запитів, не включає будь-яку інформацію про ідентифікації клієнта і не залишає сліду на сервері.

Під час створення форм для даних передбачена велика кількість різноманітних елементів. Відправка даних форми відбувається після натискання кнопки типу *submit*. Як приклад, розглянемо форму для збирання даних про інтереси відвідувача.

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8">
<title>Основи PHP</title>
</head>
<body>
<form action="sports.php" method="get">
  <fieldset>
    <legend>Оберіть вид спорту</legend>
Имя <Br>
<input type=text name="first_name">
```

```
<br>E-mail
<br><input type=text name="email">
<br>Улюблений спорт
<br>
    <select name="sport">
      <option value="Футбол">Футбол</option>
      <option value="Баскетбол">Баскетбол</option>
      <option value="Хокей">Хокей</option>
      <option value="Волейбол">Волейбол</option>
      <option value="Бальні танці">Бальні танці</option>
    </select><br><br>
<input type="submit" value="Відправити">
  </fieldset>
 </form>
</body>
</html>
```

Передача параметрів від клієнта до сервера виробляється одним із двох методів *GET* або *POST*.

Під час відправлення даних форми за допомогою методу *GET* вміст форми додається до *URL* після знаку у вигляді пар *ім'я = значення*, об'єднаних за допомогою символу &. Наприклад, для наведеної форми вона може мати вигляд:

http://localhost/sports.php? first_name=Виктор&email=ff%40hh.tt&sport=Футбол

З урахуванням особливостей подальшої обробки під час створення форми не варто використовувати у назвах або значеннях елементів символи "=" і "&", а також ці символи і символи кирилиці, у ідентифікаторах.

Взагалі можна створити рядок запиту у адресному рядку браузера і без форми, набравши вручну наведений вище рядок.

Метод *GET* зручний, якщо параметрів не багато, а також під час налагодження скриптів (тоді можна бачити значення й імена змінних, які передані).

У разі використання методу *POST* зміст форми кодується точно так же, як для методу *GET*, але сам запит відсилається блоком даних як частина операції *POST*. Цей метод рекомендується для передачі великих за обсягом блоків даних. Інформація, введена користувачем і відправлена на сервера за допомогою методу *POST*, подається на стандартне введення програмі, зазначеної у атрибуті *action*. Довжина блоку передається у змінній оточення CONTENT_LENGTH, а тип даних – у змінній CONTENT_TYPE (також, як це робиться при запуску CGI-програм). Під час використання *POST* користувач не бачить передані серверу дані.

У процесі відправлення даних на сервер будь-яким методом передаються не тільки самі дані, введені користувачем, але і ряд змінних, які характеризують клієнта, історію його роботи, шляхи до файлів тощо. Ось деякі з цих змінних:

REMOTE_ADDR – IP-адреса хоста (комп'ютера), що відправляє запит;

REQUEST_METHOD – метод, який був використаний під час відправлення запиту;

QUERY_STRING – інформація, яка знаходиться у URL після знаку питання;

SCRIPT_NAME – віртуальний шлях до програми, яка повинна виконуватися;

HTTP_USER_AGENT – інформація про браузер, який використовує клієнт.

Усередині PHP-скрипта існує кілька способів доступу до даних, які передані клієнтом згідно з протоколом HTTP. По-перше, можна використовувати асоціативні масиви $_POST і $_GET, звернення до значень яких проводиться за іменами змінних форми. Наприклад, якщо пара first_name=Віктор передана методом GET, то $_GET["first_name"] поверне "Віктор".

По-друге, для звернення до змінних, переданих за допомогою HTTP-запитів, існує спеціальний масив $_REQUEST. Він містить дані, передані методами POST і GET, а також за допомогою HTTP *cookies*. Його значення можна отримати, використовуючи у якості індексу ім'я відповідної змінної (елемента форми), значенням $_REQUEST["first_name"] буде "Віктор".

По-третє, за відповідними настройками (*register_globals=On* у файлі налаштувань) доступ до змінних форми може здійснюватися безпосередньо за їхніми іменами, тобто можна використовувати просто *$name*.

Для доступу до змінних оточення можна використовувати спеціальний масив $_SERVER або функцію *getenv()*. Наприклад, $_SERVER ["QUERY_STRING"] або getenv("QUERY_STRING") повернуть рядок запиту від клієнта, а $_SERVER["REMOTE_ADDR"] або getenv("REMOTE_ADDR") – IP-адресу користувача, який надіслав запит.

Як приклад оброблення даних із форм розглянемо скрипт, який обробляє введені користувачем дані у раніше наведеній формі та повертає їх клієнту.

```
<!-- Файл sports.php -->
<!DOCTYPE HTML>
<html>
```

```
<head>
<meta charset="utf-8">
<title>Улюблений від спорту</title>
</head>
<body>
<span><h1>
<?php
$sp=$_GET['sport'];
$fn=$_REQUEST['first_name'];
$em=$_GET['email'];
if  (empty($sp) or empty($fn) or empty($em)) echo " Ви ввели не всі дані <br>";
else echo "$fn , Ви любите $sp. Ми врахуємо це.<br>";
echo " Рядок запиту для вашої форми: {$_SERVER["QUERY_STRING"]}<br>";
$sip=getenv("REMOTE_ADDR");
echo "Запит відправлений з IP-адреса      $sip";
?>
</span></h1>
</body>
</html>
```

У прикладі використані різні засоби доступу до змінних форми і оточення.

Додаток, отримуючи дані від користувача, обов'язково повинен перевіряти їх коректність, не дивлячись на наявність аналогічної перевірки на стороні клієнта.

У прикладі далі показаний спосіб оброблення форми, що містить перевірку і передачу її результатів на клієнту.

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8">
<title>Основы PHP</title>
</head>
<body>
<?php
        $valid_name = '';   $valid_email = '';
        if (!isset($_POST['submit']))
        {
                $_POST['name'] = '';
                $_POST['email'] = '';
        }
        else
```

```
        {
                $name =($_POST['name']);
                $email =($_POST['email']);
                if (empty($name)) $valid_name='error';
                if (empty($email) || !filter_var($email, FILTER_VALIDATE_EMAIL)) $valid_email = 'error';
        }
?>
<form action="index.php" method="post">
 <fieldset>
    <legend>Контактна інформація</legend>
    <label for="name">ім'я</label>
    <input id="name" name="name" value="<?php echo $_POST['name']; ?>" class="<?php echo $valid_name; ?>"><br>
    <label for="email">Email</label>
    <input id="email" name="email" value="<?php echo $_POST['email']; ?>" class="<?php echo $valid_email; ?>"><br>
 </fieldset>
 <input type="submit" name="submit" value="Відправити інформацію">
</form>
</body>
</html>
```

У цьому прикладі під час отримання даних із форми (ім'я та адреса електронної пошти) перевіряється, чи не порожні поля і чи відповідає рядок у полі адреси структурі адреси електронної пошти. Під час першого звернення до сторінки перевірка пропускається.

Для перевірки порожнього поля використана функція *empty().*

Для перевірки значення адреси використана функція *filter_var()*. У функції два параметри – рядок і тип перевірки. Крім електронних адрес можна перевіряти URL, IP-адреси, коректність регулярних виразів і ряд інших. Повертає коректні дані або FALSE, якщо дані є некоректними. Наприклад,

```
If (!filter_var($email, FILTER_VALIDATE_EMAIL)) echo "Данные некорректны";
```

Література: [2; 3].

Самостійна робота 4. Створення WEB-додатків

Мета роботи: отримання знань та навичок у створенні WEB-додатків із використанням технології на базі мови PHP.

У результаті виконання самостійної роботи у студента формуються такі **компетентності:** здатність розробляти WEB-додатки мовою PHP.

Результатом виконання роботи є розроблені і відлагоджені додатки для кожного пункту та індивідуального завдання "Створення WEB-додатка за власним задумом". Тема індивідуального завдання вибирається студентом самостійно за узгодженням з викладачем, орієнтуючись на типові завдання.

Завдання для самостійної роботи

Виконувати дану самостійну роботу доцільно під час вивчення теми "Технології активних сторінок PHP". За кожним пунктом створити додаток на основі наведеного прикладу.

У ході роботи виконати таке.

1. Створити додаток для оброблення XML-документа.
2. Створити додаток для асинхронного запиту даних з сервера.
3. Створити додаток для збереження та використання даних за допомогою бази даних MySQL.
4. Розгорнути фреймворк та створити простіший додаток.

Контрольні запитання для самодіагностики

1. Назвіть засоби, що забезпечують обробку даних з XML-файлів, на PHP.
2. Поясніть що таке синхронна взаємодія з сервером.
3. Які переваги дає організація асинхронної роботи?
4. Що таке AJAX?
5. Що є ознакою успішного виконання асинхронного запиту?
6. Яка підтримка AJAX включена у jQuery? Назвіть імена функцій.
7. Сформулюйте послідовність дій для запису у базу даних.
8. Що таке фреймворк?
9. Сформулюйте дії з установки фреймворка Yii на сервері.
10. Сформулюйте послідовність дій зі створення додатка на основі фреймворка Yii.

Методичні рекомендації до теми

1. Засоби обробки XML-документів

Для роботи з XML-документами зручно використовувати розширення мови SimpleXML. Якщо у XML-документі є національні символи, наприклад, кирилиця, то завжди необхідно вказувати коректний XML-заголовок для правильної роботи SimpleXML, для кодування у windows <? Xml version = "1.0" encoding = "windows-1251" standalone = "yes"? >.

Незалежно від того, яке кодування вказано, після розбору SimpleXML видає результат у кодуванні UTF-8.

Усі частини XML-документа (вузли, або *Nodes*) перетворюються у екземпляри класу *SimpleXMLElement,* а всередині цього об'єкта список дочірніх вузлів і атрибутів поданий у вигляді звичайного масиву. Якщо дочірній елемент не містить вкладених елементів, а містить тільки значення, то він поданий у вигляді масиву, доступного як за номером, так і за назвою тега, а якщо він містить, у свою чергу, ще елементи, то це буде новий екземпляр класу *SimpleXMLElement*.

Для виконання найпростіших дій з оброблення XML-документа можна скористатися функціями:

simplexml_load_file() – створює у пам'яті об'єкт, відповідний XML-документу;

file_put_contents() – зберігає у файлі XML-документ;

var_dump() – виводить у вихідний потік структуру об'єкта у пам'яті;

children() – метод об'єкта *SimpleXMLElement* для доступу до вузлів;

addChild() – метод об'єкта *SimpleXMLElement* для додавання вузла.

Розглянемо оброблення на прикладі файла, що містить дані про ім'я та пароль користувача:

```
<?xml version="1.0"?>
<users>
  <user><login>MVP</login> <password>PMVP</password>  </user>
  <user> <login>MVP1</login> <password>PMVP1</password>  </user>
</users>
```

Дії з оброблення будуть складатися у запису даних реєстрації нового користувача і виведення списку зареєстрованих користувачів на екран.

Якщо дані передані методом GET з полів з іменами *log* і *pas*, то оброблення може бути виконана так:

```
$log=$_GET[log];
$pas=$_GET[pas];
$reg = simplexml_load_file("pr6.xml"); //створення об'єкта у пам'яті
$us=$reg->addChild("user");        // додавання тега user у кореневий елемент
$us->addChild("login",$log);       // додавання тега login з ім'ям
$us->addChild("password",$pas);// додавання тега password із паролем
file_put_contents('pr6.xml',$reg->asXML());//запис оновленого файлу
```

Для виведення списку паролів користувачів можна використовувати оператор *foreach*:

```
$reg = simplexml_load_file("pr6.xml");
foreach ($reg->user as $user) {  echo "$user->login – $user->password <br>";  };
```

Крім доступу до окремих вузлів і отримання їх значення можна отримати одразу вміст всього XML-документа або вміст всіх дочірніх елементів вузла. Для цього є метод *asXML()*, який повертає рядок із текстом.

Наприклад, *echo $reg->asXML()* виведе у вихідний потік вміст усього документа (тільки вміст, без тегів), а *echo $reg->user[1] -> asXML()* – вміст другого вузла <user>.

Підтримуються і більш розвинені можливості, наприклад, адресація елементів за допомогою мови XPath.

2. Асинхронна взаємодія з сервером

Технологія асинхронних запитів до сервера AJAX (*Asynchronous JavaScript and XML*) використовується у сучасних WEB-ресурсах дуже широко. Є підтримка у вигляді різних бібліотек у всіх технологічних рішеннях (ASP.NET, PHP і т.д.). Під час асинхронних запитів клієнт не чекає отримання відповіді, а продовжує роботу. Під час настання події приходу відповіді сервера отримані дані використовуються за призначенням. Перевагами такого підходу є відсутність перерв у роботі користувача.

Загальна схема взаємодії виглядає так: запит – очікування у фоновому режимі – отримання відповіді – завершення. Таким чином, AJAX – це підхід для прискорення роботи WEB-додатків за рахунок поєднання у часі очікування відповіді і роботи користувача.

Для реалізації AJAX використовується спеціальний об'єкт *XMLHttpRequest*. Це об'єкт JavaScript, і він створюється звичайним способом:

var xmlHttp = new XMLHttpRequest();

Цей об'єкт містить властивості та методи для управління всім процесом взаємодії з сервером.

Властивості об'єкта *XMLHttpRequest*:

onreadystatechange – ім'я обробника відповіді сервера;

readyState – код стану сервера;

status – код відповіді сервера;

responseText – відповідь сервера у вигляді рядка символів;

responseXML – відповідь сервера у вигляді XML-файла.

Методи об'єкта *XMLHttpRequest*:

open(тіп_запроса, url, спосіб) – створення запиту;

setRequestHeader (заголовок, значення) – додавання заголовку до запиту;

send('дані') – передача запиту на сервер.

У звичайних WEB-додатках користувачі заповнюють поля форм і натискають кнопку *Submit* (для відправки даних). Потім дані з форми передаються на сервер, сервер запускає обробник (зазвичай сценарій PHP, ASPNET або CGI-процес), а потім передає браузеру вихідний потік запущеного обробника, зазвичай нову сторінку. До тих пір, поки сценарій або програма на сервері виконується і не сформується нова сторінка, користувачі повинні чекати.

Під час асинхронної взаємодії дані з форми передаються у JavaScript-код, а не на сервер. Ці дані обробляються і формується запит до сервера. Очікування відповіді відбувається у фоновому режимі. Сторінка (форма) на екрані не блокується, а залишається доступною, користувач може навіть не помітити, що на сервер відправлений запит. Користувачі можуть продовжувати вводити дані, прокручувати сторінку і працювати з додатком.

Сервер передає дані (відповідь) назад у JavaScript-код, який виконає їх оброблення. Зі скрипта (обробника) можуть бути оновлені елементи сторінки, може бути переданий ще один запит, і все це без втручання користувача. У результаті забезпечується комфортність для користувача у поєднанні з високою інтерактивністю, разом з усіма можливостями Інтернет.

Дії, які необхідно виконати для реалізації AJAX за допомогою JavaScript:

створити об'єкт *XMLHttpRequest;*

виконати запит (отримати дані форми, передати їх на сервер, отримати відповідь);

оновити сторінку або її частину (з використанням отриманих даних HTML, XML і управління об'єктами DOM).

Створення об'єкта. З урахуванням відмінностей у об'єктних моделях браузерів створення об'єкта *XMLHttpRequest* у різних браузерах може відрізнятися.

Для Microsoft Internet Explorer різних версій рекомендується використовувати такий код:

```
var xmlHttp = false;
try {
  xmlHttp = new ActiveXObject("Msxml2.XMLHTTP");
} catch (e) {
  try {
    xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
  } catch (e2) {
    xmlHttp = false;
  }
}
```

Таким чином, забезпечується перевірка і використання двох версій об'єкта для Internet Explorer.

Для інших браузерів:

```
var xmlHttp = new XMLHttpRequest object;
```

Для кросбраузерності код створення об'єкта може виглядати так.

```
if (!xmlHttp && typeof XMLHttpRequest != 'undefined') {
  xmlHttp = new XMLHttpRequest();
}
```

Виконання запиту. Розглянемо коротко, як виглядає послідовність дій для виконання запиту й отримання відповіді під час асинхронної взаємодії з сервером:

зібрати дані з WEB-форми;

сформувати URL ресурсу, який повинен обробляти дані;

встановити з'єднання з сервером;

вказати функцію, яка виконається після отримання відповіді;

передати запит на сервер.

Усі ці дії можуть бути реалізовані у вигляді обробника події (наприклад, натискання кнопки у формі).

Обробка відповіді. У ході виконання запиту сервер передає клієнту інформацію про стан виконання запиту, яка записується у властивість *xmlHttp.readyState*.

Запит може знаходиться у одному з п'яти станів:

0 – запит не ініційовано;

1 – установлено підключення до сервера;

2 – запит отримано;

3 – відбувається оброблення запиту;

4 – оброблення запиту закінчена і відповідь готова.

Після закінчення оброблення у властивість *status* поміщається код завершення (200 – успішно).

Оброблення відповіді може полягати у перевірці готовності відповіді (*xmlHttp.readyState == 4* і *xmlHttp. Status == 200*). Під час виконання цих умов отримані дані (*xhttp.responseText*, якщо запитували текст) можуть використовуватися для зміни сторінки. Зазвичай, запитуваний текстовий файл містить HTML-розмітку, яка присвоюється у якості вмісту одному з елементів сторінки.

Для ілюстрації цих дій створимо сторінку (*index.htm*). Функція *start()* призначена для виконання ajax-запиту, а *ant()* – для оброблення відповіді.

```
<!DOCTYPE HTML>
<html>
   <head>
      <title></title>
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<style>
#d1 {width:300px;height:200px;background-color:#dddddd;}
</style>
      <script type="text/javascript">
var xhttp;
function start(){
/*…*/
                    }
function ant(){
/*…*/
                    }
</script>
   </head>
   <body">
      <div id="d1">
          Тут повинна бути відповідь сервера
      </div><br>
    <button id="bt"  onClick="start();">отримати дані </button>
    </body>
</html>
```

Функція *start()* забезпечує формування та відправку запиту серверу.

```
function start(){
 if (window.XMLHttpRequest) {xhttp=new XMLHttpRequest();}
 else                  {xhttp=new ActiveXObject("Microsoft.XMLHTTP"); }
 xhttp.onreadystatechange=ant;
 xhttp.open("GET","text.txt",true);
 xhttp.send();
           }
```

Функція *ant()* забезпечує перевірку відповідей сервера і впровадження отриманого коду у текст сторінки.

```
function ant(){
   if (xhttp.readyState==4 && xhttp.status==200){
     document.getElementById("d1").innerHTML=xhttp.responseText;
                                                    }
            }
```

У цьому прикладі розмічений текст із файла *text.txt* присвоюється у якості вмісту елемента з *id = "d1"* (елемент div на цій сторінці).

Незважаючи на відносну простоту організації асинхронної роботи, у бібліотеку jQuery включено значну кількість функцій ще більше полегшують цю роботу. Серед них найчастіше використовуються *jQuery.ajax()* – для відправки запиту до сервера і *load()* – для завантаження HTML з файла на сервері.

У разі використання функції *jQuery.ajax()* вміст тегів <script> на сторінці попереднього прикладу може виглядати так.

```
<script src=jquery.js></script>
<script type="text/javascript">
    function start(){
          $.ajax({url: "text.txt",
                  success: function(data) { $('#d1').html(data); }
                });
                 }
</script>
```

Під час використання *load()* код буде ще коротше.

```
<script src=jquery.js></script>
<script type="text/javascript">
       function start(){ $("#d1").load('text.txt'); }
</script>
```

Під час використання асинхронних запитів до сервера, особливо якщо їх на сторінці багато, необхідно брати до уваги кілька загальних міркувань.

Користувачеві важко стежити за тим, де він знаходиться. Тому доцільно виводити повідомлення або якісь маркери, що інформують його і допомагають повернутися до стану, який був до виконання запитів.

Численні асинхронні блоки можуть уповільнити роботу зі сторінкою, тому їх число повинне бути обґрунтованим.

Несподівані зміни сторінки можуть заважати комфортній роботі користувача, доцільно попереджати про майбутні зміни, під час процесу і про його завершення.

Можливе виникнення проблем і з індексацією контенту, якій довантажувалося асинхронно.

3. Засоби роботи з базою даних MySQL

Мова PHP орієнтована на роботу з СУБД MYSQL. Робота з БД організовується через SQL-сервер, причому SQL-сервер у будь-якому випадку розглядається як видалений, тобто для його використання створюється мережеве з'єднання. Завдяки цьому можна відкривати з одного скрипта або декілька призначених для користувача сесій, або працювати з різними SQL-серверами. Таким чином, навіть для автономної відладки на комп'ютері

окрім WEB-сервера має бути встановлений і запущений MySQL-сервер. Само створення бази даних та її таблиць може бути виконане як зі скрипта, так і з будь-якої програми-менеджера (наприклад, phpMyAdmin).

Якщо база даних уже створена, то у загальному випадку порядок виконання дій такий. Після встановлення з'єднання з сервером, вибирається база даних для роботи. Потім формуються запити на оброблення. Для виконання запиту створюється об'єкт, у якому зберігається результат виконання запиту або дані для запису.

Мова PHP містить безліч функцій для роботи з БД. Далі наведені прототипи функцій, що забезпечують приведену послідовність дій:

resource mysql_connect ([string server [, string username [, string password]]]) – встановлює з'єднання з сервером;

bool mysql_close ([resource link_identifier]) – розриває з'єднання з сервером;

bool mysql_select_db(string database_name[,resource link_identifier]) – вибір бази даних для роботи;

resource mysql_query(string query) – відправка запиту сервера (сам запит представляє рядок, складений за правилами SQL);

array mysql_fetch_array (resource result) – розміщення значень полів у асоціативному масиві.

Як приклад, розглянемо сайт, який забезпечує занесення у БД номерів телефонів різних організацій і проглядання всього списку. Створимо три сторінки: одну – для додавання записів *p_in.php*, другу для перегляду *p_out.php*, і початкову – *index.htm*.

Початкова сторінка (index.htm) містить два посилання:

```
<html>
<head>
<title>Сторінка записи у БД</title>
</head>
<body>
<a href="p_out.php"> проглядання даних із БД</a><br>
<a href="p_in.php"> додавання даних у БД</a>
</body>
</html>
```

Сторінка додавання (*p_in.php*) містить такий код:

```
<html>
<head>
<title>Сторінка записи у БД</title>
</head>
```

```
<body>
<p> Для додавання запису у БД заповніть форму:</p>
<form action='p_in.php' method=POST>
<p>Введіть назву <input type=text name='name'></p>
<p>введіть номер телефона <input type=text name='num'></p>
<input type=submit value='відправить'>
<input type=reset value='сброс'>
</form>
<?php
$hostname="127.0.0.1";
$username="root";
$password="1";
$dbName="test";
if ($_REQUEST["name"]) {
$my_name=$_REQUEST["name"];
$my_num=$_REQUEST["num"];
mysql_connect($hostname,$username,$password) OR DIE("Проблеми при
створенні з'єднання");
mysql_select_db($dbName) or die(mysql_error());
$query = "INSERT INTO tel VALUES('$my_name','$my_num')";
mysql_query($query) or die(mysql_error());
mysql_close();
echo "<p>Дані збережені</p>";        };
if (!isset($my_name) && $_REQUEST) echo "<p>Заповніть форму!</p>";
?>
<a href="p_out.php">Проглядання даних із БД</a>
</body>
</html>
```

Спочатку задаються атрибути для підключення до сервера (IP сервера, ім'я користувача і пароль). Зазначимо, що для роботи цього сайта база даних (test) і таблиця (tel) мають бути створені раніше. Зробити це можна, розробивши відповідний код PHP або скориставшись програмою-менеджером (наприклад, phpMyAdmin).

Потім проводиться підключення до сервера (*mysql_connect*). Функція *die()* – це синонім *exit()*, вона викликається, якщо *mysql_connect* поверне false, буде виведено повідомлення про проблему, і виконання коду припинено. Після успішного підключення вибирається БД з ім'ям test.

Запит на додавання запису має вигляд:

"INSERT INTO tel VALUES('$my_name','$my_num')", де

tel – ім'я таблиці, у яку здійснюються записи;

аргументи функції VALUES() – значення полів.

Після успішного виконання запиту з'єднання розривається.

Сторінка перегляду (*p_out.php*) містить такий код:

```
<html>
<head>
<title> Сторінка доступу до БД </title>
</head>
<body>
<!-- початок таблиці -->
<table width="50%" border="1" cellspacing="0" cellpadding="0" align="CENTER">
<tr>
<th width="75%">Назва фірми</th>
<th width="25%">Телефон</th>
</tr>
<?php
$link=@mysql_connect("127.0.0.1","root","1");
if (!$link) {echo "Проблема у підключені до сервера БД";exit();};
mysql_select_db("test", $link);
$R=@mysql_query("SELECT * FROM tel",$link");
if (!$R) {echo "Проблема доступу до записів".msql_error();exit();};
while ($T=mysql_fetch_array($R))
   { echo "<tr><td>".$T[name]."<td>".$T[num];};
mysql_close();
?>
</table>
<a href="p_in.php">Додавання даних у БД</a>
</body>
</html>
```

Цей код відрізняється від попереднього у частині оброблення помилок (ще один варіант із безлічі можливих) і запитом на вибір даних (на відміну від додавання, INSERT) *"SELECT * FROM tel",$link"*, у якому вказано ім'я таблиці і посилання на вибрану БД. Результат запиту (прочитані записи) збережені у об'єкті $R. Витягання записів виконується функцією *mysql_fetch_array()*, яка викликається у циклі. Значення полів (елементи масиву $T[]) поміщаються у елементи таблиці.

4. Використання фреймворку Yii

Yii – це фреймворк для створення WEB-додатків загального призначення, таких, як портали, форуми, системи управління контентом (CMS), системи електронної комерції і ін. у основу програми покладено архітектуру MVC (*model-view-controller*).

Для установки фреймворка необхідно розпакувати вихідні матеріали у папку сервера і запустити вхідний скрипт. У мережі міститься безліч детальних інструкцій щодо виконання цього етапу, наприклад [2].

Використовувана база даних вказується у конфігураційних файлах.

Створення власного додатка може бути виконано у такій послідовності:

створити таблицю у базі даних для зберігання даних;

створити модель даних (об'єкт для роботи з базою даних);

створити контролер (об'єкт для оброблення даних, отриманих від користувача, і загальна логіка програми);

створити уявлення (заготовка сторінки для введення і відображення даних).

Таблиця може бути створена з програми або з використанням відповідного менеджера СУБД. Нехай створена таблиця *users* (з полями: *id*, *login*, *passwd*), що містить дані про користувачів, і серед них користувач з логіном "admin".

Модель (Model). Модель – це клас для роботи програми з базою даних. Для кожної таблиці, з якою доведеться працювати, створюється своя модель (у Yii це клас, успадкований від CModel або похідного від нього).

Найпростіша модель виглядає таким чином:

```
<?php
class Users extends CActiveRecord
{
   public static function model($className=__CLASS__)
   {
      return parent::model($className);
   }
   public function tableName()
   {
      return 'user';
   }}
```

Ці два методи є обов'язковими для будь-якої моделі. Решта методів додаються відповідно до необхідних дій. Слід урахувати, що завдяки спадкоємству від класу CActiveRecord, у складі моделі є безліч вже готових методів (*find, count, findAll* тощо), яких достатньо для типових дій з базою даних. Файл моделі зазвичай розташовується у папці *protected/models* і збігається з назвою класу.

Контролер (Controller). У Yii контролером називається клас, успадкований від CController або дочірнього від нього. У додатку може використовуватися як один контролер, так і декілька.

Усередині класу контролера повинні знаходитися спеціальні методи, які відповідають за окремі дії. Вони мають префікс *action* перед своїм ім'ям. Їх можна викликати всередині програми і з командного рядка браузера.

Відносно контролерів діє спеціальна угода про імена (обов'язковий суфікс *Controller* після імені), наприклад, adminController.

Приклад контролера з ім'ям *admin*, що містить два методи:

```
<?php
class adminController extends CController
{
   public function actionIndex()
   {
die ("Виконано метод actionIndex()");
   }
   public function actionAuthor()
   {
      die ("Виконано метод actionAuthor()");
   }
}
```

Під час звернення до програми за адресою *localhost/admin/index/* буде отримано на екрані відповідний запис. Зазвичай усі контролери у Yii розташовуються за адресою *protected/controllers/*.

З контролера можна звернутися до моделі без створення екземпляра класу:

назва_моделі :: model() -> назва_методу();

або попередньо створивши екземпляр моделі:

$ Model = new Назва_моделі ();

$ Model-> назва_метода ();

Наприклад, за наявності моделі запит для отримання користувача з логіном *admin* може бути виконаний так:

```
$user_info = User::model()->findByAttributes(array('login' => 'admin'));
```

Для пошуку запису використаний вбудований метод *findByAttributes*, якщо такий запис у таблиці users існує, то у змінній $user_info буде збережений результат (об'єкт). Після цього можна отримати доступ до полів:

```
echo $user_info->login;
```

Помістивши цей код у один із методів контролера (наприклад, actionIndex), отримуємо можливість отримання даних про користувача admin.

Уявлення (View). Головним призначенням уявлень є висновок даних (наприклад, оброблених у контролері) користувачеві у вигляді HTML-коду. Відповідно до концепції MVC у уявленні повинен знаходитися мінімум коду логіки додатка (для цього спеціально існує контролер).

Приклад уявлення, що створює меню на сторінці сайта:

```
<h1>меню</h1>
<a href='#'>пункт 1</a>
```

```
<a href='#'>пункт 2</a>
<?php if ($user['status'] == "admin"): ?>
   <a href='#'>меню адміна</a>
<?php endif; ?>
```

Програмний код використаний тільки для перевірки статусу користувача і додавання ще одного пункту, якщо це адміністратор.

Для звернення до уявлення з контролера використовується спеціальний метод *render(назва_уявлення)*. За замовчуванням буде виведено на екран відображення з переданим назвою з папки views/назва_контроллера.

Уявлення для виведення інформації про користувача може виглядати так:

```
<pre>
   Id : <?php echo $info->id; ?>
   Логін: <?php echo $info->login; ?>
   Пароль: <?php echo $info->passwd; ?>
</pre>
```

Воно повинне розміщуватися у папки з ім'ям контролера (admin) всередині views. Зазвичай для кожного контролера у папці views створюється своя папка, зі своїми уявленнями.

Якщо у контролері admin написати $this->render('my_view_file'), то файл уявлення в цьому випадку буде довантажуватися з адреси protected/views/admin/my_view_file/.

В остаточному вигляді цій найпростішій додаток повинен містити три файли.

Модель (*User.php*) за адресою protected/models.

```
<?php
class User extends CActiveRecord
{
   public static function model($className=__CLASS__)
   {
      return parent::model($className);
   }
   public function tableName()
   {
      return 'users';
   }
}
```

Контролер (*Admin Controller.php*) у папці protected/controllers.

```
<?php
class AdminController extends CController
{
```

```
    public function actionIndex()
    {
        if ($user_info = User::model()->findByAttributes(array('login' => 'фвьшт'))) {
            $this->render('index', array( 'info' =>$user_info,));
        }
        else {
            die('нет такого пользователя');
        }
    }
}
```

Подання (*index.php*) у папці views/admin.

```
<pre>
    Id : <?php echo $info->id; ?>
    Логін: <?php echo $info->login; ?>
    Пароль: <?php echo $info->passwd; ?>
</pre>
```

Під час набору у адресному рядку браузера *localhost/admin/index* у вікні повинні бути виведені значення відповідних полів.

Усі ці об'єкти (моделі, контролери) можуть створюватися як шляхом написання коду, так і автоматично за допомогою генератора (спеціальна програма Gii у складі YII).

Література: [2; 3].

Самостійна робота 5. Використання графічних бібліотек

Мета роботи: отримання знань та навичок у створенні динамічної графіки на сервері.

У результаті виконання самостійної роботи у студента формуються такі **компетентності:** здатність самостійно створювати та використовувати зображення на сервері.

Результатом виконання самостійної роботи повинен стати додаток для динамічного створення і відображення зображення на основі даних, які задає користувач. Тип зображення студент обирає самостійно та узгоджує з викладачем.

Завдання для самостійної роботи

Виконувати дану самостійну роботу доцільно під час вивчення теми "Динамічна графіка на Web-сторінках".

У ході виконання роботи необхідно створити сторінку, на якій користувач буде вводити дані і переглядати створенні зображення, і PHP-додаток для оброблення введених даних, побудови зображення та відправки його користувачеві.

Контрольні запитання для самодіагностики

1. Чим відрізняється процес створення зображення на сервері від аналогічних дій на клієнті?

2. Сформулюйте послідовність дій зі створення зображення на сервері.

3. Назвіть імена основних функцій для кожного етапу і їх призначення.

4. Поясніть роль заголовків, що відправляються сервером, і їх особливості під час відправки зображень.

5. Назвіть способи виведення динамічне створених зображень у браузері.

Методичні рекомендації до теми

Створення зображень у PHP вимагає наявності спеціальної графічної бібліотеки (ImageMagick, GD, GraPHPite, pChart тощо). Найчастіше використовується GD (http://www.boutell.com/gd). Підтримка цієї бібліотеки включається під час компіляції PHP.

Генерація зображення складається з трьох основних етапів:

створення зображення;

малювання зображення;

відображення отриманої картинки у браузері.

Для створення зображення використовується функція *ImageCreate()*. У якості аргументів функція приймає ширину і висоту зображення у пікселях і повертає ідентифікатор, який у подальшому використовується для виклику функція при роботи з зображенням. Цим по суті створюється полотно для малювання зображення. Саме зображення створюється у пам'яті процесу з елементів (відрізків, дуг, текстів, зображень тощо).

Перед початком створення елементів зображення необхідно зареєструвати кольори, які будуть використовуватися (за допомогою функції *ImageColorAllocate()*). Цієї функції передаються ідентифікатор зображення і три числа, що задають колір (RGB). Функція повертає ідентифікатор кольору, який використовується у подальших операціях.

```
// реєстрація кольору
$colorHandle = imageColorAllocate($image,192,192,192);
// використання для малювання
```

imageFilledRectangle($image,0,0,$diagramWidth-1,$diagramHeight-1, $colorBackgr);

Список функцій для відтворення зображення можна знайти у Інтернеті.

Створене зображення може бути перетворено у один із графічних форматів (PNG, JPEG) і збережено у файлі або відправлено браузеру з використанням функцій *ImagePNG()* або *ImageJPEG()*.

bool imagejpeg(resource $image [,string $filename [,int $quality]])

bool imagepng (resource $image [,string $filename [,int $quality [,int $filters]]])

Функції мають кілька параметрів ($image – посилання на полотно, $filename – ім'я файла, $quality – якість, пов'язана зі стисненням), обов'язкове тільки посилання на полотно. Якщо ім'я файла не задано, то зображення вивантажується у вихідний потік, якщо задано, то пишеться у файл.

Перед використанням функцій, які відправляють зображення клієнту необхідно надіслати відповідний заголовок "Content-type: image/png" або "Content-type: image/jpeg". Заголовки належать до всього документа, під час виведення зображення не слід виводити текст (HTML-розмітку).

Для відправки заголовка слугує функція *header("Content-type: ...")*, вона повинна викликатися першої, і тільки після неї виконується виведення даних.

Після виконання дій зі створення зображення і його відправки доцільно звільнити ресурс пам'яті, зайнятий зображенням. Виконуються це функцією *imageDestroy (resource $image)*.

З урахуванням особливостей формування заголовків сервером для візуалізації створених зображень у браузері можна використовувати кілька способів (відкрити у окремому вікні, відкрити на сторінці у тегі img).

Для відображення у окремому вікні адресу скрипта необхідно набрати у адресному рядку браузера (наприклад, http://localhost/script.php) або помістити у якості значення атрибута href у посиланні.

Для вбудовування зображення у сторінку можна використовувати тег img або iframe:

<img src="http:// localhost/script.php">

Далі наводиться текст коду програми, що виконує побудову на сервері діаграми за даними, введеними користувачем у формі. Для звернення до сервера використаний клієнтський скрипт, що формує значення атрибута src з урахуванням введених користувачем даних.

Код HTML-сторінки /

<!DOCTYPE HTML>
<html>

```
<head>
<meta charset="utf-8">
<title>графіка на PHP</title>
<style>
img {width:100;}
</style>
<script>
function f1()
{
s=document.getElementById("d").value;
st="php_d.php?diag="+s;
document.getElementById("im").setAttribute("src",st);
}
</script>
</head>
<body>
<form id="fm">
<p> Введіть значення для побудови діаграми </p>
<textarea cols=30 rows=5 name="diag" id="d">
</textarea>
<input type=button value="Построїть" onClick="f1();">
</form>
<img src="" id="im">
</body>
</html>
```

Код PHP-скрипта (*php_d.php*).

```
<?php
$v_dan = $_GET["diag"];
 $dan=explode(",",  $v_dan);
   header ("Content-type: image/png");
    $wid = 400;
    $heig = 300;
    $im = imagecreate($wid, $heig) ;
    $blanc = ImageColorAllocate ($im, 255, 255, 255);
    $noir = ImageColorAllocate ($im, 0, 0, 0);
    $bleu = ImageColorAllocate ($im, 0, 0, 255);
    ImageLine ($im, 10, $heig-10, $wid-10, $heig-10, $noir);
    for ($i=1; $i<=count ( $dan ); $i++) {
                ImageString ($im, 0, $i*30, $heig-10, $i, $noir);
                                                    }
   ImageLine ($im, 10, 10, 10, $heig-10, $noir);
   $danMax = 100;
     for ($i=1; $i<=count ( $dan ); $i++) {
```

```
    $hauteurImageRectangle = round(($dan[$i-1]*$heig)/$danMax);
    ImageFilledRectangle($im, $i*30-7, $heig-$hauteurImageRectangle, $i*30+7, $heig-10, $bleu);
    ImageString ($im, 0, $i*30-7, $heig-$hauteurImageRectangle-10, $dan[$i-1], $noir);
  }
imagepng($im);
imagedestroy($im);
?>
```

Література: [5; 8].

Рекомендована література

1. Молчанов В. П. Засоби систем обробки даних : навчальний посібник / В. П. Молчанов. – Харків : Вид. ХНЕУ, 2013. – 100 с.
2. Молчанов В. П. Технології WEB-дизайну : конспект лекцій / В. П. Молчанов. – Харків : Вид. ХНЕУ, 2011. – 212 с.
3. Томсон Л. Разработка Web-приложений на PHP и MySQL / Л. Томсон, Л. Веллингтон ; пер. с англ. – 2-е изд., испр. – Санкт-Петербург : ООО "ДиаСофтЮп", 2003. – 672 с.
4. Холзнер С. XML. Энциклопедия / С. Холзнер. – 2-е изд. – Санкт-Петербург : Питер, 2004. – 1101 с.
5. Создание изображений средствами PHP [Электронный ресурс]. – Режим доступа : http://phpclub.ru/detail/article/dynamic_image.
6. Установка информационных служб Интернета (IIS) 7.5 [Электронный ресурс]. – Режим доступа : http://windows.microsoft.com/ru-ru/windows7/install-internet-information-services-iis-7-5.
7. Школы консорциума W3C [Электронный ресурс]. – Режим доступа : http://www.xml.nsu.ru/extra/xslt_2.xml.
8. GD и функции для работи с изображениями Функции [Электронный ресурс]. – Режим доступа : http://php.net/manual/ru/ref.image.php.
9. Википедия WAMP [Электронный ресурс]. – Режим доступа : https://ru.wikipedia.org/wiki/WAMP.

НАВЧАЛЬНЕ ВИДАННЯ

Методичні рекомендації до самостійної роботи з навчальної дисципліни "ТЕХНОЛОГІЇ РОЗРОБКИ WEB-РЕСУРСІВ"

для студентів напряму підготовки 6.051501 "Видавничо-поліграфічна справа" всіх форм навчання

Укладач **Молчанов** Віктор Петрович

Відповідальний за видання *О. І. Пушкар*

Редактор *В. О. Бутенко*

Коректор *В. О. Бутенко*

План 2016 р. Поз. № 100.
Підп. до друку 07.11.2016 р. Формат 60 х 90 1/16. Папір офсетний. Друк цифровий.
Ум. друк. арк. 4,00. Обл.-вид. арк. 5,00. Тираж 40 пр. Зам. № 220.

Видавець і виготовлювач – ХНЕУ ім. С. Кузнеця, 61166, м. Харків, просп. Науки, 9-А

Свідоцтво про внесення суб'єкта видавничої справи до Державного реєстру
ДК № 4853 від 20.02.2015 р.