МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ПРИКАРПАТСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВАСИЛЯ СТЕФАНИКА

Кафедра інформаційних технологій

КОНСПЕКТ ЛЕКЦІЙ

з дисципліни

«Програмування та підтримка веб-застосувань»

для студентів напряму підготовки
«Інформатика»

ІВАНО-ФРАНКІВСЬК – 2015 РІК

УДК 004.43
ББК 32.973.2-018
Л17

Рекомендовано до друку вченою радою факультету математики та інформатики Прикарпатського національного університету імені Василя Стефаника, (протокол №3 від 08 грудня 2015 р.)

Рецензенти:

Філевич П.В. – доктор фіз.-мат.наук, професор, завідувач кафедри інформаційних технологій Прикарпатського національного університету імені Василя Стефаника;

Соломко А.В. – кандидат фізико-математичних наук, доцент кафедри математичного і функціонального аналізу факультету математики та інформатики Прикарпатського національного університету імені Василя Стефаника

Лазарович І. М.

Л17 Конспект лекцій з дисципліни «Програмування та підтримка веб-застосувань» для студентів напряму підготовки «Інформатика» / І. М. Лазарович – Івано-Франківськ: Видавництво Прикарпатського національного університету імені Василя Стефаника, 2015. – 153 с.

Наведено основні характеристики та тенденції розвитку сучасних технологій веб-програмування, архітектури типових веб-додатків. В якості програмування на стороні клієнта розглядається мова JavaScript, що використовується у складі HTML-сторінок. Подано синтаксис мови, методику створення сценаріїв на основі DOM. Для програмування на стороні сервера застосовано мову PHP, яка дозволяє формувати HTML-код сторінки безпосередньо на сервері, забезпечувати інтерфейс до бази даних, працювати з формами і файлами.

З метою підвищення ефективності веб-розробки в роботі розглянуто питання оптимізації, розміщення та «розкрутки» сайту, що дозволить читачам навчитись створювати веб-весурси та забезпечувати їх підтримку.

Конспект лекцій розроблено на основі вимог освітньо-кваліфікаційної характеристики та освітньо-професійної програми підготовки фахівців з напряму «Інформатика». Наведений матеріал може бути використаний студентами інших напрямів підготовки, які вивчають веб-програмування і мають базові навики веб-дизайну та розробки сторінок на основі HTML та CSS.

УДК 004.43
ББК 32.973.2-018

ЗМІСТ

ПЕРЕДМОВА

Всесвітня мережа Інтернет - це середовище спілкування і інформаційного обміну між мільйонами людей, що живуть в різних країнах, на різних півкулях. Вони розміщують в Інтернеті, на Web-серверах або пересилають по електронній пошті різну інформацію - текст, малюнки, відеозображення, звуковий супровід, мультимедійні чи будь-які інші дані і поступово ускладнюють свої Web-сторінки, роблячи їх більше інтерактивними і динамічними, з встановлення установки зворотного зв'язку.

Такий розвиток Інтернет став можливим завдяки технологіям веб-програмування, які передбачають процес створення веб-сайту або веб-додатку. Основними етапами процесу є веб-дизайн, верстання сторінок, програмування для веб на стороні клієнта і сервера.

В межах даної роботи розглянуто архітектури типових веб-застосувань, програмування для веб на стороні клієнта з використанням мови JavaScript, що використовується у складі HTML-сторінок для збільшення їх функціональності та можливостей взаємодії з користувачами. Описано програмування на стороні сервера з використанням мови PHP. Конструкції PHP, вставлені в HTML-текст, виконуються сервером при кожному відвідуванні Web-сторінки. Результат їх обробки разом із звичайним HTML-текстом передається браузеру.

Прикінцевими етапами веб-розробки є оптимізація структури сайту, його розміщення в мережі, просування та пошукова оптимізація. В книзі розглянуто ці етапи, що дозволить читачу засвоїти методику створення сайтів.

Конспект призначений для студентів, які вивчають веб-програмування, для розуміння та засвоєння поданого матеріалу потрібно володіти навиками розробки веб-сторінок на основі HTML та CSS.

РОЗДІЛ 1. АРХІТЕКТУРА ТА ПРОГРАМУВАННЯ ВЕБ-ЗАСТОСУВАНЬ

Тема 1.1. Загальна характеристика та тенденції розвитку веб-застосувань

1.1.1 Історія виникнення та розвитку Веб

Вік Інтернету складає приблизно 20 років, і за цей час він пережив істотні зміни. В нашому стрімкому світі ніщо не стоїть на місці, і Інтернет - не виключення. Тому, тепер вже застосовують позначення як Веб 1.0, Веб 2.0, Веб 3.0, що характеризують розвиток Всесвітньої мережі. Чітких визначень такої класифікації поки що немає, проте є принципи, що дозволяють зрозуміти відмінність одного стану Інтернету від іншого.

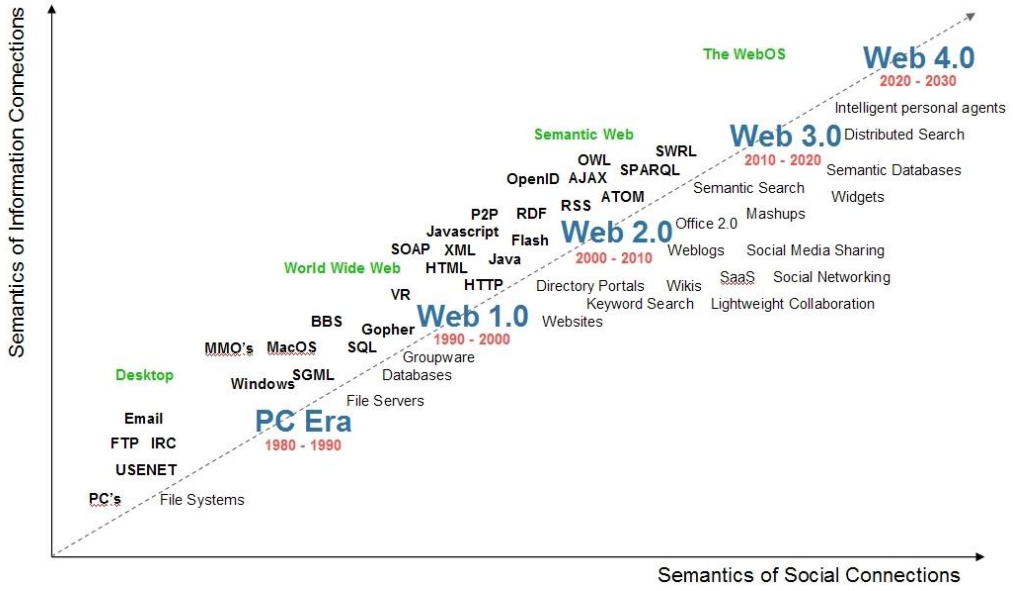


Рисунок 1.1 Розвиток комп'ютерних та Інтернет-технологій

На рисунок 1.1 наочно проілюстровано розвиток Веб та еволюцію від PC-ери до можливого виникнення в майбутньому повноцінної WEBOS.

Терміни Веб x.0 вживаються з єдиною метою - виокремити різні епохи Веб. На разі людство перебуває в епосі Веб 2.0, але непомітно відбувається перехід до епохи Веб 3.0.

Веб 1.0

Веб 1.0 є поняттям, яке відноситься до статусу служби Веб та до стилю дизайну сайтів, що використовувалися перед появою Веб 2.0. Веб 1.0 – це загальний термін, який було створено, щоб описати Інтернет перед кризою дот-комів у 2001, яка вважається поворотним моментом в історії Веб.

Криза дот-комів

Назва дот-ком походить від озвучення домену «.com» (дот (dot) - крапка). Це економічна криза, яка вразила в 2000 - 2001 великі американські Інтернет компанії і надовго відвернула інвесторів від Інтернет технологій.

Кінець дев'яностих був насичений прогнозами про чудове майбутнє Інтернет бізнесу. До 2000 року стратегія Інтернет бізнесу полягала в створенні великих Інтернет проектів, що часто бути скеровані на онлайн торгівлю. Поширеною була думка, що в найближчий час люди будуть все купувати в онлайн магазинах. За приклад був проект Yahoo!, що з невеликого сайту перетворився в надвелику торгівельну структуру, а його акції дуже котирувалися.

В результаті виникло багато дрібних компаній (StartUp), що прагнули повторити досвід успішних Інтернет магазинів. Інвестиції залучалися у венчурних фондів, які, переконавшись в успіху Yahoo! охоче надавали кошти. Стартапи на отримані гроші намагалися себе розкрутити. І робили це за допомогою реклами на ... Yahoo! Акції Yahoo! від цього росли ще вище, а інвестори дивлячись на це зростання вкладали в Інтернет компанії ще більше грошей.

І ось в 2000-20001 році прийшло усвідомлення, що вся ця схема є «мильним міхурцем», бо люди і далі купували товари в звичайних магазинах. Після чого акції Інтернет компаній обвалилися і збанкротувало багато як великих, так і дрібних компаній. Те ж саме відбулося і з «винуватцем» торжества - Yahoo!, його акції впали вдесятеро, але йому вдалося втриматися на плаву. Він і досі є самим популярним сайтом в США.

Криза дот-комів привела до того, що в інвесторів надовго пропало бажання інвестувати в Інтернет. І лише успіхи Google, Вікипедії, MySpace і YouTube почали виправляти ситуацію.

Елементна база сайтів, що є типовою для Веб 1.0

Термін «типова елементна база» вживається тут в узагальненому сенсі — абсолютно не обов'язково, що для будь-якої сторінки Веб 1.0 були властиві всі елементи списку. Але, як правило, загальна тенденція і більша частина елементів була присутньою.

Краще за все сформулювати список елементної бази, яка була типовою для Веб 1.0, можна на підставі того, з чим боролися послідовники Веб 2.0, а саме:

- Статичні сторінки замість динамічного контенту, який генерується користувачами.
- Проста гіпертекстова розмітка. Більша частина контенту, зазвичай, була простим текстом, де часто спостерігалося нехтування правилами HTML.

- Використання фреймів.
- Використання специфічних тегів HTML, які не однаково підтримувалися в різних браузерах.
- Елементи інтерактивності були присутні, зазвичай, лише в гостьових книгах, форумах або чатах.
- Рідкісне використання стилів CSS при оформленні сторінок сайту.
- Вказівка конкретної роздільної здатності монітору, при якому дизайн сайту відображається коректно (не поширюється за межі сторінки, не зсуваються елементи сторінок).
- Використання інформерів (погода, курс долара тощо) замість агрегації інформації засобами CMS.
- Вказівка на браузер, в якому сайт відображається найкраще.

Отже, під Веб 1.0 розуміють статичні сайти, що наповнені корисною, довідковою інформацією. Наповнення здійснює обмежене коло осіб, в основному їх власники і автори.

Веб 2.0

Веб 2.0 — новий етап розвитку служби WWW.

Першим, хто вжив поняття Веб 2.0, стало видавництво O'Reilly Media, що спеціалізується на інформаційних технологіях. Згідно опублікованої у вересні 2005 року статті Тіма О'Райлі, засновника компанії O'Reilly Media «Що таке Веб 2.0?», концепція Веб 2.0 з'явилася в результаті «мозкового штурму» між компаніями O'Reilly Media і MediaLive International. Зокрема, обговорювалося питання про те, чи слід вважати крах дот-комів крахом Інтернету.

Учасники цього «мозкового штурму» дійшли висновку, що колапс дот-комів виявився важливим етапом в розвитку Інтернету, внаслідок якого з’явився Веб 2.0 - Інтернет другого покоління.

Для точної характеристики Веб 2.0 використовується кілька основоположних принципів, більшість з яких було зазначено в тій статті Тіма О’Райлі.

Перший принцип. «Веб як платформа» передбачає розробку і впровадження застосувань, використання яких є можливим прямо з веб браузера. Це так звані Rich Internet Applications — насичені Інтернет-застосування. Вони мають функції традиційних програм для комп’ютера, але їх можна запустити прямо з Інтернету.

Другий принцип. Використання при створенні веб-застосувань нових Веб-технологій. Веб 2.0 подарував Інтернету AJAX, синдикацію контента (RSS), фолксономію і багато іншого.

Третій принцип. Співпраця розробників та користувачів у відкритій інформаційній інфраструктурі.

Четвертий принцип. Соціальні мережі та блогосфера. Саме вони вивели спілкування та інформаційну взаємодію між користувачами Інтернету на абсолютно новий рівень.

Отже, Веб 2.0 — це Інтернет для користувачів і від користувачів, це новий рівень взаємодії користувачів з Інтернет ресурсами. Користувач Веб 2.0 виступає не просто споживачем контенту, який дбайливо надано йому авторами сайту, а також бере посильну участь в створенні цього контенту та впливає на подальший розвиток сервісів.

Особливості Веб 2.0

Рисунок 1.2 Особливості та технології Веб 2.0

Веб-сервіси. Доступ до веб-сервісів здійснюється за допомогою звичайного браузера, користувачам не потрібно встановлювати жодних додаткових програм на своїх комп'ютерах, не потрібно піклуватися про постійні оновлення. Виконання будь яких обчислень веб-сервісами не вимагає від комп'ютера користувача жодних витрат ресурсів — всі операції виконує сервер.

Веб Mash-up (Змішування) — можливість створити новий сервіс, який повністю або частково використовує як джерела інформації інші сервіси, надаючи користувачеві нову функціональність для роботи. Цей сервіс може бути новим джерелом інформації для інших mash-up сервісів. Таким чином, утворюється мережа залежних один від одного сервісів, що інтегровані між собою.

Наприклад, поєднання сайту пошуку нерухомості з інтегрованими картами Google Maps надають новий, зручний сервіс, за допомогою якого

кожен користувач може відразу побачити всі пропозиції з продажу будинків на карті.

Ajax — методика застосування технологій JavaScript і XML, яка дозволяє завантажувати потрібні дані у відповідь на дії користувача, не перезавантажуючи веб-сторінку цілком. Завдяки використанню цього підходу можна значно пришвидшити роботу користувача з сайтом.

Можливості AJAX найкращим чином втілено у сервіси Google.

- **Google Suggest.** Під час набору запиту в рядку пошуку Google пропонуватиме вам варіанти найбільш часто використовуваних запитів, схожих на ваш, з інформацією про кількість знайдених результатів. Ця схема використовується в браузерах — коли набирається адреса в адресному рядку браузера, то з'являється випадний список пропонованих варіантів. До появи технології AJAX така схема на веб-сторінках практично не реалізовувалася, а зараз використовується повсюдно, підвищуючи юзабіліті сайтів.
- **Google Maps**. При спробі перетягування карти в різні боки, карта пересувається, але сторінка при цьому не перевантажується
- **Gmail**. Веб-інтерфейс цілком побудовано на AJAX, що дає йому право вважатися гідною альтернативою для поштових клієнтів The Bath, OutLook тощо.
- **Google Calendar.** Сервіс для планування подій, зустрічей, справ з прив'язкою до календаря.

Ці сервіси мають значні переваги перед оффлайновыми аналогами. А саме, їх мобільність. Ви не можете дістати доступ до своєї пошти чи органайзера, які зберігаються на вашому домашньому чи офісному комп'ютері, але ви завжди можете доступитися до документів, якщо вони зберігаються в Інтернеті.

RSS — технологія, що заснована на XML, яка дозволяє користувачам за допомогою спеціальної програми-агрегатора переглядати як єдиний потік новини з багатьох сайтів — стрічку новин. Завдяки цієї технології користувачам більше не потрібно проглядати десятки сайтів для того, щоб дізнатися останні новини. Підписавшись на RSS, кожен може читати нові публікації кількох сайтів в одному місці.

Теги дозволяють зручніше ідентифікувати і тематично сортувати контент (статті, малюнки, мультимедійні файли). Наприклад, об'єкт «ваза з жовтими квітами» складно віднести до однієї категорії. Чи помістити цей об'єкт в категорію «посуд»? Або все ж таки в категорію «квіти»? А може навіть в категорію «жовте»? З тегами «ваза», «жовтий» і «квіти» даний об'єкт

може бути швидко знайдений в будь-якій з цих категорій. У Веб 2.0 мітки зазвичай представляються у вигляді так званої «хмари тегів».

Wiki-сайти (яскравим прикладом є Wikipedia) дозволяють своїм користувачам власноруч редагувати, додавати або видаляти інформацію на сайті, створювати нові сторінки. Таким чином, користувачі активніше беруть участь в наповненні сайтів потрібною інформацією.

Соціалізація. Яскравим прикладом соціалізації в Веб 2.0 є ведення особистих мережних щоденників та блогів. За допомогою блогу кожен користувач може якось виділитися з натовпу, персоналізувати свою зону сайту — додати особисті аудіо і відео файли, зображення, публікувати статті або ділитися новинами. Крім того, соціалізації сприяє активне створення співтовариств, в яких кожен користувач може залишити своє повідомлення, поділитися проблемою, отримати багато різноманітних думок і висловити свої міркування з того чи іншого питання.

Однією з найважливіших складових соціальних мереж є технологія **FOAF** (*Friend Of A Friend*). Вона надає користувачеві можливість підписатися на новини чи матеріали тих користувачів, які знаходяться в «списку друзів». Цим самим заохочується спілкування користувачів Мережі.

Сервіси обміну. Ці ресурси наповнюються користувачами, їм надаються місця для різних файлів – музики, фільмів, документації тощо.

Сервіс сумісного користування документами. Подібні сервіси надають користувачам можливість одночасного і сумісного використання документів – можна створювати, змінювати, видаляти інформацію, що є доступною для загального користування. При цьому зникає необхідність в установці програмного забезпечення на локальних комп'ютерах. В даній області визнаним лідером є сервіс Writely і Google Документи.

Дизайн. Поняття Веб 2.0 також відбилося і в дизайні. Переважними стали округлість, імітація опуклих поверхонь, імітація віддзеркалень на зразок глянцевого пластика сучасних пристроїв. В цілому, сприйняття зовнішнього вигляду є приємнішим, хоча графіка таких сайтів займає більший об'єм, ніж при використанні аскетичного дизайну. У Веб 2.0 з'явилася тенденція значно збільшувати розміри шрифтів за значущістю змісту, особливо для заголовків, щоб чітко виразити їх на тлі строкатого графічного оформлення. Для текстового наповнення надається більше простору.

Проте, явною стає одноманітність класичного дизайну Веб 2.0. Провідні дизайнери вважають його вже застарілим і не креативним. Особливо це відбивається в сучасній тенденції створення інформативних сайтів, де

головну роль грає простота, витонченість, графічність і юзабіліті. В дизайні не повинно бути обмежень, а Веб 2.0 їх дещо пригальмовує.

Вказані принципи є лише невеликою частиною філософії Веб 2.0. Навколо самого поняття Веб 2.0 ведуться запеклі спори. Хтось вважає його суцільною фікцією, лише додатковим маркетинговим ходом, а хтось — справжньою революцією. Проте, незаперечним залишається той факт, що Веб 2.0 є спробою зробити Інтернет зручнішим, корисним для користувача, надати йому більшої свободи для дій.

Деякі експерти називають Веб 2.0 мильною бульбашкою і пустушкою. Критики вважають, що немає нічого принципово нового у використанні технологій, що існували і раніше. З чисто технічної крапки зору ці критики мають рацію. Є думка, що Веб 2.0 – чергова спекуляція, якій уготована доля знаменитого «дот-ком буму».

Ще одним приводом для критики Веб 2.0 стало побоювання за збереження приватної інформації про користувача. Існують підозри, що інвестори Веб 2.0 сервісів зацікавлені лише у контролі над великим об'ємом інформації особистого характеру. Наприклад, інформації про уподобання користувача, можна використати для фокусування Інтернет реклами.

Недоліки Веб 2.0

При використанні технологій Веб 2.0 власник сайту стає орендарем сервісу і/або дискового простору у певної сторонньої компанії. Залежність, що виникає при цьому, формує ряд недоліків нових сервісів:

- Залежність сайтів від рішень сторонніх компаній, залежність якості роботи сервісу від якості роботи багатьох інших компаній.
- Слабка пристосованість нинішньої інфраструктури до виконання складних обчислювальних завдань в браузері.
- Вразливість конфіденційних даних, що зберігаються на сторонніх серверах, для зловмисників (відомо про випадки розкрадання особистих даних користувачів, масові зломи облікових записів блогів).

Сайт епохи Веб 2.0 на перший погляд є інтерактивним і доброзичливим, дозволяє себе легко налаштовувати. Проте, збір статистики про користувачів, їх переваги та інтереси, особисте життя, кар'єру, коло друзів можуть допомогти власнику сайту маніпулювати спільнотою. За самими песимістичними прогнозами численні сайти Веб 2.0 вкупі з іншими сучасними технологіями створюють прообраз тоталітарної системи «Великого брата».

Переваги Веб 2.0

Проте, Веб 2.0 має і безперечні плюси. В звичайних сервісах (в сервісах Веб 1.0) користувач, за своєю суттю, є пасивним споживачем послуг.

Концепція Веб 2.0 передбачає активну діяльність користувачів, що орієнтована на участь в створенні контенту ресурсу. В процесі розвитку сервісу враховується досвід і думка користувачів даного сервісу. Це робить ресурси Веб 2.0 більш інтерактивними і надає користувачам свободу самовираження.

Приклади сайтів Веб 2.0:

- Вікипедія — Вільна багатомовна енциклопедія.
- Google Maps — Google-карти.
- Flickr — онлайн-фотоальбом.
- del.icio.us — служба онлайнових закладок.
- Netvibes — Персональний десктоп.
- Digg.com — Новинний ресурс.
- Pligg — Веб 2.0 CMS.
- Quintura — Візуальна пошукова система з інтуїтивною картою підказок.
- Live Journal — Сервіс для ведення блогів.
- Youtube — Відеосервіс.
- MySpace — Сайт мережних співтовариств.
- Last.fm — Музичне співтовариство.
- Ucoz — Веб-сервіс для створення сайтів.

Таким чином, при Веб 2.0 Інтернет сприймають перш за все, як засіб комунікації, його об'єктами є медіа-сервіси, блоги, агрегатори, соціальні мережі, а суб'єктами - співучасники.

Веб 3.0

Веб 3.0 визначають як високоякісний контент і сервіси, які створюються талановитими професіоналами на технологічній платформі Веб 2.0. Головна ідея Веб 3.0 полягає в тому, що користувач, який до цього одноосібно був залучений в процес формування контенту, відтепер працює в колективі. Його партнерами, окрім інших користувачів, є експерти напрямів, причому статус користувача може бути змінений на експертний, так само, як і форма співпраці власника контенту і порталу.

Експерт повинен бути своєрідним модератором публікованого контенту. По суті, не виключається і можливість платної основи для співпраці, але набагато важливішим моментом є поява в порталах формату Веб 3.0 «колективного розуму» (*wisdom of the crowds*), замість пануючого сьогодні «групового божевілля» (*madness of the mobs*). Веб 3.0 припускає появу вузькоспеціалізованих ресурсів, де буде проведено агрегацію всіх необхідних для користувача сервісів, інструментів професійної соціальної складової і буде здійснено публікацію експертного контенту, що підлягає модерації.

Веб 3.0 передбачає появу нової професії – «менеджер знань», яка повинна стати сполучною ланкою між Веб 1.0 (контент) і Веб 2.0 (сервіси зв'язку). Менеджер знань - це експерт в конкретній області, що привносить в співтовариство якісно відібрану інформацію і позбавляє пересічного користувача від необхідності пошуку та оцінювання.

Ким є ці «посередники»? Це, наприклад, веб-блогер, який збирає цікаві посилання на новині і ділиться ними із співтовариством своїх читачів (в них теж можуть бути мережні щоденники, але вони не є настільки фаховими в пошуку новин). До речі, спочатку терміном «веб-блог» позначався саме збір цікавих посилань, а не просто «мережний щоденник».

Культура блогів з'явилася в Інтернеті ще в 1996 році, але згодом вона розчинилася в масовій культурі звичайних щоденників нудьгуючих офісних працівників (стандартні сервіси типу LiveJournal, liveinternet.ru). Їх величезна кількість часто не дозволяє знайти хороший блог серед моря спамерів і графоманів.

Можна навести більш життєвий аналог. Людина шукає собі квартиру. Метод Веб 1.0 означав би розміщення оголошення на сайтах з продажу квартир, або самостійний пошук по чужих оголошеннях. Спосіб є незручним: параметрів для пошуку багато, а через Інтернет можна перевірити лише кілька найпростіших (число кімнат, близькість до метро, ціна тощо), причому найголовніші параметри (стан квартири і порядність господарів) перевірити взагалі неможливо. З іншого боку, можна скористатися методом Веб 2.0, тобто опитати знайомих через співтовариства або соціальні мережі. Тут картина є зворотною - багато співчуваючих і перевірених людей, але ніхто з них, на жаль, не здає квартири.

Кращий метод, яким може скористатися покупець, можна цілком характеризувати як Веб 3.0. Він зв'язується з людиною, яку йому порадили, і передає свої побажання. Той, у свою чергу, передає замовлення до групи агентів по квартирах. В цих агентів є оперативні електронні бази вільних квартир, з одного боку, і бази заявок з іншого. Отже, робота агента складається з людських стратегій «менеджера знань», який і пов'язує покупця з потрібною квартирою.

Особливості Веб 3.0

Серед основних візуальних особливостей Веб 3.0 можна відзначити її тривимірність, тоді як серед головних «внутрішніх» характеристик - наявність штучного інтелекту і здібність до самонавчання.

У Веб 1.0 і 2.0 основним елементом представлення інформації є веб-сторінка, що візуально є двомірним масивом інформації, користувач фізично

має можливість пересування лише в двох вимірах: третій вимір в «класичній» веб-сторінці відсутній.

На відміну від двовимірних сторінок Веб 1.0 і 2.0, у Веб 3.0 передбачається розвиток тривимірності, який виражається в можливості пересування в будь-яких трьох вимірах, що зробить «віртуальний світ» Інтернету таким же тривимірним як і реальний світ. З розвитком тривимірності сайтів можна очікувати їх об'єднання в єдиний тривимірний світ, що зовні нагадуватиме реальність.

Прикладом реалізації подібних моделей є тривимірні ігри, кожна з яких по суті є своїм власним світом з своїми особливостями і законами. На даний момент одним з найцікавіших варіантів реалізації подібної моделі є програма Microsoft Virtual Earth 3D в якій робиться спроба створення віртуальної тривимірної моделі реального світу.

Окрім зовнішніх, візуальних відмінностей, Веб 3.0 буде характеризуватися наявністю вираженого інтелекту і здібності до самонавчання. Зокрема, розвиток «семантичної Мережі» і аналогічних технологій приведе до створення комп'ютерних систем, що здатні розуміти не лише інформацію у формі, яка є зрозумілою для комп'ютера, але і практично будь-яку інформацію, яка є зрозумілою для людини.

Якщо розглянути це питання на прикладі пошукових систем, то можна привести наступну аналогію. Перші пошукові системи не мали здатності розуміти суть пошукового запиту: користувач задавав «ключову» фразу, і пошукова система лише здійснювала пошук документів, в яких ця фраза міститься, не розуміючи, що саме вона означає.

Наступним кроком з'явилася кластеризація, тобто розподіл різних слів і фраз по групах (кластерах). Завдяки кластеризації пошукова система «розуміє», що дане слово відноситься до певної групи слів.

У Веб 3.0 відбудеться подальший розвиток в цьому напрямі, і пошукова система зможе розуміти запит вигляду: «У мене в червні відпустка і я шукаю, куди б мені поїхати відпочивати. Збираюся витратити приблизно $2000. До того ж в мене дитина 11 років.»

Ще однією особливістю Веб 3.0 може стати поступова відмова від текстової форми взаємодії за допомогою клавіатури і домінування голосового спілкування.

Таким чином, при Веб 3.0 Інтернет стає способом життя; об'єкти - віртуальними світами, 3D-браузерами; суб'єкти - гравцями і адміністраторами.

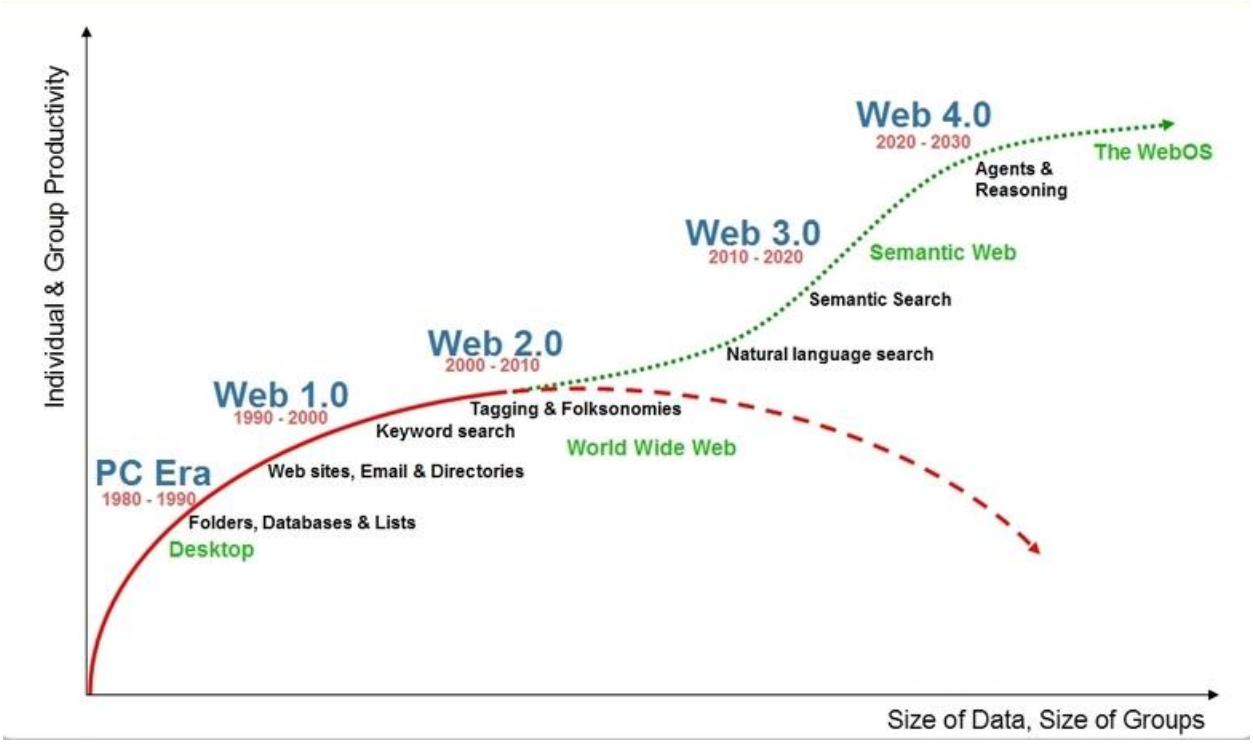


Рисунок 1.3. Майбутнє розвитку Інтернету

Основні засоби веб-технологій

HTML – мова розмітки тексту

Всесвітня павутина складається з веб-сторінок, які створено у форматі HTML (*HyperText Markup Language*, «мова розмітки гіпертексту»). HTML - це фундаментальна, базова технологія Інтернету.

HTML не є мовою програмування, це мова розмітки тексту, що використовує спеціальні оператори – **теги** (*tag*) чи інша назва **дескриптори** (*descriptor*) для розмітки текстового документа. Ці позначки вказують в якому вигляді буде виведено текстовий чи інший елемент у вікні браузера.

HTML дозволяє формувати на сторінці сайту текстові блоки, додавати до них зображення, організовувати таблиці, керувати відтворенням кольору, додавати до дизайну сайту звуковий супровід, організовувати гіперпосилання з переходом до інших розділів сервера або звертатися до інших ресурсів Інтернету і компонувати всі ці елементи між собою. Документи, що створено лише засобами HTML мають розширення *.htm* або *.html*.

Однією з основних функціональних особливостей мови HTML, завдяки якої вона і отримала свою назву, є гіперпосилання.

Гіперпосилання (*Hyperlink*) — це базовий функціональний елемент HTML-документу, який реалізовує зв'язок певного об'єкту веб-сторінки з іншим об’єктом. Для гіперпосилання може використовуватися як фрагмент тексту, так і графічний об'єкт, а сам гіперзв'язок можна встановлювати як між

об’єктами одного сайту, так і між об'єктами, що розміщені на різних сайтах Інтернету.

HTML є мовою, що лише інтерпретується, тому, для виконання коду, його не потрібно компілювати. Інтерпретатор мови втілено в браузер, і він «компілює» код безпосередньо під час відкривання документа. Якщо в коді сторінки виявлено помилку, інтерпретатор, зазвичай, не видає відповідного попередження, а просто ігнорує весь «помилковий» рядок, що може зіпсувати зовнішній вигляд завантаженої сторінки. Це є важливим для розробників, тому слід бути уважним під час складання HTML-коду і ретельно тестувати результати своєї роботи.

CSS – каскадна таблиця стилів

CSS (*Cascading Style Sheets*) — це технологія опису зовнішнього вигляду документа, що створено засобами HTML, XML і XHTML.

CSS використовується для завдання кольорів, шрифтів, розташування елементів сторінки тощо. До появи CSS оформлення веб-сторінок вказувалося безпосередньо в HTML-коді сторінки. Проте, з появою CSS стало можливим принципове розділення змісту і представлення документа. За рахунок такого нововведення стало можливим легке застосування єдиного стилю оформлення для кількох сторінок сайту, а також швидка зміна цього оформлення.

Переваги:

- Застосування кількох варіантів дизайну сторінки для різних пристроїв перегляду. Наприклад, для відображення на екрані монітора - дизайн буде розраховано на велику ширину. У разі друкування документу не буде роздруковане меню сайту, а у разі перегляду у мобільному комп’ютері чи телефоні, меню буде виведено після вмісту сторінки.
- Зменшення часу завантаження сторінок сайту за рахунок перенесення правил представлення даних до окремого CSS-файлу. В цьому випадку браузер завантажує лише структуру документа і дані, що містяться на сторінці. CSS-файл з правилами представлення цих даних завантажується браузером лише один раз і зберігається в кеші браузера.
- Простота подальшої зміни дизайну. Не потрібно виправляти кожну сторінку, достатньо лише змінити кілька правил у CSS-файлі.
- Додаткові можливості оформлення**.** Наприклад, за допомогою CSS-правил можна застосувати обтікання певного блоку текстом або зробити так, щоб меню фіксовано знаходилося в певному місці при перегортанні сторінки.

Недоліки:

- Різні браузери можуть в різний спосіб інтерпретувати правила CSS, і відповідно, по різному відображати одні і ті ж фрагменти сторінки.

DHTML – динамічна мова розмітки тексту

DHTML (*Dynamic HTML*) – це набір засобів, які реалізовують інтерактивність веб-сторінки без звертання до серверу. Тобто, певні дії відвідувача можуть спричинити зміну зовнішнього вигляду і вмісту сторінки.

DHTML побудовано на об'єктній моделі документа DOM (*Document Object Model*), яка розширює традиційний статичний HTML-документ. DOM забезпечує динамічний доступ до вмісту документа, його структури і стилів. В DOM кожен елемент веб-сторінки є об'єктом, який надається до змін. DOM не визначає нових тегів чи атрибутів, а лише забезпечує можливість програмного управління всіма тегами, атрибутами і каскадними листами стилів CSS.

JavaScript – мова сценаріїв

JavaScript – це мова, що призначена для написання сценаріїв для інтерактивних HTML-сторінок. Мова JavaScript не має жодного відношення до мови Java. Java розроблено фірмою SUN, а JavaScript – фірмою Netscape Communication Corporation. Первинною назвою було «LiveScript», але, з огляду на велику популярність мови Java, назву «LiveScript» з комерційних міркувань було змінено на «JavaScript».

JavaScript не призначено для створення автономних застосувань. Програмний код на JavaScript вписується безпосередньо в текст HTML-документа і інтерпретується браузером в міру завантаження цього документа. За допомогою JavaScript можна динамічно змінювати текст завантаженого HTML-документу і реагувати на події, які пов'язані з діями відвідувача або змінами стану документа чи вікна.

Важливою особливістю JavaScript є об'єктна орієнтованість. Програмісту є доступними численні об'єкти, такі, як документи, гіперпосилання, форми, фрейми тощо. Об'єкти характеризуються описовою інформацією (властивостями) і можливими діями (методами).

PHP - мова створення сценаріїв

PHP (*Personal Home Page*) - мова створення сценаріїв, яка давно переросла свою назву. Перша версія PHP була створена Расмусом Лердорфом в 1994 р. і була набором інструментів для відстеження поведінки відвідувачів сайту. З часом PHP з набору інструментів перетворилася на

повноцінну мову програмування, а її назву було змінено як *PHP HyperText Preprocessor* (препроцесор гіпертексту PHP).

PHP - це серверна мова. Конструкції PHP, що вставлено в HTML-текст, виконуються сервером при кожному відвідуванні сторінки. Результат обробки конструкцій разом із звичайним HTML-текстом передається браузеру.

Основними конкурентами PHP є ASP (*Active Server Pages*) від компанії Microsoft і ColdFusion від компанії Allaire.

У порівнянні з ними PHP має ряд переваг, зокрема:

- Висока продуктивність. PHP-програми працюють швидше, ніж ASP.
- Функціональність. Розробку PHP-програми можна відокремити від власне розробки веб-сторінки, що спрощує працю і для програміста і для дизайнера.
- Ціна. Мова PHP є абсолютно безкоштовною.
- Простота у використанні. Програмісти, що мають досвід програмування на поширених мовах швидко зрозуміють синтаксис PHP.
- Переносимість. Один і той же PHP-код можна використовувати як в середовищі Windows, так і на платформах UNIX.

ASP - активні сторінки сервера

ASP (*Active Server Pages*) —технологія, що є аналогічною до JavaScript і PHP. Для того, щоб зробити інтерактивну веб-сторінку із застосуванням макромови ASP, необхідно вбудувати в її код відповідний скрипт, що віддалено нагадує Java і CI. Скрипт інтерпретується і виконується безпосередньо на сервері, після чого до браузера відправляється вже готовий HTML-документ з результатами роботи сценарію ASP. Для сторінок, що містять конструкції ASP, не має значення, яке програмне забезпечення встановлено на комп'ютері користувача, але принципове значення має тип сервера, який має підтримувати дану технологію.

XML - розширена мова розмітки

Основну увагу в мові XML зосереджено на даних. Тут, структурна розмітка даних і представлення даних є строго розділеними.

Основні причини створення XML:

- Спроба надати могутні засоби форматування і структуризації даних для широкого кола розробників.

- Необхідність в стабільній реалізації мови структуризації документів, де можна легко створювати допоміжні інструменти, що є доступними для звичайних користувачів.

XML є метамовою - спеціальною мовою, якою можна скласти повний опис класу інших мов, якими створено документи. XML містить набір правил, що дозволяють створювати унікальні застосування і підмножини даних.

В багатьох розробників виникають певні труднощі у зв'язку з абстрактністю XML і довільним використанням його методів. Насправді, XML - є досить логічною і добре організованою структурою, вона має чіткий синтаксис, що змушує строго дотримуватися певних правил. Починати вивчення XML слід із застосування вже отриманих знань про HTML. XML, як і HTML, використовує теги та атрибути.

Як мова розмітки веб-документів XML має свої особливості:

- Гнучкість. XML дозволяє обробляти унікальні дані, і незалежно від їх характеру надає адекватні методи для їх зберігання і обробки.
- Можливість налаштування. Гнучкість XML безпосередньо пов'язана з можливістю визначати власні дескриптори, необхідність в яких виникає в процесі рішення задачі.
- Узгодженість. XML відрізняється синтаксичною цілісністю і строгою структурою.

Практично всі сучасні браузери підтримують XML. Вона здатна цілком замінити HTML, як засіб розмітки веб-сторінок, хоча вивчення XML є складнішим за вивчення HTML.

XSLT - розширена мова перетворення листів стилів

Мова XSLT (*eXtensible Stylesheet Language Transformations*) є транслятором, за допомогою якого можна вільно модифікувати початковий текст. XLST грає вирішальну роль в затвердженні XML як універсальної мови збереження і передачі даних. Область застосування XSLT є широкою - від електронної комерції до безпровідного Веб.

Фактична збірка результуючого документа відбувається, коли початковий документ і лист стилів XSLT передаються до синтаксичного аналізатору XSLT (XSLT-процесора).

При використанні XSLT в середовищі Веб синтаксичний аналіз може відбуватися або з боку користувача (в браузері), або з боку сервера.

Перетворення XSLT засновано на шаблонах. XSLT-процесор аналізує початковий документ і намагається знайти відповідний XSL-шаблон. Якщо

такий шаблон знайдено, то виконуються інструкції, що містяться всередині нього.

Ajax - технологія для взаємодії з сервером без перевантаження сторінок

Ajax (*Asynchronous Javascript And XML*) це підхід до створення веб-застосувань за допомогою наступних технологій:

- Стандартизоване представлення засобами XHTML і CSS.
- Динамічне відображення і взаємодія з користувачем за допомогою DOM.
- Обмін і обробка даних у вигляді XML и XSLT.
- Широке застосування мови сценаріїв JavaScript.
- Асинхронні запити за допомогою об'єкту XMLHttpRequest.

В стандартному веб-застосуванні обробкою всієї інформації займається сервер, браузер відповідає лише за взаємодію з користувачем, передачу запитів і виведення отриманих даних у форматі HTML.

В Ajax-застосуванні між користувачем і сервером з'являється ще один посередник – програмний механізм (рушій) Ajax. Він визначає, які запити можна обробити з боку клієнта, а які необхідно виконувати на сервері.

Поведінка сервера теж змінилася. Якщо раніше на кожен запит сервер видавав нову сторінку, то тепер він надсилає лише ті дані, які потрібні клієнту, а рушій Ajax в браузері формує з них HTML-код.

Асинхронність виявляється в тому, що далеко не кожен запит користувача скеровується до сервера, причому зворотне теж справедливо - далеко не кожна реакція сервера обумовлена запитом користувача. Велику частину запитів формує рушій Ajax, причому є можливим, щоб він передбачав запити користувача.

Зрозуміло, що за такою схемою роботи міняється якісне навантаження на сервер - якщо раніше запитів було мало, але кожен з них вимагав значних ресурсів (серверу потрібно витягнути інформацію з бази даних, сформувати з неї веб-сторінку і відправити до браузера), то тепер завдання сервера спрощується (формувати веб-сторінки не потрібно, та і об'єм даних, що передаються є значно меншим), але доводиться більше обробляти запитів.

Створювати застосування на Ajax є трудомістким і складним завданням. Потрібно написати і відлагодити на JavaScript рушій з десятьох чи двадцятьох тисяч рядків коду плюс реалізувати серверну частину.

На сьогодні спостерігається тенденція на стрімке поширення Ajax-застосувань. Масштабні Ajax-проекти представлено у Google - *Google Suggest* (сервіс, що підказує найбільш популярні запити), *Gmail* і *Google Maps*.

Найбільш ґрунтовну переробку програмісти Google зробили з поштовим інтерфейсом, тоді як *Google Suggest* і *Google Maps* дивують не стільки новизною підходу, скільки якістю реалізації.

Adobe Flash

Adobe Flash (раніше відома як Macromedia Flash), або просто Flash — мультимедійна платформа, що призначена для створення векторної анімації і інтерактивних застосувань (зокрема, ігор), а також для інтеграції відеороликів у веб-сторінки.

Основним призначенням даної технології є створення високоякісної інтерактивної анімації, яка має відносно невеликий розмір вихідного файлу. За допомогою Flash розробник має можливість виготовляти барвисті анімаційні заставки, певні елементи яких можуть «реагувати» на рухи миші, міні-ігри, озвучені мультиплікаційні кліпи і багато іншого.

Adobe Flash має інструменти для роботи з векторною, растровою і частково з тривимірною графікою, а також підтримує потокову трансляцію аудіо і відео. Для мобільних пристроїв випущено спеціальну «полегшену» версію платформи Flash Lite, яка має обмежену функціональність з розрахунку на можливості мобільних операційних систем та їх апаратних показників.

Основними засобами розробки є пакети Adobe Flash Professional і Adobe Flash Builder, що дозволяють створювати інтерактивні застосування, зокрема, презентації, інтро-заставки, ігри і мультфільми).

Flash використовує вбудовану мову програмування ActionScript, що базується на ECMAScript і є легкою для вивчення. Програмні модулі імпортуються в документ як аплети і вставляються в потрібний кадр анімації, де повинна відбутися динамічна зміна зображення. За допомогою спеціального редактора можна написати невелику програмку, що керує програванням кліпу, створити елементи, що надаються до індивідуальних налаштувань відвідувачами сайту, генерувати заставку з кількома варіантами продовження. Способів реалізації можливостей ActionScript є багато, але для використання всієї її потужності, необхідно мати певний досвід в програмуванні.

Стандартним розширенням для скомпільованих Flash-файлів (анімації, ігор і інтерактивних застосувань) є .SWF (*Shockwave Flash* або *Small Web Format*). Відеоролики у форматі Flash є файлами з розширенням FLV (*Flash Video*), а Flash в даному випадку використовується лише як контейнер для відеозапису. Розширення FLA відповідає формату робочих файлів в середовищі розробки.

Flash-вміст відтворюється за допомогою цілого ряду програмних засобів, але домінуюче місце на ринку займає офіційний програвач Adobe Flash Player, який поширюється як безкоштовний плагін для більшості сучасних браузерів. У випадку відсутності плеєра, браузер, стикаючись з документом у форматі Flash, як правило, сам зв'язується з відповідним вузлом, після чого починає завантаження і установку Flash Player в автоматичному режимі.

Також SWF-файли можна переглядати за допомогою інших плеєрів, наприклад, Gnash або Swfdec. FLV-файли відтворюються через Adobe Flash Player або через мультимедійні програвачі, такі як Quicktime і Windows Media Player, за наявності відповідних плагінів.

Тема 1.2. Архітектура типових веб-застосувань

1.2.1 Поняття архітектури ПЗ

Архітектура програмного забезпечення (англ. software architecture) — це структура програми або обчислювальної системи, яка містить програмні компоненти, видимі зовні властивості цих компонентів, а також відносини між ними. Цей термін також відноситься до документування архітектури програмного забезпечення. Документування архітектури ПЗ спрощує процес комунікації між зацікавленими особами , дозволяє зафіксувати прийняті на ранніх етапах проектування рішення про високорівневий дизайн системи і дозволяє використовувати компоненти цього дизайну і шаблони повторно в інших проектах.

Компоненти інформаційної системи по виконуваних функціях можна розділити на три шари: шар представлення, шар бізнес-логіки і шар доступу до даних.

Шар представлення - все, що пов'язано зі взаємодією з користувачем: натиснення кнопок, рух миші, отрисовка зображення, виведення результатів пошуку і так далі

Бізнес логіка - правила, алгоритми реакції додатка на дії користувача або на внутрішні події, правила обробки даних.

Шар доступу до даних - зберігання, вибірка, модифікація і видалення даних, пов'язаних з вирішуваною додатком прикладним завданням

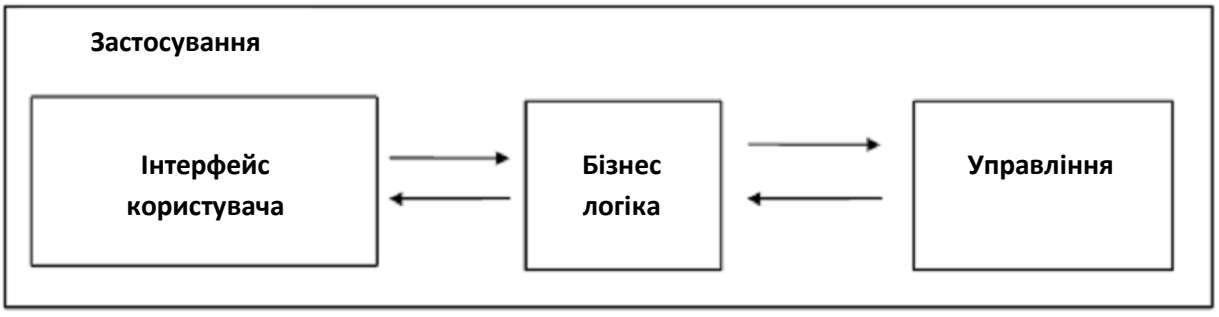


Рисунок 1.4 Компоненти інформаційної системи

1.2.2 Види архітектур сучасних програмних додатків

Історично першими з'явилися комп'ютерні системи з **централізованою архітектурою** додатків. При використанні цієї архітектури усе програмне забезпечення автоматизованої системи виконується централізовано на одному комп'ютері, що виконує одночасно багато завдань і що підтримує велику кількість користувачів. На цьому комп'ютері повністю здійснюється процес введення/ виведення інформації, а також її прикладна обробка. В якості центрального комп'ютера для такої системи може застосовуватися або велика ЕОМ(що називається також майнфреймом) або так звана МІНІ-ЕОМ. До подібного комплексу підключаються периферійні пристрої для введення/виведення інформації від кожного користувача.

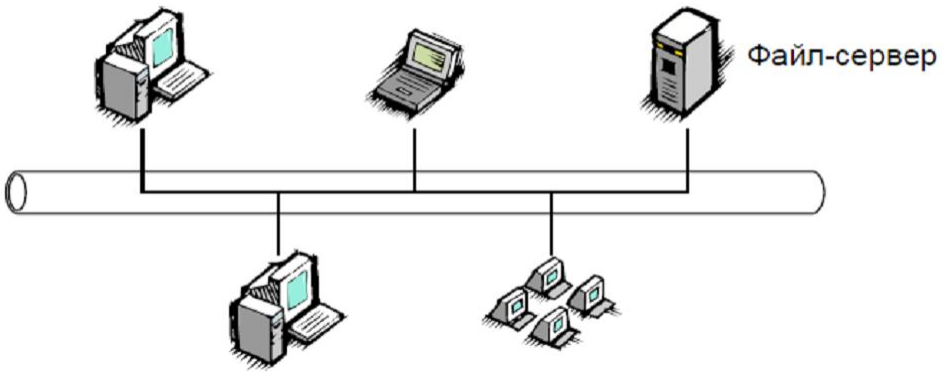


Рисунок 1.5 Файл-серверна архітектура

Наступний етап розвитку архітектури ПЗ — **файл-сервер** — це виділений сервер, призначений для виконання файлових операцій введення-виводу і зберігаючий файли будь-якого типу. Як правило, має великий об'єм дискового простору, реалізованому у формі RAID— масиву для забезпечення безперебійної роботи і підвищеної швидкості запису і читання даних.

Файл-серверні додатки — додатки, схожі по своїй структурі з локальними застосуваннями що використовують мережевий ресурс для зберігання даних у вигляді окремих файлів. Функції сервера у такому разі зазвичай обмежуються зберіганням даних(можливо також зберігання виконуваних файлів), а обробка даних відбувається виключно на стороні клієнта. Кількість клієнтів обмежена десятками зважаючи на неможливість одночасного доступу на запис до одного файлу. Проте клієнтів може бути в рази більше, якщо вони звертаються до файлів виключно в режимі читання.

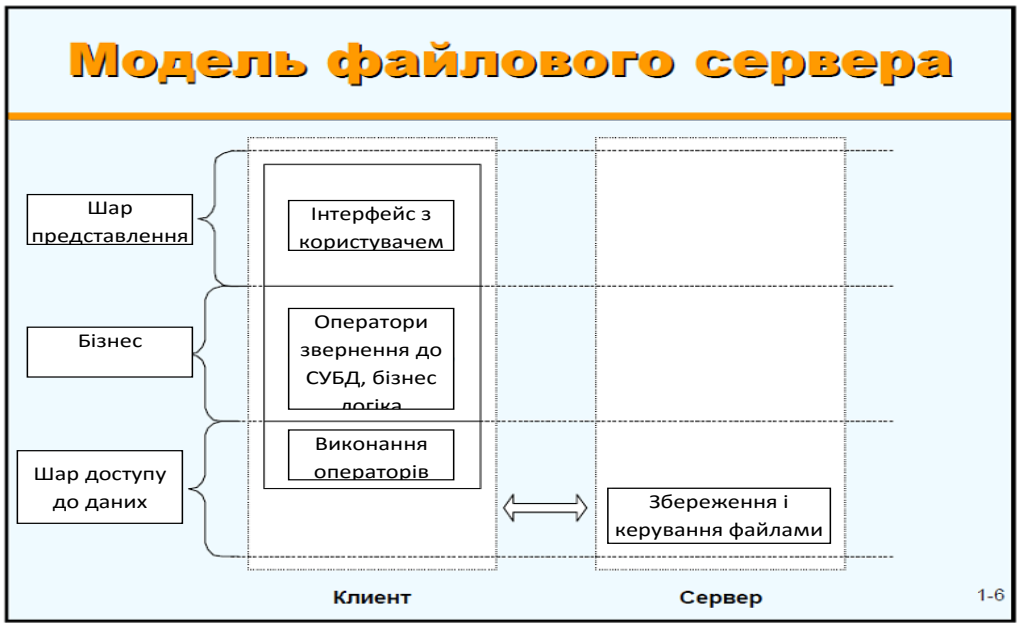


Рисунок 1.6 Модель файлового сервера

Переваги:

- низька вартість розробки;
- висока швидкість розробки;
- невисока вартість оновлення і зміни ПЗ.

Недоліки:

- зростання числа клієнтів різко збільшує об'єм трафіку і навантаження на мережі передачі даних;
- високі витрати на модернізацію і супровід сервісів бізнес-логіки на кожній клієнтській робочій станції;
- низька надійність системи.

Архітектура клієнт-сервер

Архітектура є одним із архітектурних шаблонів програмного забезпечення та є домінуючою концепцією у створенні розподілених мережних застосунків і передбачає взаємодію та обмін даними між ними. Вона передбачає такі основні компоненти:

- набір серверів, які надають інформацію або інші послуги програмам, які звертаються до них;
- набір клієнтів, які використовують сервіси, що надаються серверами;
- мережа, яка забезпечує взаємодію між клієнтами та серверами.

Сервери є незалежними один від одного. Клієнти також функціонують паралельно і незалежно один від одного. Немає жорсткої прив'язки клієнтів до серверів. Більш ніж типовою є ситуація, коли один сервер одночасно

обробляє запити від різних клієнтів; з другого боку, клієнт може звертатися то до одного сервера, то до іншого. Клієнти мають знати про доступні сервери, але можуть не мати жодного уявлення про існування інших клієнтів.

Загальноприйнятим є положення, що клієнти та сервери – це перш за все програмні модулі. Найчастіше вони знаходяться на різних комп'ютерах, але бувають ситуації, коли обидві програми – і клієнтська, і серверна, фізично розміщуються на одній машині; в такій ситуації сервер часто називається локальним

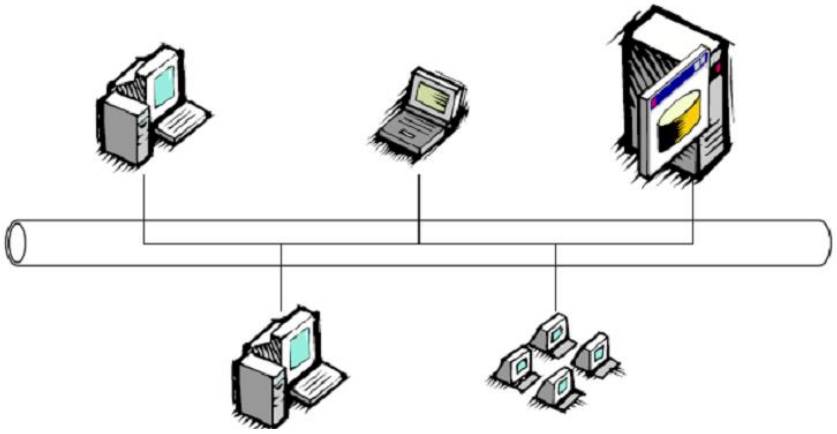


Рисунок 1.7 Архітектура клієнт-сервер

Модель клієнт-серверної взаємодії визначається перш за все розподілом обов'язків між клієнтом та сервером. Логічно можна виокремити три рівні операцій:

- рівень представлення даних, який по суті являє собою інтерфейс користувача і відповідає за представлення даних користувачеві і введення від нього керуючих команд;
- прикладний рівень, який реалізує основну логіку застосунку і на якому здійснюється необхідна обробка інформації;
- рівень управління даними, який забезпечує зберігання даних та доступ до них.

Дворівнева клієнт-серверна архітектура передбачає взаємодію двох програмних модулів – клієнтського та серверного. В залежності від того, як між ними розподіляються наведені вище функції, розрізняють:

- модель тонкого клієнта, в рамках якої вся логіка застосунку та управління даними зосереджена на сервері. Клієнтська програма забезпечує тільки функції рівня представлення;
- модель товстого клієнта, в якій сервер тільки керує даними, а обробка інформації та інтерфейс користувача зосереджені на стороні клієнта. Товстими клієнтами часто також називають пристрої з обмеженою потужністю: кишенькові комп’ютери, мобільні телефони та ін.

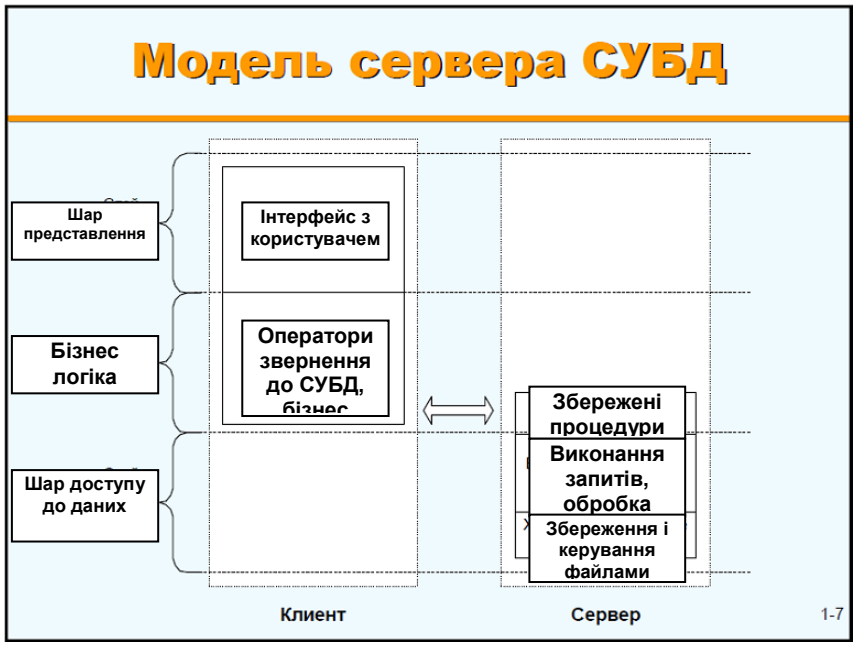


Рисунок 1.8 Архітектура клієнт-сервер

Плюси:

- Повна підтримка розрахованої на багато користувачів роботи
- Гарантія цілісності даних

Мінуси:

- Бізнес логіка додатків залишилася в клієнтському ПЗ. При будь-якій зміні алгоритмів, потрібно оновлювати призначене для користувача ПЗ на кожному клієнтові.
- Високі вимоги до пропускної спроможності комунікаційних каналів з сервером, що перешкоджає використання клієнтських станцій інакше як в локальній мережі.
- Слабкий захист даних від злому, особливо від недобросовісних користувачів системи.
- Висока складність адміністрування і налаштування робочих місць користувачів системи.
- Необхідність використати потужні ПК на клієнтських місцях.
- Висока складність розробки системи із-за необхідності виконувати бізнес-логіку і забезпечувати призначений для користувача інтерфейс в одній програмі

Трирівнева архітектура

Трирівнева архітектура, або триланкова архітектура (англ. three — tier або англ. Multitier architecture) — архітектурна модель програмного комплексу, що припускає наявність в нім трьох компонентів : клієнтського застосування(що зазвичай називається «тонким клієнтом» або терміналом),

сервера додатків, до якого підключено клієнтське застосування, і сервера бази даних, з яким працює сервер додатків.

- Клієнт — це інтерфейсний(зазвичай графічний) компонент, який представляє перший рівень, власне додаток для кінцевого користувача. Перший рівень не повинен мати прямих зв'язків з базою даних(за вимогами безпеки), бути навантаженим основною бізнес-логікою(за вимогами масштабованості) і зберігати стан додатка(за вимогами надійності). На перший рівень може бути винесена і зазвичай виноситься проста бізнес-логіка: інтерфейс авторизації, алгоритми шифрування, перевірка значень, що вводяться, на допустимість і відповідність формату, нескладні операції(сортування, угрупування, підрахунок значень) з даними, вже завантаженими на термінал.
- Сервер додатків розташовується на другому рівні. На другому рівні зосереджена бóльшая частина бізнес-логіки. Поза ним залишаються фрагменти, що експортуються на термінали(см.выше), а також занурені в третій рівень процедури, що зберігаються, і тригери.
- *Сервер бази даних* забезпечує зберігання даних і виноситься на третій рівень. Звичайно це стандартна реляційна або об'єктно-орієнтована СУБД. Якщо третій рівень є базою даних разом з процедурами, що зберігаються, тригерами і схемою, що описує додаток в термінах реляційної моделі, то другий рівень будується як програмний інтерфейс, що зв'язує клієнтські компоненти з прикладною логікою бази даних.

У простій конфігурації фізично сервер додатків може бути поєднаний з сервером бази даних на одному комп'ютері, до якого по мережі підключається один або декілька терміналів.
У «правильній»(з крапки зору безпеки, надійності, масштабування) конфігурації сервер бази даних знаходиться на виділеному комп'ютері(чи кластері), до якого по мережі підключені один або декілька серверів додатків, до яких, у свою чергу, по мережі підключаються термінали.

Переваги

В порівнянні з клієнт-серверною або файл-серверной архітектурою можна виділити наступні перевага трирівневої архітектури :

- масштабованість
- конфігурується — ізольованість рівнів один від одного дозволяє(при правильному розгортанні архітектури) швидко і простими засобами переконфігурувати систему при виникненні збоїв або при плановому обслуговуванні на одному з рівнів

- висока безпека
- висока надійність
- низькі вимоги до швидкості каналу(мережі) між терміналами і сервером додатків
- низькі вимоги до продуктивності і технічних характеристик терміналів, як наслідок зниження їх вартості. Терміналом може виступати не лише комп'ютер, але і, наприклад, мобільний телефон.

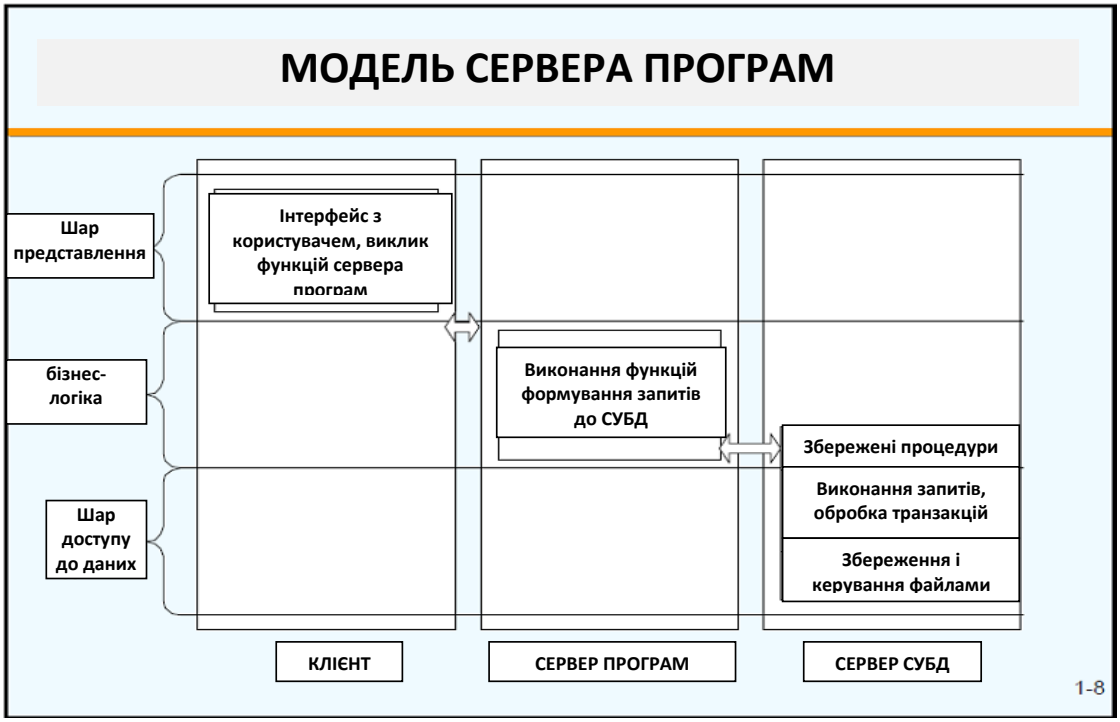


Рисунок 1.9 Архітектура клієнт-сервер

Недоліки

Недоліки витікають з переваг. По порівнянню с клієнт-серверної або файл-серверной архітектурою можна виділити наступні недоліки трирівневої архітектури :

- більш висока складність створення додатків;
- складніше в розгортанні і адмініструванні;
- високі вимоги до продуктивності серверів додатків і сервера бази даних, а, означає, і висока вартість серверного устаткування;
- високі вимоги до швидкості каналу(мережі) між сервером бази даних і серверами додатків.

Розподілені інформаційні системи.

У літературі можна знайти різні визначення розподілених систем, причому жодне з них не є задовільним і не узгоджується з іншими. Для наших завдань вистачить досить вільної характеристики.

Розподілена система — це набір незалежних обчислювальних машин, що представляється їх користувачам єдиною об'єднаною системою.У цьому визначенні обмовляються два моменти. Перший відноситься до апаратури: усі машини автономні.

Другий торкається програмного забезпечення: користувачі думають, що мають справу з єдиною системою. Важливі обидва моменти. Пізніше в цій главі ми до них повернемося, але спочатку розглянемо деякі базові питання, що стосуються як апаратного, так і програмного забезпечення.

Характеристики розподілених систем :

1. Від користувачів приховані відмінності між комп'ютерами і способи зв'язку між ними. Те ж саме відноситься і до зовнішньої організації розподілених систем.

2. Користувачі і додатки однаково працюють в розподілених системах, незалежно від того, де і коли відбувається їх взаємодія.

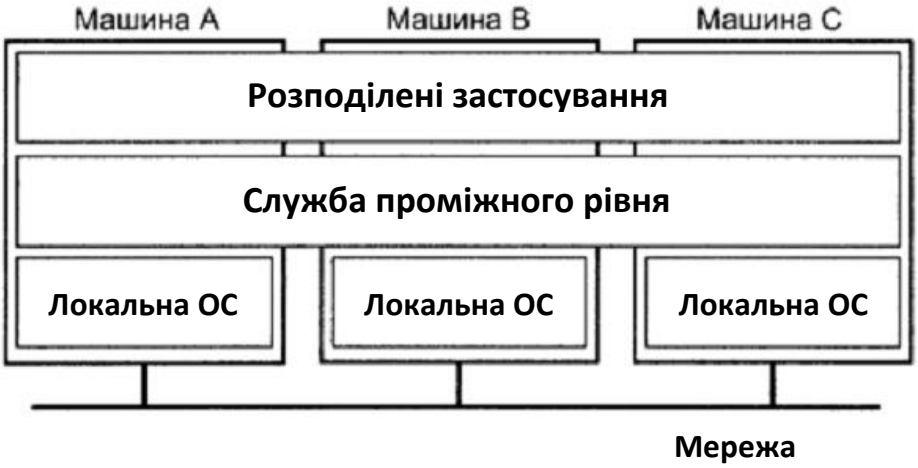


Рисунок 1.10 Модель розподіленої системи

Розподілені системи повинні також відносно легко піддаватися розширенню, або масштабуванню. Ця характеристика є прямим наслідком наявності незалежних комп'ютерів, але в той же час не вказує, яким чином ці комп'ютери насправді об'єднуються в єдину систему.

Розподілені системи зазвичай існують постійно, проте деякі їх частини можуть тимчасово виходити з ладу. Користувачі і додатки не повинні повідомлятися про те, що частини системи замінені або полагоджені або, що додані нові для підтримки додаткових користувачів.

Для того, щоб підтримати представлення системи в єдиному виді, організація розподілених систем часто включає додатковий рівень програмного забезпечення, що знаходиться між верхнім рівнем, на якому знаходяться користувачі і додатки, і нижнім рівнем, що складається з операційних систем.

Сервісно-орієнтована архітектура

Сервісно-орієнтована архітектура (англ. Service-oriented architecture, SOA) — архітектурний шаблон програмного забезпечення, модульний підхід до розробки програмного забезпечення, заснований на використанні розподілених, слабко пов'язаних замінних компонентів, оснащених стандартизованими інтерфейсами для взаємодії за стандартизованими протоколам.

Дуже часто становлення того чи іншого підходу супроводжується появою невірних або хибних трактувань, як це було, наприклад, з концепцією федеративного сховища даних. Не оминуло стороною це і сервіс-орієнтовану архітектуру. Так вважає представник компанії BEA Джерімі Уестерман (Jeremy Westerman). Саме тому в одній із своїх статей, присвячених SOA, він спеціально зупиняється на «проблемних місцях» і наводить докладні пояснення:

- SOA не є чимось новим: IT-відділи компаній успішно створювали і розгортали додатки, що підтримують сервіс-орієнтовану архітектуру, вже багато років — задовго до появи XML і Web-сервісів.
- SOA — це не технологія, а спосіб проектування і організації інформаційної архітектури та бізнес-функціональності.
- Купівля найновіших продуктів, що реалізують XML і Web-сервіси, не означає побудову додатків згідно з принципами SOA.

Джерімі Уестерман дає наступне визначення SOA: це парадигма, призначена для проектування, розробки та управління дискретних одиниць логіки (сервісів) в обчислювальному середовищі. Застосування цього підходу вимагає від розробників проектування додатків як набору сервісів, навіть якщо переваги такого рішення відразу неочевидні. Розробники повинні вийти за межі своїх додатків і подумати, як скористатися вже існуючими сервісами, або вивчити, як їх сервіси можуть бути використані їх колегами.

SOA підштовхує до використання альтернативних технологій і підходів (таких як обмін повідомленнями) для побудови додатків за допомогою зв'язування сервісів, а не за допомогою написання нового програмного коду. У цьому випадку, при належному проектуванні, застосування повідомлень

дозволяє компаніям своєчасно реагувати на зміну ринкових умов — настроювати процес обміну повідомленнями, а не розробляти нові програми. Ще до недавніх пір термін «сервіс-орієнтована архітектура» був синонімом «Web-сервіс». SOA — виклик Web-сервісів за допомогою засобів і мов управління бізнес-процесами. SOA — це термін, який з'явився для опису виконуваних компонентів — таких як Web-сервіси — які можуть викликатися іншими програмами, які виступають у якості клієнтів або споживачів цих сервісів. Ці сервіси можуть бути повністю сучасними — або навіть застарілими — прикладними програмами, які можна активізувати як чорний ящик. Від розробника не потрібно знати, як працює програма, необхідно лише розуміти, які вхідні та вихідні дані потрібні, і як викликати ці програми для виконання. У найзагальнішому вигляді SOA припускає наявність трьох основних учасників: постачальника сервісу, споживача сервісу та реєстру сервісів. Взаємодія учасників виглядає досить просто: постачальник сервісу реєструє свої сервіси в реєстрі, а споживач звертається до реєстру із запитом.

Для використання сервісу необхідно дотримуватися угоди про інтерфейс для звернення до сервісу — інтерфейс повинен не залежати від платформи. SOA реалізує масштабованість сервісів — можливість додавання сервісів, а також їх модернізацію. Постачальник сервісу і його споживач виявляються непов'язаними — вони спілкуються за допомогою повідомлень. Оскільки інтерфейс повинен не залежати від платформи, то і технологія, використовувана для визначення повідомлень, також повинна не залежати від платформи. Тому, як правило, повідомлення є XML-документами, які відповідають XML-схемі.

Дійсно, відкриті стандарти, що описують XML і Web-сервіси, дозволяють застосовувати SOA до всіх технологій і додатків, встановлених в компанії. Як відомо, Web-сервіси базуються на широко поширених і відкритих протоколах: HTTP, XML, UDDI, WSDL і SOAP. Саме ці стандарти реалізують основні вимоги SOA — по-перше, сервіс повинен піддаватися динамічному виявленню і викликом (UDDI, WSDL і SOAP), по-друге, повинен використовуватися незалежний від платформи інтерфейс (XML). Нарешті, HTTP забезпечує функціональну сумісність.

Переваги використання SOA

Перш ніж, перерахувати переваги використання SOA, буде доречним нагадати, що переваги бувають різні: стратегічні і тактичні. SOA має ряд переваг як стратегічних, так і тактичних.

Стратегічна цінність SOA:

- Скорочення часу реалізації проектів, або «часу виходу на ринок».

- Підвищення продуктивності.
- Більш швидка і менш дорога інтеграція додатків і інтеграція B2B — зупинимося більш детально на цьому пункті.

Тактичні переваги SOA:

- Простіша розробка і впровадження додатків.
- Використання поточних інвестицій.
- Зменшення ризику, пов'язаного з впровадженням проектів в області автоматизації послуг і процесів.
- Можливість безперервного поліпшення наданої послуги.
- Скорочення числа звернень за технічною підтримкою.
- Підвищення показника повернення інвестицій (ROI).
- Перспективи

Контрольні питання до розділу 1

1. Розкрийте суть еволюції епох Веб.
2. Основні принципи Веб 2.0
3. Переваги і недоліки Веб 2.0
4. Наведіть приклади сайтів Веб 2.0
5. Порівняйте Веб 2.0 та Веб 3.0
6. Переваги і недоліки Веб 3.0
7. Наведіть приклади сайтів Веб 3.0
8. Назвіть основні засоби веб-технологій.
9. Що таке архітектура програмного забезпечення
10. З чого складається інформаційна система
11. Які є види сучасного ПЗ?
12. Порівняйте файл-серверну та клієнт-серверну архітектуру.
13. Переваги і недоліки трирівневої архітектури.
14. Принцип роботи розподілених систем.
15. SOA, її суть і особливості.

РОЗДІЛ 2. ПРОГРАМНІ ЗАСОБИ СТВОРЕННЯ І ПІДТРИМКИ КЛІЄНТ-СЕРВЕРНИХ ЗАСТОСУВАНЬ

Тема 2.1. Основи мови Java Script

2.1.1 Загальний огляд мови JavaScript.

JavaScript – це мова програмування, що використовується в складі HTML-сторінок для збільшення їх функціональності та можливостей взаємодії з користувачем. JavaScript є однією із складових динамічного HTML. Ця мова програмування була створена фірмами Netscape та Sun Microsystems на базі мови програмування Sun's Java. На сьогодні є декілька версій JavaScript. Однією із найбільш поширених є версія JavaScript 1.3. За допомогою JavaScript на HTML-сторінці можливо зробити те, що не можливо зробити за допомогою стандартних тегів HTML.

Код програми JavaScript розміщується або в середині HTML-сторінки, або в текстовому файлі, що пов'язаний за допомогою спеціальних команд з HTML-сторінкою. Цей код, як правило, розміщується в середині тегу HTML та завантажується в браузер разом з кодом HTML-сторінки. Програма JavaScript не може існувати самостійно, тобто без HTML-сторінки.

Виконання програми JavaScript відбувається при перегляді HTML-сторінки в браузері, звісно, тільки в тому випадку, коли браузер містить інтерпретатор JavaScript. Практично всі сучасні популярні браузери оснащені таким інтерпретатором. Відзначимо, що крім JavaScript на HTML-сторінках можливо використовувати інші мови програмування. Наприклад, VBScript або JScript, яка є варіантом JavaScript від фірми Microsoft. Але виконання програм VBScript та JScript гарантовано коректне тільки при перегляді HTML-сторінки за допомогою браузеру Microsoft Internet Explorer. Тому в більшості випадків використання JavaScript доцільніше, хоча функціональність програм VBScript та JScript дещо краща.

Досить часто програму JavaScript називають *скриптом* або *сценарієм*. Скрипти виконуються в результаті того, що відбулась деяка подія, пов'язана з HTML-сторінкою. В багатьох випадках виконання вказаних подій ініціюється діями користувача.

Скрипт може бути пов'язаний з HTML-сторінкою двома способами:

За допомогою парного тегу SCRIPT;

Як оброблювач події, що стосується конкретного тегу HTML.

Сценарій, вбудований в HTML-сторінку з використанням тегу SCRIPT, має наступний формат:

```
<SCRIPT>
   // Код програми
</SCRIPT>
```

Все, що розміщується між тегами <SCRIPT> та </SCRIPT>,

інтерпретується як код програми на мові JavaScript. Обсяг вказаного коду не обмежений. Інколи скрипти розміщують в середині HTML-коментарію. Це роблять для того, щоб код JavaScript не розглядався старими браузерами, які не мають інтерпретатора JavaScript. В цьому випадку сценарій має формат:

```
<SCRIPT>
    <!--
   // Код програми
    -->
</SCRIPT>
```

Тег SCRIPT має декілька необов'язкових параметрів. Найчастіше використовуються параметри *language* та *src*. Параметр *language* дозволяє визначити мову та версію мови сценарію. Параметр src дозволяє задати файл з кодом сценарію. Для пояснення використання параметрів тегу SCRIPT розглянемо задачу.

Задача. Необхідно для HTML-сторінки hi.htm створити сценарій на мові JavaScript 1.3 для показу на екрані вікна повідомлення з текстом "Привіт!".

Відзначимо, що для показу на екрані вікна повідомлення можливо використати функцію alert.

Для ілюстрації можливостей пов'язування скриптів з HTML-кодом вирішення задачі реалізуємо двома варіантами.

Варіант 1. Визначення сценарію безпосередньо на HTML-сторінці hi.htm

```
<html><head>
    <title>Використання JavaScript</title>
</head>
<body>
<script language="JavaScript1.3">
   alert('hi');
</script>
</body></html>
```

Варіант 2. Визначення сценарію в файлі a.js, пов'язаному з HTML-сторінкою hi.htm за допомогою параметру src тегу SCRIPT. Код HTML-сторінки hi.htm:

```
<html><head>
    <title>Використання JavaScript</title>
    <script language="JavaScript1.3"
            src="a.js"> </script>
</head><body>
</body></html>
```

Програмний код, записаний в файлі a.js:

alert('hi');

Результат виконання обох варіантів вирішення задачі однаковий і показаний на рисунку 2.1.

Рисунок 2.1 Показ вікна повідомлення засобами JavaScript

Змінні та вирази JavaScript можливо використовувати в якості значень параметрів тегів HTML. В цьому випадку елементи JavaScript розміщуються між амперсандом (&) та крапкою з комою (;), але повинні бути обмежені фігурними дужками {} і використовуватись тільки в якості значень параметрів тегів.

Наприклад, нехай визначена змінна *c* і їй присвоєно значення *green*. Наступний тег буде виводити текст зеленого кольору:

```
<font color="&{c};"> текст зеленого кольору </font>
```

Відзначимо, що мінімальним комплектом програмного забезпеченням для розробки та тестування програм JavaScript є текстовий редактор та браузер з підтримкою JavaScript.

Синтаксис. Визначення та ініціалізація змінних

Сценарій JavaScript являє собою набір операторів, що послідовно інтерпретуються браузером. Оператори можливо розміщувати як в одному, так і в окремих рядках. Якщо оператори розміщені в одному рядку, то між ними необхідно поставити ;. В протилежному випадку ; не обов'язкова. Будь-який оператор можливо розмістити в декількох рядках без символу продовження.

Будь-яка послідовність символів, розміщених в одному рядку, якій передує //, розглядається як *коментар*. Для визначення *багаторядкових коментарів* використовується конструкція:

```
/*
Багаторядковий коментар
*/
```

В мові JavaScript рядкові та приписні букви вважаються різними

символами. JavaScript використовує змінні для зберігання даних визначеного типу. При цьому JavaScript є мовою з вільним використанням типів. Тобто не обов'язково задавати тип змінної, який залежить від типу даних, що в ній зберігаються. При зміні типу даних автоматично змінюється і тип змінної.

JavaScript підтримує чотири простих типи даних. Ілюстрацією цього є табл. 2.1. Для присвоєння змінним значень основних типів використовуються *літерали*.

Таблиця 2.1 Типи даних JavaScript

Тип даних	Пояснення	Приклад літерала
Цілий	Послідовність цілих чисел	789 +456 -123
З плаваючою крапкою	Числа з крапкою, яка відділяє цілу частину від дробової, або числа в науковій нотації	7.25 0.525e01 71.2E-4
Рядковий	Послідовність алфавітно-цифрових символів, взятих в одинарні (’), або подвійні (") лапки.	"Привіт" "234" "Hello World!!!"
Булевий або логічний	Використовуються для оброблення ситуацій так/ні в операторах порівняння	true false

Ім'я змінної повинно містити тільки букви латинського алфавіту, символ підкреслення _, арабські цифри та починатись з букви або символу підкреслення _. Довжина імені повинна бути менша від 255 символів. Заборонено використовувати імена, що збігаються з ключовими словами JavaScript. Приклади імен: _hello, go, go123.

Визначити змінну можливо:

Оператором *var*, наприклад, var a;

При *ініціалізації*, за допомогою оператору присвоєння (=), наприклад, b=126.

Визначення та ініціалізацію змінних можливо реалізувати в будь-якому місці програми.

Перетворення типів

Тип змінної залежить від того, який тип інформації в ній зберігається. JavaScript - слаботипізована мова. Це означає, що в декларації змінної не вказується його тип і надалі можна надавати їй значення будь-яких типів. Виконуюча система JavaScript сама виконує автоматичне перетворення типів даних у міру необхідності. Для явного перетворення типів використовуються методи Boolean, Number, Object і String.

Тип змінної привласнюється змінній автоматично протягом виконання

скрипта. Так, наприклад, ви можете визначити змінну в такий спосіб:

```
var answer=42
```

А пізніше, ви можете присвоїти тій же змінній значення, що наприклад випливає:

```
answer="Thanks for all the fish..."
```

Або розглянемо наступний вираз:

```
//приклад
var onestring="1"
var oneint=1
var oneconcatenate=onestring+oneint
// У результаті виходить "11"
var oneaddition=oneint+onestring
// У результаті виходить 2
```

У першій операції додавання перший операнд є рядком. Javascript припускає, що проводиться операція із двома рядками. Коли Javascript виявляє в якості другого операнда ціле число, він у відповідності зі своїми виставами перетворить змінну в рядок.

Оскільки Javascript вільно типізований мова, то це не викличе помилки.

Тому що Javascript не підтримує ніяких методів і властивостей для визначення типу поточного значення змінної, дуже важливо уважно відстежувати типи змінних щоб уникнути несподіваних результатів.

Взагалі, у виразах рядкові значення, що включають числові Javascript перетворює числові значення в строкові. Наприклад, розглянемо наступні твердження:

```
x="The answer is " + 42
y=42 + " is the answer."
```

Перше твердження буде рядок "The answer is - 42 ". Друге твердження повертає рядок " 42 - The answer is".

Вирази та оператори

Вираз – це комбінація змінних, літералів та операторів, в результаті обчислення яких можливо отримати тільки одне значення, яке може бути числовим, рядковим або булевим.

Для реалізації обчислень в JavaScript використовуються арифметичні, рядкові, логічні вирази та декілька типів операторів.

Арифметичні вирази – обчислюють число, наприклад, a=7+5;

Рядкові вирази – обчислюють рядок символів, наприклад, "Джон" або "234";

Логічні вирази – обчислюють true (істина) або false (хибна).

Оператор присвоювання (=) – присвоює, значення лівому операнду, базуючись на значенні правого операнда. Наприклад, для присвоєння змінній *a* значення числа 5 необхідно записати:

a=5

До стандартних *арифметичних операторів* відносяться: оператори

додавання (+), віднімання (-), множення (*), ділення (/), остача від ділення чисел (%), збільшення числової змінної на 1 (++), зменшення числової змінної на 1 (--).

Відзначимо, що оператор додавання можна використовувати не тільки для чисел, але й для додавання (контрактації / конкатенації) текстових рядків.

Для створення логічних виразів використовуються *логічні оператори* та *оператори порівняння*.

До логічних операторів відносяться – логічне *І* (&&), логічне *АБО* (||), логічне *НІ* (*!*).

Оператори порівняння не відрізняються від таких операторів в інших мовах програмування. До операторів порівняння відносяться (==, >, >, =<, <=,*!*=).

2.1.2 Умовні та циклічні оператори

Оператори вибору відносяться до *операторів управління,* призначенням яких є зміна напрямку виконання програми. Крім операторів вибору до операторів управління відносяться: оператори циклу та оператори маніпулювання об'єктами.

Оператори вибору призначені для виконання деяких блоків операторів в залежності від істинності деякого логічного виразу. До операторів вибору відносяться: оператор умови *if...else* та перемикач *switch*.

Синтаксис оператора умови такий:

```
if (умова) {
    група операторів 1
    . . . . . .
}
[else] {
    група операторів 2
    . . . . . .
}
```

Перша група операторів виконується при умові істинності виразу умова. Необов'язковий блок *else* визначає другу групу операторів, яка буде виконуватись в випадку хибності умови, заданої в блоці if. В середині групи операторів можуть бути використані будь-які інші оператори, в тому числі і інші оператори умови. Це дозволяє створювати групу вкладених операторів умови if та реалізовувати складні алгоритми перевірки. Однак, якщо кількість вкладених операторів if більша ніж три, то програма стає складною для розуміння. В такому випадку доцільно використовувати оператор *switch.* В цьому операторі обчислюється деякий вираз та порівнюється з значенням, заданим в блоках *case*. Синтаксис оператора switch такий:

```
switch  (вираз) {
case значення1:
[оператори1]
break;
```

```
case значення2:
[оператори2]
break;
...
default:
[оператори]
}
```

Якщо значення виразу в блоці *switch* дорівнює *значення1*, то виконується група операторів *оператори1*, якщо дорівнює *значення2*, то виконується група операторів *оператори2* і так далі. Якщо значення виразу не дорівнює ні одному із значень, що задані в блоках *case*, то обчислюється група операторів блоку *default*, якщо це блок заданий, інакше – виконується вихід із оператору *switch*. Необов'язковий оператор *break,* який можливо задавати в кожному із блоків *case*, виконує безумовний вихід із оператору *switch*. Якщо він не заданий, то продовжується виконання операторів в наступних блоках *case* до першого оператору *break* або до кінця оператору *switch*.

Оператори циклу

Цикл – це деяка група команд, що повторюється доки вказана умова не буде виконана. JavaScript 1.3 підтримує дві форми циклу: *for* та *while*. Крім того оператори *break* та *continue* використовуються разом з циклами.

Цикл for повторює групу команд до тих пір, доки вказана умова хибна. Синтаксис оператору for такий:

```
for([initial-expression];[condition];[increment-expression])
{
   statements
}
```

Виконання циклу *for* проходить в такій послідовності:

1. Вираз *initial-expression* служить для ініціалізації змінної лічильника. Цей вираз розраховується один раз на початку виконання циклу

2. Вираз *condition* розраховується на кожній ітерації циклу. Якщо значення виразу *condition* дорівнює true, виконується група операторів *statements* в тілі циклу. Якщо значення виразу *condition* дорівнює false, то цикл for закінчується. Якщо вираз *condition* пропущено, то він вважається рівним true. В цьому випадку цикл продовжується до оператора break.

3. Вираз *increment-expression* використовується для зміни значення змінної лічильника.

4. Розраховується група операторів *statements* та реалізується перехід на наступну ітерацію циклу, тобто на крок 2.

Приклад. Цикл для розрахунку суми цілих чисел від 1 до 100.

```
s=0
for (i=1;i<101;i++) {
    s=s+1;
}
```

Оператор *while* повторює цикл, доки вказана умова істина. Оператор while виглядає таким чином:

```
while (condition) {
    statements
}
```

Цикл *while* виконується таким чином. Спочатку перевіряється умова *condition.* Якщо умова істинна, то виконується група операторів *statements* в середині циклу. Перевірка істинності виконується на кожному кроці циклу. Якщо умова хибна, то цикл закінчує своє виконання.

Інколи необхідно закінчити цикл не по умові, що задана в його заголовку, а в результаті виконання деякої умови в тілі циклу. Для цього використовуються оператори *break* та *continue*. Оператор *break* завершує цикл while або for та передає керування програмою першому оператору після циклу. Оператор continue передає управління оператору перевірки істинності умови в циклі while та оператору оновлення значення лічильника в циклі for і продовжує виконання циклу.

Оператор `for…in` виконує задані дії для кожної властивості об'єкта чи для кожного елемента масиву і має такий вигляд:

```
for (змінна in вираз) оператор
```

Оператор `for…in` діє таким чином:

1. *Змінній* надає назву чергової властивості об'єкту чи чергового елемента масиву (залежно від природи *виразу*).
2. Виконують *оператор*.
3. Переходять до етапу 1.

При ітерації властивостей об'єкту неможливо передбачити, в якому порядку їх буде проглянуто. Але їх буде проглянути усі без виключення. Подамо приклад створення об'єкту `ob` з наступним послідовним виведенням усіх його властивостей на екран користувача.

```
<html><script>
var ob = {"а": "Літера а", "б" : 2012};
for (var key in ob) document.write(key+": "+ob[key]
    +"<BR>");</script></html>
```

На екрані побачимо такий текст:

а: Літера а

б: 2012

Оператор `with` задає назву об'єкту за замовчуванням і має такий вигляд:

`with` (*вираз*) *оператор*

Цей оператор діє таким чином. Для кожної назви в *операторі* перевіряють, чи є вона назвою властивості об'єкту, заданого згідно із замовчуванням. Якщо відповідь ствердна, то цю назву вважають назвою відповідної властивості, інакше — назвою змінної. Це допомагає скоротити

код. Например, для доступу до математичних функцій маємо кожного разу вказувати назву об'єкту Math, як у такому коді:

```
x = Math.cos(Math.PI/2) + Math.sin(Math.LN10) +
Math.tan(2 * Math.E);
```

Того самого можна досягнути таким чином:

```
with (Math) { x=cos(PI/2) +sin(LN10)+tan(2 * E);};
```

Оператор with можна застосовувати лише до наявних методів.

2.1.3 Обробка виключних ситуацій

При виконанні сцернарію можливе винекнення помилок, які називають *виключеннями*: звертання до відсутнього об'єкта чи неможливість перетворення величини до заданого типу тощо.

Оператор try…catch використовують для опрацювання такого виключення. Він має такий вигляд:

try { *оператор*$_1$ } catch (*виключення*) { *оператор*$_2$ }

Тут *виключення* — довільна назва змінної, *оператор*$_1$ — містить код, що може спричинити виключення. Якщо виключення не відбулося, то після виконання *оператору*$_1$ здійснюють перехід до наступного за try…catch оператору. Інакше інформацію про виключення зберігають як величину локальної змінної *виключення*, а керування передають *оператор*$_2$, який має містити код опрацювання цього виключення. Якщо виключення неможливо на даному рівні опрацювати, то *оператор*$_2$ має містити оператор throw для переходу до опрацювання виключення на вищому рівні (див. далі).

Оператор throw породжує виключення, яке вже можна опрацювати оператором try…catch. Він має такий вигляд:

```
throw expression
```

Тут *expression* — будь-який вираз. Результат обчислення expression буде кинутий як виняток.

Подамо приклад породження й опрацювання виключення.

```
<html><body><script>
var e;
function getMonthName(m)
  {    var    months=new    Array    ("січень","лютий",
  "березень","квітень","травень","червень","липень",
  "серпень", вересень","жовтень","листопад","грудень");
  m = m - 1;
  if (months[m] != null) return months[m]
  else throw "trow";
}
```

```
try        { monthName = getMonthName(7)}
catch (e) { monthName="Неправильний № місяця"};
document.write(monthName);
</script></body></html>
```

При заміні (7) на (77) замість слова «липень на екрані побачимо текст: «Неправильний № місяця».

2.1.4 Використання функцій в JavaScript

Функція – JavaScript це іменована група команд, які вирішують певну задачу та можуть повернути деяке значення. Функція визначається за допомогою оператору *function* , що має такий синтаксис:

```
function Ім'я_функції ([параметри])
{
[оператори]
return [значення_що повертається]
}
```

Параметри, що передаються функції, розділяються комами. Необов'язковий оператор *return* в тілі функції (блок операторів, що обмежений фігурними дужками), визначає значення, що повертається функцією. Визначення функції тільки задає її ім'я и визначає, що буде робити функція при її визові. Безпосереднє виконання функції реалізується, коли в сценарії відбувається її виклик та передаються необхідні параметри. Відзначимо, що визначення функції необхідно реалізувати на HTML-сторінці до її виклику. Наприклад, для показу на екрані вікна повідомлення з текстом "Це виклик функції" визначимо функцію Go та реалізуємо її виклик:

```
<html><head><title>Використання JavaScript</title>
    <script>
    function Go() {
    alert("Це виклик функції")
    }
    </script>
</head><body>
<script>
Go();
</script></body></html>
```

Тема 2.2. Об’єктна модель Java Script

JavaScript відноситься до об’єктно-орієнтованих мов програмування.

В JavaScript всі елементи (теги) на HTML-сторінці вистроєні в ієрархічну структуру. Причому кожен елемент представлений в вигляді об’єкту, з визначеними властивостями та методами. Керування об’єктами на HTML-сторінці можливо багато в чому за рахунок того, що JavaScript дозволяє одержати доступ до цих властивостей та методів. При реалізації доступу необхідно враховувати ієрархію об’єктів на HTML-сторінці. Відзначимо, що загальним об’єктом контейнером є об’єкт *window*, який відповідає вікну браузера. В свою чергу цей об’єкт містить деякі елементи оформлення, наприклад рядок стану. Завантаженій в вікно браузера HTML-сторінці відповідає об’єкт *document*. Всі без виключення елементи HTML-сторінки є властивостями об’єкту document. Прикладами об’єктів HTML є таблиця, гіперпосилання або форма.

2.2.1 Операції з об’єктами JavaScript

Об'єкт - це складений тип даних, він об'єднує безліч значень в єдиний модуль і дозволяє зберігати і витягати значення по їх іменах. Говорячи іншими словами, об'єкти - це неврегульовані колекції властивостей, кожне з яких має своє ім'я і значення. В якості властивостей можуть бути звичайні змінні, що містять значення, або об'єкти.

Створення об'єктів

У JavaScript існує два способи створення об'єктів - за допомогою оператора new за яким йде функція конструктор або за допомогою літерала об'єкту :

```
var empty = new Object(); //порожній об'єкт створюється за допомогою оператора new
var empty ={};  //порожній об'єкт створюється за допомогою літерала об'єкту
```

Літерал об'єкту - це поміщений у фігурні дужки список з нуля або більше властивостей (пара ім'я: значення), що розділені комами. Ім'ям властивості може бути ідентифікатор або рядковий літерал. Значенням властивості може будь-яке значення примітивного типу або типу посилального, а також будь-який виражз, допустимий в JavaScript, отримане значення виразу стане значенням властивості.

```
var empty ={}; // порожній об'єкт (без властивостей)
var point ={x :0,y :0}; // об'єкт з двома властивостями x і y зі значеннями 0
var homer ={ // об'єкт з декількома властивостями
  "name": "Гомер Сімпсон"
```

```
  age: 34,
  married: true
};
```

Зверніть увагу, що після фігурної дужки повинна стояти крапка з комою.

Зазвичай для доступу до значень властивостей об'єкту використовується оператор крапка . (крапка). Значення в лівій частині оператора має бути посиланням на об'єкт, до властивостей якого вимагається отримати доступ. Значення в правій частині оператора має бути ім'ям властивості. Властивості об'єкту працюють як змінні: в них можна зберігати значення і прочитувати їх:

```
var homer ={
  name: "Гомер Сімпсон"
  age: 34,
  married: true
};
homer.age = 35; //надаємо властивості нове значення
document.write(homer.age + "<br>"); //виведення значення властивості age
homer.male = "чоловік"; //додаємо в об'єкт нову властивість зі значенням
document.write(homer.male); //виведення значення нової властивості об'єкту
```

В даному прикладі важливо звернути увагу на два моменти - нова властивість об'єкту, можна додати у будь-якому місці, просто присвоївши цій властивості значення, так само і значення властивостей вже створених, можна змінювати у будь-який момент, наприклад простим наданням нового значення.

Як ви могли вже помітити - значення властивостей вказаних усередині літерала об'єкту вказуються після двокрапки, якщо додається нова властивість або привласнюється нове значення вже існуючому за межами літерала об'єкту, то замість двокрапки використовується операція привласнення.

Як згадувалося раніше, значенням властивості може бути об'єкт:

```
var obj ={
  name: "Гомер"
  colors: {
    first: "yellow"
    second: "blue"
  }
};//для доступу до значень властивостей використовується стандартний синтаксис
document.write(obj.colors.first);
```

Значенням властивості colors є об'єкт {first: "yellow", second: blue }. Тут

слід звернути увагу на те, що у об'єкту, який виступає значенням, у кінці відсутня крапка з комою. Це пояснюється тим що в літералі об'єкту усі властивості вказуються через кому і ніяких крапок з комою там не повинно бути використано.

Перевірка, перерахування і видалення властивостей

Для видалення властивостей об'єкту використовується оператор delete.

```
var homer ={
  name: "Гомер Сімпсон",
  age: 34,
  married: true
};
delete homer.age;
```

Зверніть увагу, що при видаленні властивості його значення не просто встановлюється в значення undefined, оператор delete дійсно видаляє властивість з об'єкту. Цикл for in демонструє цю відмінність: він перераховує властивості, яким присвоєно значення undefined, але не перераховує видалені властивості.

Цикл за властивостями for in, який згадувався в главі "цикли" дозволяє здійснювати послідовний перебір усіх властивостей об'єкту. Його можна використати наприклад при відладці сценаріїв, при роботі з об'єктами, які можуть мати довільні властивості із заздалегідь невідомими іменами, для виведення імен властивостей на екран або для роботи з їх значеннями.

Перед виконанням циклу ім'я однієї з властивостей привласнюється змінній у вигляді рядка. У тілі циклу цю змінну можна використати як для отримання імені властивості так і для набуття значення властивості за допомогою оператора [].

```
var homer ={
  name: "Гомер Сімпсон",
  age: 34,
  married: true
};
 document.write("до видалення властивості : <br>")
for (var name in homer){
  document.write(name+"<br>"); //виводить імена властивостей
}
 delete homer.age;
 document.write("<br> після видалення властивості :
<br>")
for (var name in homer){
  document.write(name + " = " + homer[name] +
"<br>"); //виводить імена і значення властивостей
```

```
}
```

На замітку: цикл for in не перераховує властивості в якому-небудь заданому порядку, і хоча він перераховує усі властивості, визначені користувачем, деякі зумовлені властивості і методи JavaScript він не перераховує.

Для перевірки факту існування тієї або іншої властивості використовується оператор in. З лівого боку від оператора поміщається ім'я властивості у вигляді рядка, з правого боку - об'єкт, що перевіряється, на наявність вказаної властивості.

```
var obj ={};
if ("a" in obj){
  obj.a = 1;
}
else {
  document.write("Такої властивості не існує");
}
```

При зверненні до неіснуючої властивості повертається значення undefined. Тому, для перевірки так само досить часто використовується інший спосіб - порівняння значення з undefined.

```
if (obj.x !== undefined) obj.x = 1;
```

Різниця між цими двома способами перевірки наступна: перевірка за допомогою порівняння "=== undefined" не працює, якщо значення властивості рівне undefined:

```
var obj ={};
boj.x = undefined;
if (obj.x !== undefined) //false, хоча така властивість існує
```

Оператор in в цьому випадку гарантує точний результат

```
var obj ={};
boj.x = undefined;
if ("x" in obj) //true, оскільки така властивість існує
```

У реальних програмах значення undefined не привласнюють, цей приклад був наведений, щоб показати різницю перевірки на існування властивості, якщо значення властивість undefined.

Доступ до властивості через квадратні дужки []

Як ми знаємо, доступ до властивостей об'єкту здійснюється за допомогою оператора "крапка". Доступ до властивостей об'єкту можливий також за допомогою оператора [], який зазвичай використовується при роботі з масивами. Таким чином, наступні два вирази мають однакове значення і на перший погляд нічим не відрізняються:

```
obj.property = 10;
obj['property'] = 10;
```

Важлива відмінність між цими двома синтаксисами, на яку слід звернути увагу, полягає в тому, що в першому варіанті ім'я властивості є ідентифікатором, а в другому - рядок.

Бувають випадки, коли ім'я властивості зберігається в змінній, тому в квадратні дужки можна так само передати змінну, якій присвоєно ім'я властивості у виді терміни :

```
var obj ={ name: "Гомер" };
//надаємо змінній у вигляді рядка ім'я властивості
var str = "name";
 //запис obj[str] еквівалентна obj["name"]
document.write(str + ": " + obj[str]);
```

Якщо ім'я властивості зберігається в якості значення в змінній як в прикладі (var str = "name";), то єдиним способом звернутися до нього - це через квадратні дужки (obj[str]), це буде теж саме, що і (obj["name"]).

Примітка: ще одна різниця між доступом до властивості об'єкту через крапку і [] полягає в тому, що на ім'я властивості при доступі через оператор "крапка" накладені синтаксичні обмеження - це ті ж правила іменування, що і для звичайної змінної, тоді як при зверненні до властивості об'єкту за допомогою оператора [], ім'я властивості задається у вигляді рядка і може містити будь-які символи.

```
obj["Моє ім'я"] = "Гомер";
```

Який же спосіб краще використати при написанні програми? Звичайне звернення до властивості через крапку використовується, якщо ви на етапі написання програми вже знаєте які будуть назви властивостей. А якщо властивості визначатимуться по ходу виконання, наприклад, вводитимуться відвідувачем і записуватися в змінну, то єдиний вибір — квадратні дужки.

Методи об'єкту

Метод - це функція, яка зберігається в якості значення у властивості об'єкту і може викликатися за допомогою цього об'єкту.

```
var obj ={
  name: "Гомер"
    write_hello: function(){
    document.write("Привіт");
  }
};
```

```
obj.write_hello();  //виклик методу
```

Метод повинен мати доступ до даних об'єкту для повноцінної роботи. Для доступу до об'єкту з методу використовується ключове слово this. Воно посилається на об'єкт, в контексті якого викликаний метод і дозволяє звертатися до інших його методів і властивостей :

```
var calc ={
  num1: 5,
  num2: 5,
  compute: function( ){
    this.result = this.num1 * this.num2;
  }
};
calc.compute(); //Обчислюємо скільки буде 5*5?
document.write(calc.result); // Виводимо результат
```

Такий запис this.result = this.num1 * this.num2, в принципі можна читати як calc.result = calc.num1 * calc.num2, оскільки слово this в якості значення містить посилання на об'єкт.

Функція конструктор і оператор new

Окрім літерального синтаксису створення об'єкту, об'єкт можна створювати за допомогою функції - конструктора і оператора new.

Конструктор - це функція, яка виконує ініціалізацію властивостей об'єкту і призначена для використання спільно з оператором new :

```
//визначуваний конструктор
function Car(seats){
  this.seats = seats;
  this.canDrive = true;
}
//викликаємо конструктор для створення нового об'єкту
var myCar = new Car("leather");
```

Давайте розберемо як це усе працює. Оператор new створює новий порожній об'єкт без яких-небудь властивостей, а потім викликає функцію-конструктор (можна називати просто конструктор), передаючи їй тільки що створений об'єкт. Головне завдання конструктора полягає в ініціалізації знову створеного об'єкту - установці усіх його властивостей, які необхідно ініціалізувати до того, як об'єкт зможе використовуватися програмою. Після того, як об'єкт створений і ініціалізував, змінній myCar привласнюється посилання на об'єкт.

Результатом виконання коду з прикладу вище є створення нового екземпляра об'єкту :

```
myCar ={
```

```
    seats: "leather"
    canDrive: true
};
```

Створювані об'єкти таким чином зазвичай називають екземпляром об'єкту (чи класу), в нашому випадку myCar є екземпляром об'єкту Car.

Примітка: при виклику конструктора без аргументів, дужки можна не ставити.

```
var myCar = new Car;
//теж саме, що і
var myCar = new Car();
```

Обробка подій

Важливою ознакою інтерактивних HTML-сторінок є можливість реакції на дії користувача. Наприклад, натиск на кнопці повинен викликати появу діалогового вікна, або виконання перевірки правильності введених користувачем даних. В JavaScript інтерактивність реалізована за допомогою *перехоплення* та *обробки подій*, викликаних в результаті дій користувача. Для цього в теги деяких елементів введені параметри *обробки подій*. Ім’я параметру обробки події починається з префіксу on, за яким йде назва події. Наприклад, події клік кнопкою миші Click, відповідає параметр обробки події з назвою onClick. Назви та характеристики деяких подій наведені в табл. 2.2.

Таблиця 2.2 Події JavaScript

Подія	Характеристика події	Обробник події
Click	Клік кнопкою миші на елементі форми або гіперпосилання	onClick
KeyDown	Натиск на клавіші клавіатури	onKeyDown
Load	Завантажується документ в браузер	onLoad
MouseDown	Натиск на кнопці миші	onMouseDown
MouseOver	Курсор знаходиться над елементом	onMouseOver
MouseOut	Курсор покидає зону над елементом	onMouseOut

Задача. Необхідно, щоб при наведені курсору на комірку таблиці із написом "Привіт" з’являлось вікно повідомлення з фразою "Hello". Можливі рішення:

Варіант 1:

```
<td onClick="alert('Hello')"> Привіт </td>
```

Варіант 2:

```
<script>
function Go() {
alert("Hello")
}
</script>
```

```
<td onClick="Go()"> Привіт </td>
```

В варіанті вирішення 1, код JavaScript був записаний безпосередньо в тезі, а в варіанті 2 наслідком кліку став виклик функції. Варіант 2 слід використовувати, якщо код обробки події великий за обсягом.

2.2.2 Стандартні об’єкти і функції JavaScript

В ядрі JavaScript визначені об’єкти та функції, які можливо використовувати не використовуючи контекст завантаженої сторінки. До основних об’єктів відносяться: Array, Date, Math, String.

Array –масив. Масив це упорядкований набір однотипних даних, до елементів якого можливо звернутись по імені або по індексу. Для створення масиву необхідно використати одну із двох конструкцій:

```
ім’я_масиву=new Array([елемент1],[елемент2],[елемент3],...)
ім’я_масиву = new Array([довжина масиву])
```

Відзначимо, що перший елемент масиву має номер 0.

В першій конструкції в якості параметрів використовуються елементи масиву, в другій конструкції використовується довжина масиву.

Наприклад:

ar1 = new Array(1, 2, 3)

ar2 = new Array(3)

Для доступу до значень елементів масиву в квадратних дужках біля імені масиву необхідно вказати порядковий номер елементу. Наприклад:

a = ar1[2]

ar1[0] = 7

В цьому прикладі, змінній *a* присвоюється значення елементу масиву за номером 2, а в елемент масиву за номером 0 записується значення 7.

Особливістю масивів JavaScript є те, що розмір масиву може встановлюватись динамічно. Наприклад, якщо для масиву із попереднього прикладу написати:

ar1[100] = 7,

то розмір масиву буде автоматично установлений рівним 101.

Для визначення довжини масиву можна скористатись властивістю *length*. Наприклад:

a = ar1. length

Зручність використання масивів забезпечується рядом методів, представлених в табл. 2.3

Таблиця 2.3 Методи об’єкту Array

Метод	Призначення
concat	Об’єднує два масиви в один. var alpha = ["a", "b", "c"]; var numeric = [1, 2, 3];

	// створює масив ["a", "b", "c", 1, 2, 3]; var alphaNumeric = alpha.concat(numeric);

join	Об’єднує всі елементи масиву в один рядок.
	var arr = [1, 2 , 3];
	arr.join('+') ; // "1+2+3"
	arr.join() ; // "1,2,3"

pop	Знищує останній елемент із масиву і повертає його значення.
	myFish = ["angel", "clown", "mandarin", "surgeon"];
	popped = myFish.pop();

push	Додає один або декілька елементів в кінець масиву і повертає останній добавлений елемент. var array = ["one", "two"] // додати елементи "three", "four" var pushed = array.push("three", "four")
reverse	Переставляє елементи масиву в зворотному порядку: перший елемент стає останнім, а останній першим. arr = [1,2,3] a = arr.reverse()
shift	Знищує перший елемент масиву і повертає його значення. var arr = ["мій","маленький", "масив"] var my = arr.shift() // => "мій" alert(arr[0])

slice	Створює перетин масиву в вигляді нового масиву
	var arr = [1, 2, 3, 4, 5]
	arr.slice(2) // => [3, 4, 5]
	arr.slice(1, 4) // => [2, 3, 4]

splice	Додає та/або знищує елементи масиву
	arr = ["a", "b", "c", "d", "e"];
	removed = arr.splice(1,2);

sort	Сортує елементи масиву arr = [1,-1, 0]; a = arr.sort();

unsift	Додає один або більше елементів в початок масиву та повертає нову довжину масиву
	var arr = ["a", "b"];
	unshifted = arr.unshift(-2, -1);

Об’єкт *Date* використовується для роботи з датами. Синтаксис оператора створення екземпляра об’єкту дати:

ім’я_об’єкту_дати = new Date([параметри])

Якщо параметри відсутні, то значенням об’єкту буде поточна дата. Параметром може бути рядок типу: "місяць день, рік часи:хвилини;секунди".

Наприклад, для створення дати – "5 лютого 2005 року 23:12:07" необхідно:

day = new Date("February 5, 2005 23:12:07")

Прочитати або змінити параметри створеного об’єкту Date можливо за

допомогою ряду методів.

Найбільш вживані методи показані в табл. 2.4

Таблиця 2.4 - Методи об'єкту Date

Метод	Призначення
getDate	Повертає число місяця для вказаної дати var sputnikLaunch = new Date("October 4, 1957 19:28:34 GMT")
getDay	Повертає день тижня для вказаної дати
getHours	Повертає годину для вказаної дати
getMinutes	Повертає хвилини для вказаної дати
getMonth	Повертає місяць для вказаної дати
getSeconds	Повертає секунди для поточного часу
getYear	Повертає рік для вказаної дати
setDate	Встановлює число місяця для вказаної дати theBigDay = new Date("July 27, 1962 23:30:00") theBigDay.setDate(24)
setDay	Встановлює день тижня для вказаної дати
setHours	Встановлює годину для вказаної дати
setMinutes	Встановлює хвилини для вказаної дати
setMonth	Встановлює місяць для вказаної дати
setSeconds	Встановлює секунди для вказаної дати
setYear	Встановлює рік для вказаної дати

Об'єкт *Math* дозволяє використовувати вбудовані в JavaScript математичні функції та константи. При зверненні до методів та властивостей цього об'єкту створювати його не потрібно, але необхідно явно вказувати його ім'я.

Наприклад для того, щоб записати в змінну a результат розрахунку функції sin від 1 радіану необхідно:

a = Math.sin(1)

Для того, щоб записати в змінну a результат виразу 5 в степені 6 необхідно:

a = Math. pow(5,6)

Методи об'єкту Math, що використовуються найбільш часто представлені в табл. 2.5.

Таблиця 2.5 Методи об'єкту Math

Метод	Призначення
abs	Повертає абсолютне значення змінної. Math.abs(-2);
sin, cos, tan	Повертають значення тригонометричних функцій. Аргументи задаються в радіанах.
acos, asin, atan	Повертають значення обернених тригонометричних функцій
exp, log	Повертають значення експоненціальної функції та

	функції натурального логарифму
ceil	Повертає найменше ціле число, більше або рівне значенню аргументу
floor	Повертає найменше ціле число, менше або рівне значенню аргументу
min, max	Повертає найбільше/ найменше значення з двох аргументів
pow	Повертає значення функції: pow(x,y)=x^y
round	Повертає значення аргументу, округлене до найближчого цілого числа
sqrt	Повертає квадратний корінь аргументу

Об'єкт *String* використовується для роботи з рядковими типами даних. Створення об'єкту String відбувається коли змінній присвоюється рядковий літерал:

a = "Не явний спосіб створення рядкового об'єкту"

Крім того, можливо явно створити рядковий об'єкт, використовуючи оператор new та конструктор String:

ім'я_об'єкту = new String(Рядок)

Параметром конструктору може бути будь-який рядок. Наприклад:

a=new String("Явний спосіб створення рядкового об'єкту")

Єдиною властивістю об'єкту String є *length*, що зберігає довжину рядка. Наприклад для запису в змінну h довжини рядка a необхідно:

h=a.length

Методи об'єкту String, що використовуються найбільш часто перераховані в табл. 2.6. Наведемо приклад використання методу toLowerCase для переводу рядкової змінної a в верхній регістр:

a=a.toLowerCase();

Таблиця 2.6 Методи об'єкту String

Метод	Призначення
anchor	Створює HTML якір, який використовується, як гіпертекстове посилання.
link	Створює гіпертекстове посилання, по якій можливо перейти на інший URL.
fontsize	Виводить рядок, з встановленим розміром шрифта
bold	Виводить рядок, що відображається напівжирним шрифтом
italics	Виводить рядок, що відображається курсивом
strike	Виводить рядок, що відображається перекресленим шрифтом
substring	Повертає частину рядка об'єкта string
sub	Виводить рядок, що відображається як нижній індекс
sup	Виводить рядок, що відображається як верхній індекс

toLowerCase, toUperCase	Переводить зміст рядка в нижній/верхній регістр

На додаток до стандартних об’єктів JavaScript існує декілька функцій, для виклику яких не потрібно створювати об’єктів. Ці функції дістали назву "функцій верхнього рівня". До цих функцій відносяться: *parseFloat(параметр)* та *parseInt(параметр)*. Також досить часто використовуються функції *Number(об’єкт)* та *String(об’єкт)*, які перетворюють об’єкт, що використовується в якості параметру в число або рядок.

Функція parseInt перетворить перший аргумент в число по вказаній основі, а якщо це неможливо - повертає NaN. Синтаксис parseInt:

```
var intValue = parseInt(string[, radix])
```

Аргументи: string - рядкове представлення числа;

radix - основа системи числення

Наприклад, radix=10 дасть десяткове число, 16 - шістнадцяткове і тому подібне. Для radix>10 цифр після дев'яти представлені буквами латинського алфавіту.

Якщо в процесі перетворення parseInt виявляє цифру, яка не є цифрою в системі числення з основою radix, наприклад G в 16-й системі або A в десятковій, то процес перетворення тут же завершується і повертається значення, отримане з рядка на даний момент.

Функція parseInt округлює дробові числа, і зупиняється на десятковій крапці.Якщо radix не вказаний або дорівнює 0, то javascript припускає наступне:

- Якщо вхідний рядок починається з "0x", то radix = 16
- Якщо вхідний рядок починається з "0", то radix = 8. Цей пункт залежить від реалізації і в деяких браузерах (Google Chrome) відсутній.
- У будь-якому іншому випадку radix=10

Функція parseFloat аналогічна:

```
var floatValue parseFloat(strVal);
```

Аргументи: strVal - рядок, що представляє числове значення

2.2.3 Використання об’єктів window, document

Об’єкт *window* створюється автоматично при запуску браузера. Крім того нове вікно можливо створити і засобами JavaScript. Для цього необхідно використати метод open. Синтаксис методу такий:

Ім’я_змінної = window.open([Ім’я_ файлу],[Ім’я_вікна],[Параметри])

Ім’я_змінної – це ім’я для звернення до нового вікна в програмі JavaScript.

Ім’я_ файлу – це повна або відносна URL-адреса документу, що буде відкриватись в вікні браузеру.

Ім’я_вікна – це ім’я, що буде вказане в якості цілі в гіпертекстовому

посиланні на це вікно із іншого HTML-документу.

Параметри – задають значення параметрів вікна. Основні параметри представлені в табл. 2.7. Якщо можливе значення властивості yes або no, то при стандартних настройках використовується значення yes.

Таблиця 2.7 - Основні параметри вікна

Назва	Призначення	Можливі значення
directories	Наявність/відсутність панелі "Ссылки"	yes/no
height	Висота вікна	кількість пікселів
location	Наявність/відсутність адресного рядка	yes/no
menubar	Наявність/відсутність рядка меню	yes/no
resizable	Можливість/не можливість зміни розмірів віна користувачем	yes/no
scrollbars	Наявність/відсутність смуг прокрутки віна	yes/no
status	Наявність/відсутність рядка стану браузера	yes/no
toolbar	Наявність/відсутність панелей інструментів	yes/no
width	Ширина вікна	кількість пікселів

Наприклад, для створення нового вікна браузера в якому буде завантажено файл a.html необхідно:

```
<script>
myw=window.open("a.html","displayWindow",
    "width=400,height=300,status=no,toolbar=no,
    menubar=no")   </script>
```

При цьому, ширина вікна дорівнює 400 пікселів, висота вікна 300 пікселів, рядок стану вікна, панель інструментів та рядок меню будуть відсутні. Відзначимо, що наведений код необхідно записати в одному рядку.

Для закриття вікна браузера використовується метод close. Наприклад для закриття вікна попереднього прикладу необхідно:

```
myw.close()
```

Відзначимо, що при звернення до методів та властивостей вікна браузера в якому знаходиться програма JavaScript імені вікна або ключового слова window можна не вказувати. Наприклад, закрити поточне вікна браузера можливо так:

```
close()
```

Цікавим методом об’єкту window є метод *setTimeout*, за допомогою якого можливо запрограмувати виконання деяких команд після закінчення встановленого терміну часу. Синтаксис методу такий:

```
setTimeout("Код_JavaScript",інтервал_часу)
```

В якості першого параметру функції setTimeout, як правило

використовують функцію. Відзначимо, що цю функцію необхідно визначити на HTML-сторінці до використання функції setTimeout. Другим параметром функції setTimeout є інтервал часу закінчення якого буде сигналом про початок виконання команд JavaScript. Цей параметр задається в мілісекундах. Наприклад, виконання функції myfunction через 3000 мілісекунд після завантаження HTML-сторінки можливо запрограмувати так:

```
setTimeout("myfunction()",3000)
```

Досить часто функція setTimeout використовується для створення анімаційних ефектів. Це може бути, наприклад, циклічна зміна кольору тексту, або циклічна заміна одного зображення іншим.

Об'єкт *document* містить інформацію про завантажену сторінку. Всі елементи HTML-сторінки є властивостями цього об'єкту. Найбільш використовуваним методом об'єкту document є метод *write*, за допомогою якого можливо зробити запис в HTML-сторінку. Наприклад, для запису рядка "Привіт JavaScript" в документ в якому знаходиться сценарій необхідно:

```
document.write("Привіт JavaScript")
```

Метод writeln виводить переданий текст на сторінку, відступаючи при цьому новий рядок, після кожного виводу.

Інші методи об'єкта window наведено в таблиці 2.8.

Таблиця 2.8 – Методи об'єкта Window

alert()	Викликає вікно сповіщення, яке містить текст повідомлення і клавішу ОК. `window.alert("Вітаю, це сповіщення");`
blur()	Робить вікно неактивним. `win.blur()`
clearInterval()	Припиняє повторне виконання коду заданого setInterval(). `clearInterval(id);`
clearTimeout()	Відміняє заплановане методом setTimeout() виконання коду. `clearTimeout(id)`
confirm()	Викликає вікно підтвердження що містить текст повідомлення і клавіші ОК і Відміна. Повертає true або false `var x = confirm('Натисніть ОК або Отмена.');`
focus()	Робить вікно активним. `window.focus()`

moveBy()	Зміщує вікно відносно його поточної позиції.
	`win = window.open();`
	`win.moveBy(200,100);`

moveTo()	Переміщає вікно на вказану позицію. `win.moveTo(500,400);`

	`win.focus();`
print()	Роздруковує вміст поточного вікна. `print();`
prompt()	Викликає вікно запиту, спонукаюче відвідувача ввести в нього певні дані. `<html> <head>` `<script type='text/javascript'>` `function message()` `{ var x = prompt('Введіть Ваше ім'я:','Ім'я');` `//Виведемо введені користувачем дані на сторінку` `document.getElementById('mes').innerHTML = x; }` `</script> </head>` `<body>` `<input type='button' value='Викликати вікно запиту' onclick='message()' />` `  ` `<p style='display:inline'> Ваше ім'я: </p>` `<b><div style='display:inline' id='mes'> Невідомо </div></b>` `</body> </html>`
scrollBy()	Прокручує вміст вікна на вказану кількість пікселів. `scrollBy(0,100);`
scrollTo()	Прокручує вміст вікна до вказаних координат. `scrollTo(0,960);`

Пошук елементів на сторінці

Стандарт DOM передбачає декілька засобів пошуку елементу. Це методи getElementById, getElementsByTagName і getElementsByName.

Потужніші способи пошуку пропонують javascript -библиотеки.

Пошук по id

Найзручніший спосіб знайти елемент в DOM - це отримати його по id. Для цього використовується виклик document.getElementById(id)

Наприклад, наступний код змінить колір тексту на блакитний в div 'і з id="dataKeeper" :

```
document.getElementById('dataKeeper').style.color = 'blue'
```

Пошук по тегу

Наступний спосіб - це отримати усі елементи з певним тегом, і серед них шукати потрібний. Для цього служить document.getElementsByTagName(tag). Вона повертає масив з елементів, що мають такий тег.

Наприклад, можна отримати другий елемент(нумерація в масиві йде з

нуля) з тегом li :

```
document.getElementsByTagName('LI')[1]
```

Що цікаво, getElementsByTagName можна викликати не лише для document, але і взагалі для будь-якого елементу, у якого є тег (не текстового).

При цьому будуть знайдені тільки ті об'єкти, які знаходяться під цим елементом.

Наприклад, наступний виклик отримує список елементів LI, що знаходяться усередині першого тега div :

```
document.getElementsByTagName('DIV')[0].getElements
      ByTagName('LI')
```

Отримати усіх нащадків

Виклик elem.getElementsByTagName('*') поверне список з усіх дітей вузла elem в порядку їх обходу.

Наприклад, на такому DOM:

```
<div id="d1">
      <ol id="ol1">
      <li id="li1">1</li>
      <li id="li2">2</li>
      </ol>
</div>
```

Такий код:

```
var div = document.getElementById('d1')
var elems = div.getElementsByTagName('*')
for(var i=0; i<elems.length; i++) alert(elems[i].id)
```

виведе послідовність: ol1, li1, li2.

Пошук по name.

Метод document.getElementsByName(name) повертає усі елементи, у яких ім'я (атрибут name) рівно даному.

Він працює тільки з тими елементами, для яких в специфікації явно передбачений атрибут name: це form, input, a, select, textarea і ряд інших, рідкісніших.

Метод document.getElementsByName не працюватиме з іншими елементами типу div, p і тому подібне.

Тема 2.3. Загальні відомості про мову PHP

2.3.1Вступ в PHP

Мова PHP була розроблена як інструмент для вирішення чисто практичних завдань. Його творець, Расмус Лердорф, хотів знати, скільки чоловік читають його online -резюме, і написав для цього простеньку CGI - оболонку на мові Perl, тобто це був набір Perl - скриптів, призначених виключно для певної мети - збору статистики відвідувань.

Для довідки. CGI (Common Gateway Interface - загальний інтерфейс шлюзів) є стандартом, який призначений для створення серверних застосувань, працюючих по протоколу HTTP. Такі застосування (їх називають шлюзами або CGI - програмами) запускаються сервером в режимі реального часу. Сервер передає запити користувача CGI - програмі, яка їх обробляє і повертає результат своєї роботи на екран користувача. Таким чином, відвідувач отримує динамічну інформацію, яка може змінюватися в результаті впливу різних чинників. Сам шлюз (скрипт CGI) може бути написаний на різних мовах програмування - Ci/C++, Fortran, Perl, TCL, UNIX Shell, Visual Basic, Python та ін.

Незабаром з'ясувалося, що оболонка має невелику продуктивність, і довелося переписати її наново, але вже на мові Ci. Після цього первинники були викладені на загальний огляд для виправлення помилок і доповнення. Користувачі сервера, де розташовувався сайт з першою версією PHP, зацікавилися інструментом, з'явилися охочі його використати. Так що скоро PHP перетворився на самостійний проект, і на початку 1995 року вийшла перша відома версія продукту, Personal Home Page Tools (засоби для персональної домашньої сторінки), що називалася.

До середини 1995 року після грунтовної переробки з'явилася друга версія продукту, названа PHP/FI (Personal Home Page / Forms Interpreter - персональна домашня сторінка/ інтерпретатор форм).

У 1997 вийшла друга версія Ci -реализація PHP - PHP/FI 2.0. До того моменту PHP використали вже декілька тисяч чоловік по всьому світу, приблизно з 50 тис. доменів, що складало близько 1% усього числа доменів Internet.

PHP 3.0 була першою версією, PHP, що нагадує, яким ми знаємо його сьогодні. Він дуже сильно відрізнявся від PHP/FI 2.0 і з'явився знову ж таки як інструмент для вирішення конкретного прикладного завдання. Його творці, Енді Гутманс (Andi Gutmans) і Зів Сураски (Zeev Suraski), в 1997 році переписали наново код PHP/FI, оскільки він здався ним непридатним для розробки додатка електронної комерції, над яким вони працювали. Однією з сильних сторін PHP 3.0 була можливість розширення ядра. Саме властивість розширюваності PHP 3.0 притягнула увагу безлічі розробників, бажаючих додати свій модуль розширення. Крім того, PHP 3.0 надавала широкі можливості для взаємодії з базами даних, різними протоколами і API.

До кінця 1998 року число користувачів PHP зросло до десятків тисяч.

Сотні тисяч web -сайтов повідомляли про те, що вони працюють з використанням цієї мови. Майже на 10% серверів Internet був встановлений PHP 3.0. Нове ядро було назване " Zend Engine " (від імен творців : Zeev і Andi) і уперше представлено в середині 1999 року. PHP 4.0, заснований на цьому ядрі і такий, що приніс з собою набір додаткових функцій, офіційно вийшов в травні 2000 року, майже через два роки після свого попередника, PHP 3.0.

Нині ведуться роботи по поліпшенню Zend Engine і впровадженню нововведень в PHP 5.0. Одна з істотних змін сталася в об'єктній моделі мови, її грунтовно підлатали і додали багато нових можливостей.

Шоста версія PHP розроблялася з жовтня 2006 року. Було зроблено безліч нововведень, як, наприклад, виключення з ядра регулярних виразів POSIX і "довгих" суперглобальних масивів, видалення директив safe_mode, magic_quotes_gpc і register_globals з конфігураційного файлу php.ini. Одним з основних нововведень повинна була стати підтримка Юнікода. Проте у березні 2010 року розробка PHP6 була визнана безперспективною із-за складнощів з підтримкою Юнікоду. Початковий код PHP6 переміщений на гілку, а основною лінією розробки стала версія 5.4.)

Сьогодні PHP використовується сотнями тисяч розробників. Декілька мільйонів сайтів написані на PHP, що складає більше 20% доменів Internet.

Можливості PHP

" PHP може усе", - заявляють його творці. В першу чергу PHP використовується для створення скриптів, працюючих на стороні сервера, для цього його, власне, і придумали. PHP здатний вирішувати ті ж завдання, що і будь-хто інші CGI - скрипти, у тому числі обробляти дані html -форм, динамічно генерувати html сторінки і тому подібне. Але є і інші області, де може використовуватися PHP . Всього виділяють три основні сфери застосування PHP.

Перша область, як вже говорилося, - це створення додатків (скриптів), які виконуються на стороні сервера. PHP найширше використовується саме для створення такого роду скриптів. Для того, щоб працювати таким чином, знадобиться PHP -парсер (тобто обробник php - скриптів) і web -сервер для обробки скрипта, браузер для перегляду результатів роботи скрипта, ну, і, звичайно, який-небудь текстовий редактор для написання самого php -кода.

Друга область - це створення скриптів, що виконуються в командному рядку. Тобто за допомогою PHP можна створювати такі скрипти, які виконуватимуться, незалежно від web -сервера і браузеру, на конкретній машині. Для такої роботи знадобиться лише парсер PHP (в цьому випадку його називають інтерпретатором командного рядка (cli, command line interpreter)).

І остання область - це створення GUI -додатків (графічних інтерфейсів), що виконуються на стороні клієнта. В принципі це не самий кращий спосіб використати PHP, особливо для початківців, але якщо ви вже досконально вивчили PHP, то такі можливості мови можуть виявитися дуже

корисні. Для застосування PHP в цій області знадобиться спеціальний інструмент - PHP - GTK, який є розширенням PHP.

Отже, сфера застосування PHP досить велика і різноманітна. Проте існує безліч інших мов програмування, здатних вирішувати схожі завдання. Чому варто вивчати PHP? Що це нам дає? По-перше, PHP дуже простий у вивченні. Досить ознайомитися лише з основними правилами синтаксису і принципами його роботи, і можна починати писати власні програми, причому братися за такі завдання, рішення яких на іншій мові вимагало б серйозної підготовки.

По-друге, PHP підтримується майже на усіх відомих платформах, майже в усіх операційних системах і на самих різних серверах. Це теж дуже важливо. Навряд чи комусь захочеться переходити, наприклад, від роботи під Windows до роботи під Linux або від сервера IIS до сервера Apache тільки для того, щоб вивчити ще одну мову програмування.

Перша PHP -программа

Розглянемо приклад.

```
<html>  <head>
  <title>Приклад</title>
  </head>  <body>
  <?php
  echo "<p>Привіт, я - скрипт PHP!</p>";
  ?>
</body> </html>
```

Це простий HTML -файл, в який вбудований за допомогою спеціальних тегів код, написаний на мові PHP.

PHP -скрипти - це програми, які виконуються і обробляються сервером. Так що порівнювати його із скриптовими мовами типу JavaScript неможливо, тому що написані на них скрипти виконуються на машині клієнта. У чому відмінність скриптів, що виконуються на сервері і на клієнтові? Якщо скрипт обробляється сервером, клієнтові посилаються тільки результати роботи скрипта . Наприклад, якщо на сервері виконувався скрипт, подібний до приведеного вище, клієнт отримає згенеровану HTML -сторіноку виду :

```
<html>
  <head>
  <title>Приклад</title>
  </head>
  <body>
  <p>Привіт, я - скрипт PHP!</p>
  </body>
</html>
```

В цьому випадку клієнт не знає, який код виконується. Можна навіть конфігурувати свій сервер так, щоб HTML -файли оброблялися процесором PHP, так що клієнти навіть не зможуть дізнатися, чи отримують вони

звичайний HTML-файл або результат виконання скрипта. Якщо ж скрипт обробляється клієнтом (наприклад, це програма на мові JavaScript), то клієнт отримує сторінку, що містить код скрипта.

Ми відмічали вище, що PHP - скрипти вбудовуються в HTML -код. Виникає питання, яким чином? Є декілька способів. Один з них приведений в найпершому прикладі - за допомогою відкриваючого тега <?php і закриваючого тега ?> . Такого виду спеціальні теги дозволяють перемикатися між режимами HTML і PHP. Цей синтаксис найбільш прийнятний, оскільки дозволяє задіяти PHP в XML-сумісних програмах (наприклад, написаних на мові XHTML), але проте можна використати наступні альтернативні варіанти (команда echo "Some text" ; виводить на екран текст "Some text".) :

```
<? echo "Це проста інструкція для обробки PHP"; ?>
<script language="php">
         echo "Текст на сорінці";
     </script>

<% echo "Можна використати теги у стилі ASP "; %>
```

Перший з цих способів не завжди доступний. Щоб їм користуватися, треба включити короткі теги або за допомогою функції short_tags () для PHP 3, або включивши установку short_open_tag в конфігураційному файлі PHP, або скомпілювавши PHP з параметром --enable - short - tags. Навіть якщо це включено за умовчанням в php.ini - dist, використання коротких тегів не рекомендується. Другий спосіб аналогічний вставці, наприклад, JavaScript - кода і використовує для цього відповідний html тег. Тому використати його можна завжди, але це робиться рідко із-за його громіздкості. Третій спосіб можна застосувати, тільки якщо теги в стилі ASP були включені, використовуючи конфігураційну установку asp_tags.

Коли PHP обробляє файл, він просто передає його текст, поки не зустріне один з перерахованих спеціальних тегів, який повідомляє його про необхідність почати інтерпретацію тексту як кода PHP. Потім він виконує увесь знайдений код до закриваючого тега, що говорить інтерпретатору, що далі знову йде просто текст. Цей механізм дозволяє впроваджувати PHP -код в HTML - усе за межами тегів PHP залишається незмінним, тоді як усередині інтерпретується як код. Помітимо також, що php -файл не схожий на CGI - скрипт. Php файл не має бути здійснимим або ще яким-небудь чином поміченим.

Для того, щоб відправити php -файл на обробку серверу, треба в рядку браузеру набрати шлях до цього файлу на сервері. Скрипти php повинні розташовуватися там, де дозволений доступ через www, наприклад там же, де лежить домашня сторіночка. Якщо php -файл лежить на локальній машині, то його можна обробити за допомогою інтерпретатора командного рядка.

2.3.2 Загальний синтаксис PHP

Перше, що треба знати відносно синтаксису PHP, - це те, як він вбудовується в HTML -код, як інтерпретатор дізнається, що це код на мові PHP. У попередній лекції ми вже говорили про це. Повторюватися не будемо, відмітимо тільки, що в прикладах ми найчастіше використовуватимемо варіант <?php ?>, і іноді скорочений варіант <? ?>.

Програма на PHP (та і на будь-якій іншій мові програмування) - це набір команд (інструкцій). Обробникові програми (парсеру) необхідно якось відрізняти одну команду від іншої. Для цього використовуються спеціальні символи - роздільники. У PHP інструкції розділяються так само, як і в Си або Perl, - кожен вираз закінчується крапкою з комою.

Закриваючий тег "?> " також має на увазі кінець інструкції, тому перед ним крапку з комою не ставлять. Наприклад, наступні фрагменти коду еквівалентні:

```
<?php
echo "Hello, world"!; // крапка з комою
                      // у кінці команди
                      // обов'язкова
?>
<?php
echo "Hello, world"! ?>
<!-- крапка з комою
    опускається із-за "?>" -->
```

Часто при написанні програм виникає необхідність робити які-небудь коментарі до коду, які ніяк не впливають на сам код, а тільки пояснюють його. Це важливо при створенні великих програм і у разі, якщо декілька чоловік працюють над однією програмою. За наявності коментарів в програмі в її коді розібратися набагато простіше. Крім того, якщо вирішувати задачу по частинах, недороблені частини рішення також зручно коментувати, щоб не забути про них надалі. У усіх мовах програмування передбачена можливість включати коментарі в код програми. PHP підтримує декілька видів коментарів : в стилі Си, C++ і оболонки Unix. Символи // і # означають початок однорядкових коментарів, /* і */ - відповідно почало і кінець багаторядкових коментарів.

```
<?php
echo "Мене звуть Вася";
  // Це однорядковий коментар
  // у стилі C++
echo "Моє прізвище Петренко";
/* Це багаторядковий коментар.
При виконанні програми усе, що
знаходиться тут (закоментовано)
буде ігноровано. */
echo "Я вивчаю PHP ";
```

```
    # Це коментар в стилі
    # оболонки Unix
?>
```

Важливим елементом кожної мови є змінні, константи і оператори, що застосовуються до цих змінних і констант. Розглянемо, як виділяються і обробляються ці елементи в PHP.

Змінна в PHP позначається знаком долара, за яким йде її ім'я. Наприклад:

$my_var

Ім'я змінної чутливе до регістра, тобто змінні $my_var і $My_var різні.

Імена змінних відповідають тим же правилам, що і інші найменування в PHP: правильне ім'я змінної повинне розпочинатися з букви або символу підкреслення з подальшими у будь-якій кількості буквами, цифрами або символами підкреслення.

У PHP 3 змінні завжди присвоювалися за значенням. Тобто коли ви присвоюєте вираз зміній, усі значення оригінального виразу копіюються в цю змінну. Це означає, наприклад, що після присвоєння однієї змінної значення інший, зміна однієї з них не впливає на значення іншої.

```
<?php
$first = ' Text '; // Присвоюваний $first  значення ' Text '
$second = $first; // Присвоюваний $second  значення $first
?>
```

PHP 4, окрім цього, пропонує ще один спосіб присвоєння значень змінним: присвоєння по посиланню. Для того, щоб присвоїти значення змінної по посиланню, це значення повинне мати ім'я, тобто воно має бути представлене якою-небудь змінною. Щоб вказати, що значення однієї змінної присвоюється іншій змінній по посиланню, треба перед ім'ям першої змінної поставити знак амперсанд &.

Розглянемо той же приклад, що і вище, тільки присвоюватимемо значення змінної first змінної second по посиланню:

```
<?php
$first = 'Text'; //Присвоюємо $first значення 'Text'
$second = &$first;
/* Робимо посилання на $first через $second.  Тепер значення
цих змінних завжди співпадатимуть */
?>
```

Для зберігання постійних величин, тобто таких величин, значення яких не міняється в ході виконання скрипта, використовуються константи. Такими величинами можуть бути математичні константи, паролі, шляхи до файлів і тому подібне. Основна відмінність константи від змінної полягає в тому, що їй не можна присвоїти значення більше одного разу і її значення не можна анулювати після її оголошення. Крім того, у константи немає приставки у вигляді знаку долара і її не можна визначити простим наданням значення. Як

же тоді можна визначити константу? Для цього існує спеціальна функція define(). Її синтаксис такий:

```
define("Ім'я_константи", "Значення_константи",
       [Нечутливість_до_регістра])
```

За умовчанням імена констант чутливі до регістра. Для кожної константи це можна змінити, вказавши в якості значення аргументу Нечутливість_до_регістра значення True. Існує угода, по якій імена констант завжди пишуться у верхньому регістрі.

Отримати значення константи можна, вказавши її ім'я. На відміну від змінних, не треба упереджати ім'я константи символом $. Крім того, для набуття значення константи можна використати функцію constant() з ім'ям константи в якості параметра.

```
<?php
// визначаємо константу  PASSWORD
define("PASSWORD", "qwerty");
//визначаємо регістронезалежну
//константу PI зі значенням 3.14
define("PI", "3.14", True);
// виведемо значення константи PASSWORD,
// тобто qwerty
echo (PASSWORD);
// теж виведе qwerty
echo constant("PASSWORD");
?>
```

Окрім констант, що оголошуються користувачем, про яких ми тільки що розповіли, в PHP існує ряд констант, визначуваних самим інтерпретатором. Наприклад, константа __FILE__ зберігає ім'я файлу програми (і шлях до нього), яка виконується в даний момент, __FUNCTION__ містить ім'я функції, __CLASS__ - ім'я класу, PHP_VERSION - версія інтерпретатора PHP. Повний список зумовлених констант можна отримати, прочитавши керівництво по PHP.

Оператори

Оператори дозволяють виконувати різні дії зі змінними, константами і виразами. Ми ще не згадували про те, що таке вираз. Вираз можна визначити як усе, що завгодно, що має значення. Змінні і константи - це основні і найбільш прості форми виразів. Існує безліч операцій (і операторів, що відповідають їм), які можна робити з виразами. В таблицях 2.9-2.14 детально розглянуто основні види операцій в PHP.

Таблиця 2.9. Арифметичні оператори

Позначення	Назва	Приклад
+	Додавання	$a + $b
-	Віднімання	$a - $b
*	Множення	$a * $b

/	Ділення	$a / $b
%	Залишок від ділення	$a % $b

Таблиця 2.10. Оператори роботи з рядками

Позначення	Назва	Приклад
.	Конкатенація (додавання рядків)	$c = $a . $b (це рядок, що складається з $a і $b)

Таблиця 2.11. Оператори присвоєння

Позначення	Назва	Опис	Приклад
=	Присвоєння	Змінній зліва від оператора буде присвоєно значення, отримане в результаті виконання яких-небудь операцій або змінної / константи з правого боку	$a = ($b = 4) +5; ($a дорівнюватиме 9, $b дорівнюватиме 4)
+=		Скорочення. Додає до змінної число і потім привласнює їй отримане значення	$a += 5; (еквівалентно $a = $a + 5;)
.=		Скорочено означає комбінацію операцій конкатенації і присвоєння (спочатку додається рядок, потім отриманий рядок записується в змінну)	$b = "Привіт "; $b .= "усім"; (еквівалентно $b = $b . "усім";) В результаті: $b="Привіт усім"

Таблиця 2.12. Логічні оператори

Позначення	Назва	Опис	Приклад
and	І	$a і $b істинні (True)	$a and $b
&&	І		$a && $b
or	АБО	Хоч би одна зі змінних $a або $b істинна (можливо, що і обоє)	$a or $b
||	АБО		$a || $b
xor	Виключне АБО	Одна зі змінних істинна. Випадок, коли вони обоє істинні, виключається	$a xor $b
!	Інверсія (NOT)	Якщо $a=True, то !$a=False і навпаки	! $a

Таблиця 2.13. Оператори порівняння

Позначення	Назва	Опис	Приклад
==	Рівність	Значення змінних рівні	$a == $b
===	Еквівалентність	Рівні значення і типи змінних	$a === $b
!=	Нерівність	Значення змінних не рівні	$a != $b
<>	Нерівність		$a <> $b

!==	Нееквівалентність	Змінні не еквівалентні	$a !== $b
<	Менше		$a < $b
>	Більше		$a > $b
<=	Менше або рівно		$a <= $b
>=	Більше або рівно		$a >= $b

Таблиця 2.14. Оператори інкременту і декременту

Позначення	**Назва**	**Опис**	**Приклад**
++$a	Пре-інкремент	Збільшує $a на одиницю і повертає $a	<? $a=4; echo "Має бути 4:" .$a++; echo "Має бути 6:" .++$a; ?>
$a++	Пост-інкремент	Повертає $a, потім збільшує $a на одиницю	
--$a	Пре-декремент	Зменшує $a на одиницю і повертає $a	
$a--	Пост-декремент	Повертає $a, потім зменшує $a на одиницю	

Типи даних

PHP підтримує вісім простих типів даних.

Чотири скалярні типи:

- boolean(логічний);
- integer(цілий);
- float(з плаваючою крапкою);
- string(строковий).

Два змішані типи:

- array(масив);
- object(об' єкт).

І два спеціальні типи:

- resource(ресурс);
- NULL.

У PHP не прийнято явне оголошення типів змінних. Прийнятніше, щоб це робив сам інтерпретатор під час виконання програми залежно від контексту, в якому використовується змінна. Розглянемо по порядку усі перераховані типи даних.

Тип boolean (булевий або логічний тип)

Цей простий тип виражає істинність значення, тобто змінна цього типу може мати тільки два значення - істина TRUE або брехня FALSE .

Щоб визначити булевий тип, використовують ключове слово TRUE або FALSE. Обидві регістонезалежні.

```
<?php
$test = True;
?>
```

Логічні змінні використовуються в різних керівниках конструкціях (циклах, умовах і тому подібне, детальніше мова про них піде в одній з наступних лекцій). Мати логічний тип, тобто набувати тільки два значення, істину або брехню, можуть також і деякі оператори (наприклад, оператор рівності). Вони також використовуються в конструкціях, що управляють, для перевірки яких-небудь умов. Наприклад, в умовній конструкції перевіряється істинність значення оператора або змінної і залежно від результату перевірки виконуються ті або інші дії. Тут умова може бути істинна або неправдива, що якраз і відбиває змінна і оператор логічного типу.

```
<?php
// Оператор '==' перевіряє рівність
// і повертає булеве значення
if ($know == False){ // якщо $know має
                     //значення false
echo "Вивчай PHP"!;
}
?>
```

Тип integer (цілі) задає число з множини цілих чисел Z ={..., - 2, - 1, 0, 1, 2, ...}. Цілі можуть бути вказані в десятковій, шістнадцятиричній або вісімковій системі числення, за бажанням з попереднім знаком " - " або " + ".

Якщо ви використовуєте вісімкову систему числення, ви повинні упередити число 0 (нулем), для використання шістнадцятиричної системи треба поставити перед числом 0x.

```
<?php
# десяткове число
$a = 1234;
# від'ємне число
$a = - 123;
# вісімкове число (екв. 83 у десятковій системі)
$a = 0123;
# шістнадцяткове число (екв. 26 у десятк.системі)
$a = 0x1A;
?>
```

Розмір цілого залежить від платформи, хоча, як правило, максимальне значення близько двох мільярдів (це 32-бітове знакове). Беззнакові цілі PHP не підтримує.

Якщо ви визначите число, що перевищує межі цілого типу, воно буде інтерпретовано як число з плаваючою крапкою. Також якщо ви використовуєте оператор, результатом роботи якого буде число, що перевищує межі цілого, замість нього буде повернено число з плаваючою крапкою.

У PHP не існує оператора ділення цілих. Результатом 1/2 буде число з

плаваючою крапкою 0.5. Ви можете привести значення до цілого, що завжди округлює його в меншу сторону, або використати функцію round(), що округлює значення за стандартними правилами. Для перетворення змінної до конкретного типу треба перед змінною вказати в дужках потрібний тип. Наприклад, для перетворення змінної $a=0.5 до цілого типу необхідно написати(0.5) або (integer) $a або використати скорочений запис(0.5). Можливість явного приведення типів за таким принципом існує для усіх типів даних (звичайно, не завжди значення одного типу можна перевести в інший тип). Ми не поглиблюватимемося в усі тонкощі приведення типів, оскільки PHP робить це автоматично залежно від контексту.

Тип float (числа з плаваючою крапкою) (вони ж числа подвійної точності або дійсні числа) можуть бути визначені за допомогою будь-якого з наступних синтаксисів :

```
<?php
$a = 1.234;
$b = 1.2e3;
$c = 7E-10;
?>
```

Розмір числа з плаваючою крапкою залежить від платформи, хоча максимум, як правило, ~1.8e308 з точністю близько 14 десяткових цифр.

Тип string (рядки)

Рядок - це набір символів. У PHP символ - це те ж саме, що байт, це означає, що існує рівно 256 різних символів. Це також означає, що PHP не має вбудованої підтримки Unicode. У PHP практично не існує обмежень на розмір рядків, тому немає абсолютно ніяких причин турбуватися про їх довжину.

Рядок в PHP може бути визначений трьома різними способами:

- за допомогою одинарних лапок ;
- за допомогою подвійних лапок ;
- heredoc - синтаксисом.

Одинарні лапки

Простий спосіб визначити рядок - це взяти її в одинарних лапок " ' ". Щоб використати одинарну лапку усередині рядка, як і у багатьох інших мовах, перед нею необхідно поставити символ оберненої косої риски " \ ", т. е. екранувати її. Якщо обернена коса риска повинна йти перед одинарною лапкою або бути у кінці рядка, необхідно продублювати її " \\' ".

Якщо усередині рядка, поміщеного в одинарні лапки, зворотний слеш " \ " зустрічається перед будь-яким іншим символом (відмінним від " \ " і " ' "), то він розглядається як звичайний символ і виводиться, як і усі інші. Тому обернену косу риску необхідно екранувати, тільки якщо вона знаходиться у кінці рядка, перед закриваючою лапкою.

У PHP існує ряд комбінацій символів, що розпочинаються з символу оберненої косої риски. Їх називають послідовностями, що управляють, і вони мають спеціальні значення, про які ми розповімо трохи пізніше. Так от, на відміну від двох інших синтаксисів, змінні і такі, що управляють

послідовності для спеціальних символів, що зустрічаються в рядках, поміщених в одинарні лапки, не обробляються .

```
<?php
// Виведе: Щоб вивести ' потрібно
// перед нею поставити \
echo 'Щоб вивести \' потрібно перед
нею поставити \\';?>
```

Якщо рядок поміщений в подвійні лапки " " ", PHP розпізнає більшу кількість послідовностей, що управляють, для спеціальних символів. Деякі з них приведені в таблиці 2.15.

Таблиця 2.15. Керуючі послідовності

Послідовність	Значення
\n	Новий рядок (LF або 0x0A (10) в ASCII)
\r	Повернення каретки (CR або 0x0D (13) в ASCII)
\t	Горизонтальна табуляція (HT або 0x09 (9) в ASCII)
\\	Обернена коса риска
\$	Знак долара
\"	Подвійна лапка

Повторюємо, якщо ви захочете екранувати будь-який інший символ, обернена коса риска також буде надрукована!

Найважливішою властивістю рядків в подвійних лапках є обробка змінних.

```
‹?php
$str = "Іван";
// виведе: Привіт, Іван!
echo "Привіт, $str"!;
?›
```

В даному випадку замість підрядка $str підставляється значення змінної $str. Відбувається це за наступною схемою. Підрядок вважатиметься змінній, якщо вона утворює правильне ім'я змінної. Наприклад, в рядку Hello, $strs! змінною вважатиметься $strs, а не $str як в попередньому прикладі. Якщо все-таки вимагається розпізнати саме змінну $str в цьому рядку, то потрібно застосувати фігурні дужки.

```
‹?php
$str = 'Ivan';
// виведе: Hello, Ivans!
echo "Hello, {$str}s"!;
?›
```

У тому разі якщо вам знадобиться вивести знак долара (наприклад, для виведення імені змінної), можна використати одинарні лапки або екранувати його.

```
‹?php
```

```
$str = "Hello";
// виведе: Змінна має ім'я $str
echo 'Змінна має ім'я $str';
echo "‹br -";
// виїде: Змінна має ім'я $str
echo 'Змінна має ім'я \$str';
?›
```

У обох випадках змінна в рядках не визначається.

Heredoc

Інший спосіб визначення рядків - це використання heredoc -синтаксиса. В цьому випадку рядок повинен розпочинатися з символу <<<, після якого йде ідентифікатор. Закінчується рядок цим же ідентифікатором. Закриваючий ідентифікатор повинен починатися в першому стовпці рядка. Крім того, ідентифікатор повинен відповідати тим же правилам іменування, що і усі інші мітки в PHP: містити тільки буквено-цифрові символи і знак підкреслення і розпочинатися не з цифри або знаку підкреслення.

Heredoc -текст поводиться так само, як і рядок в подвійних лапках, при цьому їх не маючи. Це означає, що вам немає необхідності екранувати лапки в heredoc, але ви як і раніше можете використати перелічені вище послідовності, що управляють. Змінні усередині heredoc теж обробляються.

```
<?php
$str = <<<EOD
Приклад рядка, що охоплює декілька
рядків, з використанням heredoc - синтаксису
EOD;
// Тут ідентифікатор - EOD. Нижче
// ідентифікатор EOD
$name = 'Вася';
echo <<<EOD
Мене звуть "$name".  EOD;
// виведе: Мене звуть "Вася".
?>
```

Зауваження: Підтримка heredoc була додана в PHP 4.

Тип array (масив)

Масив в PHP є впорядкованою картою - типом, який перетворить значення в ключі. Цей тип оптимізований в декількох напрямах, тому ви можете використати його як власне масив, список (вектор), хеш-таблицю (що є реалізацією карти), стек, чергу і так далі. Оскільки ви можете мати в якості значення інший масив PHP, можна також легко емулювати дерева.

Визначити масив можна за допомогою конструкції array () або безпосередньо задаючи значення його елементам.

Визначення за допомогою array()

array (key => value
key1 => value1, ...)

Мовна конструкція array () приймає як параметри пари ключ => значення, розділені комами. Символ => встановлює відповідність між значенням і його ключем. Ключ може бути як цілим числом, так і рядком, а значення може бути будь-якого наявного в PHP типу. Числовий ключ масиву часто називають індексом. Індексування масиву в PHP розпочинається з нуля. Значення елементу масиву можна отримати, вказавши після імені масиву в квадратних дужках ключ шуканого елементу. Якщо ключ масиву є стандартним записом цілого числа, то він розглядається як число, інакше - як рядок. Тому запис $a["1"] рівносильна запису $a[1], так само як і a["-1"] рівносильно $a[- 1].

```
<?php
$books = array ("php" =>"PHP guide", 12 => true);
echo $books["php"];
//виведе "PHP guide"
echo $books[12];
//виведе 1
?>
```

Якщо для елементу ключ не заданий, то в якості ключа береться максимальний числовий ключ, збільшений на одиницю. Якщо вказати ключ, якому вже було присвоєно якесь значення, то воно буде перезаписано. Починаючи з PHP 4.3.0, якщо максимальний ключ – від’ємне число, то наступним ключем масиву буде нуль (0).

Якщо використати в якості ключа TRUE або FALSE, то його значення переводиться відповідно в одиницю і нуль типу integer. Якщо використати NULL, то замість ключа отримаємо порожній рядок. Можна використати і сам порожній рядок в якості ключа, при цьому її потрібно брати в лапки. Так що це не те ж саме, що використання порожніх квадратних дужок. Не можна використати в якості ключа масиви і об'єкти.

Створення масиву за допомогою синтаксису квадратних дужок

Створити масив можна, просто записуючи в нього значення. Як ми вже говорили, значення елементу масиву можна отримати за допомогою квадратних дужок, усередині яких треба вказати його ключ, наприклад, $book["php"]. Якщо вказати новий ключ і нове значення, наприклад, $book["new_key"]="new_value", то в масив додасться новий елемент. Якщо ми не вкажемо ключ, а тільки присвоїмо значення $book[]="new_value", то новий елемент масиву матиме числовий ключ, на одиницю більший максимального існуючого. Якщо масив, в який ми додаємо значення, ще не існує, то він буде створений.

```
<?
$books["key"]= value; // додали в масив $books
                      // значення value з ключем key
$books[] = value1; /* додали в масив значення
```

```
                                value1 з ключем 13,
                                оскільки максимальний ключ у
                                нас був 12 */
?>
```

Для того, щоб змінити конкретний елемент масиву, треба просто присвоїти йому з його ключем нове значення. Змінити ключ елементу не можна, можна тільки видалити елемент (пару ключ / значення) і додати нову. Щоб видалити елемент масиву, треба використати функцію unset ().

```
<?php
$books = array ("php" => "PHP users guide",
                 12 => true);
unset($books[12]); //Видаляє елемент з
                   // ключем 12 з масиву
unset ($books);    // видаляє масив повністю
?>
```

Помітимо, що, коли використовуються порожні квадратні дужки, максимальний числовий ключ шукається серед ключів, існуючих в масиві з моменту останнього переиндексирования. Переіндексувати масив можна за допомогою функції array_values ().

Тип object (об'єкти)

Об'єкти - тип даних, що прийшов з об'єктно-орієнтованого програмування (ТОП). Згідно з принципами ТОП, клас - це набір об'єктів, що мають певні властивості і методи роботи з ним, а об'єкт відповідно - екземпляр класу. Наприклад, програмісти - це клас людей, які пишуть програми, вивчають комп'ютерну літературу і, крім того, як усі люди, мають ім'я і прізвище. Тепер, якщо узяти одного конкретного програміста, Васю Іванова, то можна сказати, що він є об'єктом класу програмістів, має ті ж властивості, що і інші програмісти, теж має ім'я, пише програми і тому подібне

У PHP для доступу до методів об' єкту використовується оператор ->. Для ініціалізації об' єкту використовується вираз new, що створює в змінній екземпляр об' єкту.

```
<?php
class Person //створюємо клас людей
{    // метод, який навчає людину PHP
function know_php()
        {
          echo "Тепер я знаю PHP";
        }
}
$bob = new Person; // створюємо об' єкт
                   // класу людина
$bob -> know_php(); // навчаємо його PHP
?>
```

Тип resource (ресурси)

Ресурс - це спеціальна змінна, що містить посилання на зовнішній ресурс (наприклад, з'єднання з базою даних). Ресурси створюються і використовуються спеціальними функціями (наприклад, mysql_connect(), pdf_new() і тому подібне).

Тип Null

Спеціальне значення NULL говорить про те, що змінна не має значення.

Змінна вважається NULL, якщо:

- їй була присвоєна константа NULL ($var = NULL);
- їй ще не було присвоєно яке-небудь значення;
- вона була видалена за допомогою unset ().

Існує тільки одне значення типу NULL - регістронезалежне ключове слово NULL.

Задача. Потрібно скласти листа різним людям з приводу різних подій. Спробуємо використати для вирішення цього завдання вивчені засоби - змінні, оператори, константи, рядки і масиви. Залежно від одержувача змінюється подія і звернення, вказані в листі, тому природно винести ці величини в змінні. Більше того, оскільки подій і людей багато, зручно використати змінні типу масив. Підпис в листі залишається постійним завжди, тому логічно задати її як константу. Щоб не писати занадто довгі і громіздкі рядки, використовуваний оператор конкатенації.

```
<?
// нехай наш підпис буде константою
define("SIGN", "З повагою, Вася");
// задамо масиви людей і подій
$names = array("Іван Іванович", "Петро Петрович", "Семен Семенович");
$events = array("f" => "день відкритих дверей", "o" => "відкриття виставки"
   "p" => "бал випускників");
// складемо текст запрошення
$str = "Шанований (а), $names[0]".;
$str .= "<br> Запрошуємо Вас на ".  $events["f"];
$str .= "<br>" . SIGN;
echo $str; // виведемо текст на экран
?>
```

2.3.3 Основні операції в PHP

Оператор if - це один з найважливіших операторів багатьох мов, включаючи PHP. Він дозволяє виконувати фрагменти коду залежно від умови. Структуру оператора if можна представити таким чином:

```
if (вираз) блок_виконання
```

Тут вираз є будь-який правильний PHP-вираз (тобто усе, що має значення). В процесі обробки скрипта вираз перетвориться до логічного типу. Якщо в результаті перетворення значення виразу істинне (True), то виконується блок_виконання. Інакше блок_виконання ігнорується. Якщо блок_виконання містить декілька команд, то він має бути поміщений у фігурні дужки { }.

Правила перетворення виразу до логічного типу:

У FALSE перетворяться наступні значення:

- логічне False
- цілий нуль (0)
- дійсний нуль (0.0)
- порожній рядок і рядок "0"
- масив без елементів
- об'єкт без змінних (детально про об'єкти буде розказано в одній з наступних лекцій)
- спеціальний тип NULL

Усі інші значення перетворяться в TRUE.

```
<?
$names = array("Іван", "Петро", "Семен");
if ($names[0]=="Іван"){
   echo "Привіт, Ваня"!;
   $num = 1;
   $account = 2000;
}
if ($num) echo "Іван перший в списку"!;
$bax = 30;
if ($account > 100*$bax+3)
   echo "Цей рядок не з'явиться
   на екрані, оскільки умова не виконана";
?>
```

Ми розглянули тільки одну, основну частину оператора if . Існує декілька розширень цього оператора. Оператор else розширює if на випадок, якщо що перевіряється в if вираз є невірним, і дозволяє виконати які-небудь дії за таких умов.

Структуру оператора if, розширеного за допомогою оператора else, можна представити таким чином:

```
if (вираз) блок_виконання
else блок_виконання1
```

Цю конструкцію if...else можна інтерпретувати приблизно так: якщо виконана умова (тобто вираз=true), то виконуємо дії з блоку_виконання, інакше - дії з блоку_виконання1. Використати оператор else не обов'язково.

Подивимося, як можна змінити попередній приклад, враховуючи необхідність здійснення дій і у разі невиконання умови.

```
<?
$names = array("Іван", "Петро", "Семен");
if ($names[0]=="Іван"){
    echo "Привіт, Ваня"!;
    $num = 1;
    $account = 2000;
} else {
    echo "Привіт, $names[0].
    А ми чекали Ваню:(";
}
if ($num) echo "Іван перший в списку"!;
else echo "Іван НЕ перший в списку"?!;
$bax = 30;
if ($account > 100*$bax+3)
    echo "Цей рядок не з'явиться на екрані
    оскільки умова не виконана";
    else echo "Зате з'явиться цей рядок"!;
?>
```

Ще один спосіб розширення умовного оператора if - використання оператора elseif . elseif - це комбінація else і if . Як і else, він розширює if для виконання різних дій у тому випадку, якщо умова, що перевіряється в if, невірна. Але на відміну від else, альтернативні дії будуть виконані, тільки якщо elseif-умаова є вірна. Структуру оператора if, розширеного за допомогою операторів else і elseif, можна представити таким чином:

```
if (вираз) блок_виконання
elseif(вираз1) блок_виконання1
...
else блок_виконанняN
```

Операторів elseif може бути відразу декілька в одному if -блоці. Elseif - твердження буде виконане, тільки якщо попередня if-умова є False, усі передуючі elseif –умови являються False, а дана elseif -умова - True.

```
<?
$names = array("Іван", "Петро", "Семен");
if ($names[0]=="Іван"){
    echo "Привіт, Ваня"!;
}elseif ($names[0] == "Петро"){
    echo "Привіт, Петя"!;
}elseif ($names[0] == "Семен"){
    echo "Привіт, Сеня"!;
}else {
```

```
    echo "Привіт, $names[0]. А ти хто такий"?;
}
?>
```

Альтернативний синтаксис. PHP пропонує альтернативний синтаксис для деяких своїх структур, що управляють, а саме для if, while, for, foreach і switch . У кожному випадку відкриваючу дужку треба замінити на двокрапку (:), а що закриває - на endif;, endwhile; і так далі відповідно.

Наприклад, синтаксис оператора if можна записати таким чином:

if(вираз) : блок_виконання endif;

Сенс залишається тим же : якщо умова, записана в круглих дужках оператора if, виявилася істиною, виконуватиметься увесь код, від двокрапки " :" до команди endif;. Використання такого синтаксису корисне при вбудовуванні php в html -код.

Оператор switch

Ще одна конструкція, що дозволяє перевіряти умову і виконувати залежно від цього різні дії, - це switch . На російську мову назву цього оператора можна перевести як "перемикач". І сенс у нього саме такий. Залежно від того, яке значення має змінна, він перемикається між різними блоками дії. switch дуже схожий на оператор if...elseif...else або набір операторів if . Структуру switch можна записати таким чином:

```
switch (вираз або змінна){
case значення1:
    блок_дій1
break;
case значення2:
    блок_дій2
break;
...
default:
    блок_дій_по_замовчуванню
}
```

На відміну від if, тут значення вираз не приводиться до логічного типу, а просто порівнюється зі значеннями, перерахованими після ключових слів case (значення1, значення2 і так далі). Якщо значення виразу співпало з якимсь варіантом, то виконується відповідний блок_дій - від двокрапки після значення, що співпало, до кінця switch або до першого оператора break, якщо такий знайдеться. Якщо значення виразу не співпало ні з одним з варіантів, то виконуються дії по замовчуванню (блок_дій_по_замовчуванню), що знаходяться після ключового слова default. Вираз в switch обчислюється тільки один раз, а в операторі elseif - кожного разу, тому, якщо вираз досить складний, то switch працює швидше.

```
<?
$names = array("Іван", "Петро", "Семен");
switch($names[0]){
```

```
case "Іван":
    echo "Привіт, Ваня"!;
break;
case "Петро":
    echo "Привіт, Петя"!;
break;
case "Семен":
    echo "Привіт, Сеня"!;
break;
default:
    echo "Привіт, $names[0].
    А як Вас звуть"?;
}
?>
```

Якщо в даному прикладі опустити оператор break, наприклад, в case "Петро":, то, якщо змінна виявиться рівному рядку "Петро", після виводу на екран повідомлення "Привіт, Петя"! програма піде далі і виведе також повідомлення "Привіт, Сеня"! і тільки потім, зустрівши break, продовжить своє виконання за межами switch .

Для конструкції switch, як і для if, можливий альтернативний синтаксис, де відкриваюча switch фігурна дужка замінюється двокрапкою, а що закриває - endswitch; відповідно.

Цикли

У PHP існує декілька конструкцій, що дозволяють виконувати дії, що повторюються, залежно від умови. Це цикли while, do.while, foreach і for . Розглянемо їх детальніше.

Оператор while. Структура:

```
while (вираз){ блок_виконання }
```

або

```
while (вираз) : блок_виконання endwhile;
```

while - простий цикл. Він пропонує PHP виконувати команди блоку_виконання до тих пір, поки вираз обчислюється як True (тут, як і в if, відбувається приведення виразу до логічного типу). Значення виразу перевіряється кожного разу на початку циклу, так що, навіть якщо його значення змінилося в процесі виконання блоку_виконання, цикл не буде зупинений до кінця ітерації (тобто доки усі команди блоку_виконання не будуть виконані).

```
<?
//ця програма надрукує усі парні цифри
    $i = 1;
    while ($i < 10){
        if ($i % 2 == 0) print $i;
    // друкуємо цифру, якщо вона парна
```

```
        $i++;        // і збільшуємо $i на одиницю
    }
?>
```

Оператор do... while.

Цикли do.while дуже схожі на цикли while, з тією лише різницею, що істинність виразу перевіряється у кінці циклу, а не на початку. Завдяки цьому блок_виконання циклу do...while гарантовано виконується хоч би один раз.

Структура:

```
do {блок_виконання} while (вираз);
<?
// ця програма надрукує число 12, незважаючи на те
// що умова циклу не виконана
$i = 12;
    do{
        if ($i % 2 == 0) print $i;
    // якщо число парне, то друкуємо його
        $i++;
    // збільшуємо число на одиницю
    }while ($i<10)
?>
```

Оператор for.

Це найскладніші цикли в PHP. Вони нагадують відповідні цикли C.

Структура:

for (вираз1; вираз2; вираз3) {блок_виконання}

або

for (вираз1; вираз2; вираз3) : блок_виконання endfor;

Тут, як ми бачимо, умова складається відразу з трьох виразів. Перший вираз вираз1 обчислюється безумовно один раз на початку циклу. На початку кожної ітерації обчислюється вираз2. Якщо він являється True, то цикл триває і виконуються усі команди блоку_виконання. Якщо вираз2 обчислюється як False, то виконання циклу зупиняється. У кінці кожної ітерації (тобто після виконання усіх команд блоку_виконання) обчислюється вираз3.

Кожен з виразів 1, 2, 3 може бути порожнім. Якщо вираз2 є порожнім, то це означає, що цикл повинен виконуватися невизначений час (в цьому випадку PHP вважає цей вираз завжди істинним). Це не так безглуздо, як здається, адже цикл можна зупиняти, використовуючи оператор break .

Наприклад, усі парні цифри можна вивести з використанням циклу for таким чином:

```
<?php
for ($i=0; $i<10; $i++){
    if ($i % 2 == 0) print $i;
    // друкуємо парні числа
}
?>
```

Якщо опустити другий вираз (умова $i<10), то таке ж завдання можна вирішити, зупиняючи цикл оператором break .

```
<?php
for ($i=0; ; $i++){
    if ($i>=10) break;
      // якщо $i більше або рівне 10,
      // то припиняємо роботу циклу
    if ($i % 2 == 0) print $i;
      // якщо число парне,
      // то друкуємо його
}
?>
```

Можна опустити усі три вирази. В цьому випадку просто не буде задано початкове значення лічильника $i і воно не змінюватиметься кожного разу у кінці циклу. Усі ці дії можна записати у вигляді окремих команд або у блоці_виконання, або перед циклом:

```
<?php
$i=2; // задаємо початкове значення лічильника
for ( ; ; ){
    if ($i>=10) break;
      // якщо $i більше або рівне 10,
      // то припиняємо роботу циклу
    if ($i % 2 == 0) print $i;
      // якщо число парне,
      // то друкуємо його
$i++; // збільшуємо лічильник на одиницю
}
?>
```

У третьому виразі конструкції for можна записувати через кому відразу декілька простих команд. Наприклад, якщо ми хочемо просто вивести усі цифри, то програму можна записати зовсім просто:

```
<?php
for ($i=0; $i<10; print $i, $i++)
/* Якщо блок_виконання не містить команд
чи містить тільки одну команду,
фігурні дужки, в які він поміщений,
можна опускати*/
?>
```

Оператор foreach.

Ще одна корисна конструкція. Вона з'явилася тільки в PHP4 і призначена виключно для роботи з масивами.

Синтаксис:

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

```
{
 блок_виконання
 }
```

У першому випадку формується цикл по усіх елементах масиву, заданого змінній $array. На кожному кроці циклу значення поточного елементу масиву записується в змінну $value, і внутрішній лічильник масиву пересувається на одиницю (так що на наступному кроці буде видний наступний елемент масиву). Усередині блоку_виконання значення поточного елементу масиву може бути отримане за допомогою змінної $value. Виконання блоку_виконання відбувається стільки разів, скільки елементів в масиві $array.

Друга форма запису на додаток до переліченого вище на кожному кроці циклу записує ключ поточного елементу масиву в змінну $key, яку теж можна використати у блоці_виконання.

Коли foreach починає виконання, внутрішній покажчик масиву автоматично встановлюється на перший елемент.

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

Оператори передачі управління

Іноді вимагається негайно завершити роботу циклу або окремої його ітерації. Для цього використовують оператори break і continue .

Оператор break закінчує виконання поточного циклу, будь то for, foreach, while, do.while або switch . break може використовуватися з числовим аргументом, який говорить, роботу скількох керівників структур, що містять його, треба завершити.

```
<?php
$i=1;
while ($i){
    $n = rand(1,10);
    // генеруємо довільне число від 1 до 10
    echo "$i:$n ";
    // виводимо номер ітерації і згенероване число
    if ($n==5) break;
```

```
    /* Якщо було згенеровано число 5, то припиняємо
    роботу циклу. В цьому випадку усе, що
    знаходиться після цього рядка усередині циклу,
    не буде виконано */
    echo "Цикл працює <br>";
    $i++;
}
echo "<br>Число ітерацій циклу $i ";
?>
```

Результатом роботи цього скрипта буде приблизно наступне:

```
1:7 Цикл працює
2:2 Цикл працює
3:5
Число ітерацій циклу 3
```

Якщо після оператора break вказати число, то урветься саме така кількість тих, що містять цей оператор циклів. У наведеному вище прикладі це неактуально, оскільки вкладених циклів немає. Трохи змінимо наш скрипт:

```
<?php
$i=1;
while ($i){
  $n = rand(1,10);
  // генеруємо довільне число від 1 до 10
  switch ($n){
    case 5:
      echo "<font color=blue>
        Вихід з switch (n=$n)</font>";
    break 1;
    // припиняємо роботу switch
    // (першого break циклу, що містить)
    case 10:
      echo "<font color=red>
        Вихід з switch і
        while (n=$n)</font>";
    break 2;
    // припиняємо роботу switch і while
    // (двох break циклів, що містять)
    default:
      echo "switch працює (n=$n) ";
}
echo " while працює - крок $i <br>";
$i++;
}
echo "<br>Число ітерацій циклу $i ";
?>
```

Іноді треба не повністю припинити роботу циклу, а тільки почати його нову ітерацію. Оператор continue дозволяє пропустити подальші інструкції з блоку_виконання будь-якого циклу і продовжити виконання з нового круга. continue можна використати з числовим аргументом, який вказує, скільки його управляють конструкцій, що містять, повинні завершити роботу.

Замінимий в прикладі попереднього параграфа оператор break на continue . Крім того, обмежимо число кроків циклу трьома.

```
<?php
$i=1;
while ($i<=4){
    $n = rand(1,10);
    // генеруємо довільне число від 1 до 10
    echo "$i:$n ";
    // виводимо номер ітерації і згенероване число
    if ($n==5){
        echo "Нова ітерація <br>";
        continue;
        /* Якщо було згенеровано число 5,
        те починаємо нову ітерацію циклу,
        $i не збільшується */
    }
    echo "Цикл працює <br>";
    $i++;
}
--$i;
echo "<br>Число ітерацій циклу $i ";
?>
```

Результатом роботи цього скрипта буде

```
1:10
Цикл працює
2:5 Нова ітерація
2:1 Цикл працює
3:1 Цикл працює
4:1 Цикл працює
Число ітерацій циклу 4
```

Помітимо, що після виконання оператора continue робота циклу не закінчується. У прикладі лічильник циклу не міняється у разі отримання числа 5, оскільки він знаходиться після оператора continue . Фактично за допомогою continue ми намагаємося уникнути ситуації, коли буде згенеровано число 5. Тому можна було просто написати, замінивши оператор continue на перевірку істинності вираз :

```
<?php
$i=1;
while ($i<4){
```

```
    $n = rand(1,10);
    // генеруємо довільне число від 1 до 10
    if ($n!==5){
        echo "$i:$n <br>";
    // виводимо номер ітерації і згенероване число
    $i++;
    }
}
?>
```

У PHP існує одна особливість використання оператора continue - в конструкціях switch він працює так само, як і break . Якщо switch знаходиться усередині циклу і треба почати нову ітерацію циклу, слід використати continue 2.

Оператори включення

Оператор include дозволяє включати код, що міститься у вказаному файлі, і виконувати його стільки разів, скільки програма зустрічає цей оператор. Включення може робитися будь-яким з перерахованих способів :

```
include 'ім'я_файлу';
include $file_name;
include ("ім'я_файлу");
```

Приклад. Нехай у файлі params.inc у нас зберігається набір якихось параметрів і функцій. Кожного разу, коли нам треба буде використати ці параметри (функції), ми вставлятимемо в текст нашої основної програми команду include 'params.inc'.

Файл params.inc:

```
<?php
$user = "Вася";
$today = date("d.m.y");
/* функція date() повертає дату і час
 (тут - дату у форматі день.місяць.рік) */
?>
```

Файл include.php:

```
<?php
include ("params.inc");
/* змінні $user і $today задані у файлі params.inc.
Тут ми теж можемо ними користуватися
завдяки команді include("params.inc") */
echo "Привіт, $user!<br>";
    // виведе "Привіт, Вася"!
echo "Сьогодні $today";
    // виведе, наприклад, "Сьогодні 7.07.05"
?>
```

Помітимо, що використання оператора include еквівалентно простій вставці змістовної частини файлу params.inc в код програми include.php. Можливо, тоді можна було в params.inc записати простий текст без всяких тегів, що вказують на те, що це php -код? Не можна! Річ у тому, що у момент вставки файлу відбувається перемикання з режиму обробки PHP в режим HTML. Тому код усередині файлу, що включається, який треба обробити як PHP-скрипт, має бути поміщений у відповідні теги.

Пошук файлу для вставки відбувається за наступними правилами.

Спочатку ведеться пошук файлу в include_path відносно поточної робочої директорії.

Якщо файл не знайдений, то пошук здійснюється в include_path відносно директорії поточного скрипта.

Параметр include_path, визначуваний у файлі налаштувань PHP, задає імена директорій, в яких треба шукати файли, що включаються.

Наприклад, ваш include_path це . (тобто поточна директорія) поточна робоча директорія це /www/. В основний файл include.php ви включаєте файл my_dir/a.php, який у свою чергу включає b.php. Тоді парсер насамперед шукає файл b.php в директорії /www/, і якщо такого немає, то в директорії /www/my dir/.

Якщо файл включений за допомогою include, то код, що міститься в ньому, наслідує зону видимості змінних рядка, де з'явився include . Будь-які змінні викликаного файлу будуть доступні в файлі з цього рядка і далі. Відповідно, якщо include з'являється усередині функції викликаю чого файлу, то код, що міститься у файлі, що викликається, поводитиметься так, як ніби він був визначений усередині функції. Таким чином, він успадкує зону видимості цієї функції. Хоча ми і не знайомилися ще з поняттям функції, все ж приводимо тут ці відомості з розрахунку на інтуїтивне його розуміння.

Приклад. Нехай файл для вставки params.inc залишиться таким же, а include.php буде наступним:

```
<?php
function Footer(){ // оголош.функцію з ім'ям Footer
include ("params.inc");
/* включаємо файл params.inc.   Тепер його
змінними можна користуватися але тільки
усередині функції */
$str = "Сьогодні: $today <br>";
$str .= "<a
href='mailto:help@intuit.ru'>Сторінку створив
     $user</a>";
    echo "$str";
}
Footer();
    // викликаємо функцію Footer(). Отримаємо:
```

```
    //Сьогодні: 08.07.05
    //Сторінку створив Вася

echo "$user, $today";
    // виведе кому, оскільки
    // ці змінні видно тільки
    // усередині функції
?>
```

Окрім локальних файлів, за допомогою include можна включати і зовнішні файли, вказуючи їх url-адресу. Ця можливість контролюється директивою url_fopen_wrappers у файлі налаштувань PHP і за умовчанням, як правило, включена. Але у версіях PHP для Windows до PHP 4.3.0 ця можливість не підтримується зовсім, незалежно від url_fopen_wrappers.

include() - це спеціальна мовна конструкція, тому при використанні усередині умовних блоків її треба брати у фігурні дужки.

```
<?php
/*  Це невірний запис. Отримаємо помилку.
    Ми ж вставляємо не одну команду, а декілька,
    вони тільки записані у іншому файлі */
if ($condition) include("first.php");
else include("second.php");
// А ось так правильно.
if ($condition){ include("first.php"); }
else { include("second.php"); }
?>
```

При використанні include можливі два види помилок - помилка вставки (наприклад, не можна знайти вказаний файл, невірно написана сама команда вставки і тому подібне) або помилка виконання (якщо помилка міститься у файлі, що вставляється). У будь-якому випадку при помилці в команді include виконання скрипта не завершується.

Оператор **require** діє приблизно так само, як і #include в C++. Усе, що ми говорили про include, лише за деякими виключеннями, справедливо і для require . require також дозволяє включати в програму і виконувати який-небудь файл. Основна відмінність require і include полягає в тому, як вони реагують на виникнення помилки. Як вже говорилося, include видає попередження, і робота скрипта триває. Помилка в require викликає фатальну помилку роботи скрипта і припиняє його виконання.

Умовні оператори на require() не впливають. Хоча, якщо рядок, в якому з'являється цей оператор, не виконується, то жоден рядок коду з файлу, що вставляється, теж не виконується. Цикли також не впливають на require(). Хоча код, що міститься у файлі, що вставляється, є об'єктом циклу, але вставка сама по собі відбувається тільки одного разу.

2.4.1 Протокол HTTP і способи передавання даних на сервер

Internet побудований за багаторівневим принципом, від фізичного рівня, пов'язаного з фізичними аспектами передачі двійкової інформації, і до прикладного рівня, що забезпечує інтерфейс між користувачем і мережею.

HTTP (HyperText Transfer Protocol, протокол передачі гіпертексту) - це протокол прикладного рівня, розроблений для обміну гіпертекстовою інформацією в Internet.

HTTP надає набір методів для вказівки цілей запиту, що відправляється серверу. Ці методи засновані на дисципліні посилань, де для вказівки ресурсу, до якого має бути застосований цей метод, використовується універсальний ідентифікатор ресурсів (Universal Resource Identifier) у вигляді місцезнаходження ресурсу (Universal Resource Locator, URL) або у вигляді його універсального імені (Universal Resource Name, URN).

Повідомлення по мережі при використанні протоколу HTTP передаються у форматі, схожому з форматом поштового повідомлення Internet (RFC - 822) або з форматом повідомлень MIME (Multipurpose Internet Mail Exchange).

HTTP використовується для комунікацій між різними призначеними для користувача програмами і програмами-шлюзами, що надають доступ до існуючим Internet-протоколам, таким як SMTP (протокол електронної пошти), NNTP (протокол передачі новин), FTP (протокол передачі файлів), Gopher і WAIS. HTTP розроблений для того, щоб дозволяти таким шлюзам через проміжні програми- сервери (proxy) передавати дані без втрат.

Протокол реалізує принцип запит/відповідь. Просяча програма- клієнт ініціює взаємодію з програмою, що відповідає- сервером, і посилає запит, що містить :

- метод доступу;
- адреса URL;
- версію протоколу;
- сполучення (схоже за формою на MIME) з інформацією про тип передаваних даних, інформацією про клієнта, що послав запит, і, можливо, зі змістовною частиною (тілом) повідомлення.

Відповідь сервера містить:

- рядок стану, в який входить версія протоколу і код повернення (успіх або помилка);
- повідомлення (у формі, схожій на MIME), в яке входить інформація сервера, метаінформація (тобто інформація про зміст повідомлення) і тіло повідомлення.

У протоколі не вказується, хто повинен відкривати і закривати з'єднання між клієнтом і сервером. На практиці з'єднання, як правило, відкриває клієнт, а сервер після відправки відповіді ініціює його розрив.

Давайте розглянемо детальніше, в якій формі вирушають запити на сервер.

Форма запиту клієнта

Клієнт посилає серверу запит в одній з двох форм : в повній або скороченій. Запит в першій формі називається відповідно до повним запитом, а в другій формі - простим запитом.

Простий запит містить метод доступу і адресу ресурсу. Формально це можна записати так:

```
<Простій-запит> :=<Метод> <символ пропуск>
    <Запитаний-URL> <символ нового рядка>
```

В якості методу можуть бути вказані GET, POST, HEAD, PUT, DELETE і інші. Про найбільш поширених з них ми поговоримо трохи пізніше. Як прошеного URL найчастіше використовується URL -адрес ресурсу.

Приклад простого запиту :

```
GET http://phpbook.info/
```

Тут GET - це метод доступу, тобто метод, який має бути застосований до прошеного ресурсу, а http://phpbook.info/ - це URL -адрес прошеного ресурсу.

Повний запит містить рядок стану, декілька заголовків (заголовок запиту, загальний заголовок або заголовок змісту) і, можливо, тіло запиту. Формально загальний вигляд повного запиту можна записати так:

```
<Повний запит> :=<Рядок Стану>
    (<Загальний заголовок>|<Заголовок запиту>|
    <Заголовок змісту>)
    <символ нового рядка>
    [<зміст запиту>]
```

Квадратні дужки тут означають необов'язкові елементи заголовка, через вертикальну рису перераховані альтернативні варіанти. Елемент <Рядок стану> містить метод запиту і URL ресурсу (як і простий запит) і, крім того, використовувану версію протоколу HTTP. Наприклад, для виклику зовнішньої програми можна задіяти наступний рядок стану :

```
POST http://phpbook.info/cgi-bin/test HTTP/1.0
```

В даному випадку використовується метод POST і протокол HTTP версії 1.0.

У обох формах запиту важливе місце займає URL прошеного ресурсу. Частіше усього URL використовується у вигляді URL -адреса ресурсу. При зверненні до сервера можна застосовувати як повну форму URL, так і спрощену.

Повна форма утримує тип протоколу доступу, адресу сервера ресурсу і адресу ресурсу на сервері (рисунок 2.2).

У скороченій формі опускають протокол і адресу сервера, вказуючи тільки місце розташування ресурсу від кореня сервера. Повну форму використовують, якщо можлива пересилка запиту іншому серверу. Якщо ж робота відбувається тільки з одним сервером, то частіше застосовують скорочену форму.

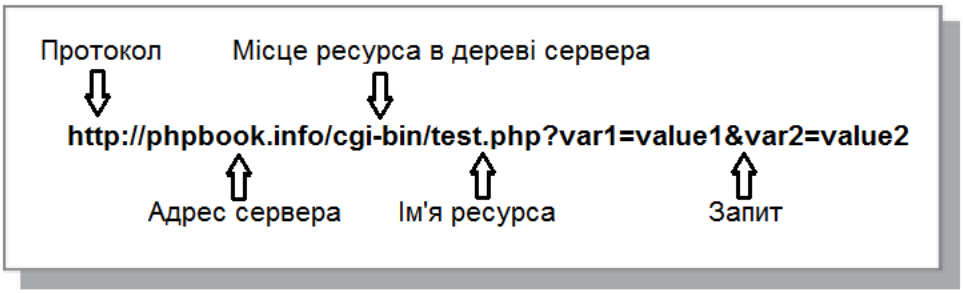


Рисунок 2.2. Повна форма URL

Далі ми розглянемо найбільш поширені методи відправки запитів.

Методи

Як вже говорилося, будь-який запит клієнта до сервера повинен розпочинатися з вказівки методу. Метод повідомляє про мету запиту клієнта. Протокол HTTP підтримує досить багато методів, але реально використовуються тільки три: POST, GET і HEAD . Метод GET дозволяє отримати будь-які дані, ідентифіковані за допомогою URL в запиті ресурсу. Якщо URL вказує на програму, то повертається результат роботи програми, а не її текст (якщо, звичайно, текст не є результат її роботи). Додаткова інформація, необхідна для обробки запиту, вбудовується в сам запит (у адресний рядок). При використанні методу GET в поле тіла ресурсу повертається інформація (текст HTML -документа, наприклад), що власне зажадалася.

Існує різновид методу GET - умовний GET . Цей метод повідомляє сервер про те, що на запит треба відповісти, тільки якщо виконана умова, що міститься в полі if, - Modified - Since заголовка запиту. Якщо говорити точніше, то тіло ресурсу передається у відповідь на запит, якщо цей ресурс змінювався після дати, вказаної в if, - Modified - Since.

Метод HEAD аналогічний методу GET, тільки не повертає тіло ресурсу і не має умовного аналога. Метод HEAD використовують для отримання інформації про ресурс. Це може згодитися, наприклад, при рішенні задачі тестування гіпертекстових посилань.

Метод POST розроблений для передачі на сервер такої інформації, як анотації ресурсів, новинні і поштові повідомлення, дані для додавання у базу даних, тобто для передачі інформації великого об'єму і досить важливої. На відміну від методів GET і HEAD, в POST передається тіло ресурсу, яке і є інформацією, що отримується з полів форм або інших джерел введення.

Досі ми тільки теоретизували, знайомилися з основними поняттями. Тепер пора навчитися використати усе це на практиці. Далі в лекції ми розглянемо, як посилати запити серверу і як обробляти його відповіді.

Використання HTML -форм для передачі даних на сервер

Як передавати дані серверу? Для цього в мові HTML є спеціальна

конструкція - форми. Форми призначені для того, щоб отримувати від користувача інформацію. Наприклад, вам треба знати логін і пароль користувача для того, щоб визначити, на які сторінки сайту його можна допускати. Чи вам потрібні особисті дані користувача, щоб була можливість з ним зв'язатися. Форми якраз і застосовуються для введення такої інформації. У них можна вводити текст або вибирати відповідні варіанти зі списку. Дані, записані у форму, вирушають для обробки спеціальній програмі (наприклад, скрипту на PHP) на сервері. Залежно від введених користувачем даних ця програма може формувати різні web -сторінок, відправляти запити до бази даних, запускати різні застосування і тому подібне

Розберемося з синтаксисом HTML -форм. Можливо, багато хто з ним знайомий, але ми все ж повторимо основні моменти, оскільки це важливо.

Отже, для створення форми в мові HTML використовується тег FORM . Усередині нього знаходиться одна або декілька команд INPUT . За допомогою атрибутів action і method тега FORM задаються ім'я програми, яка оброблятиме ці форми, і метод запиту, відповідно. Команда INPUT визначає тип і різні характеристики прошеної інформації. Відправка даних форми відбувається після натиснення кнопки input типу submit . Створимо форму для реєстрації учасників заочної школи програмування.

```
<h2>Форма для реєстрації учасників</h2>
<form action="1.php" method=POST> <!--створюємо форму-->
<!--ці форми оброблятиме файл 1.php, при
    відправці запиту буде використаний метод POST-->
Ім'я <br><input type=text name="first_name"
    value="Введіть Ваше ім'я"><br>
Прізвище <br><input type=text name="last_name"><br>
E - mail <br><input type=text name="email"><br>
<p>
Виберіть курс, який ви б хотіли відвідувати :<br>
<input type=radio name="kurs" value="PHP">PHP<br>
<input type=radio name="kurs" value="Lisp">Lisp<br>
<input type=radio name="kurs" value="Perl">Perl<br>
<input type=radio name="kurs" value="Unix">Unix<br></p>
<p>Що ви хочете, щоб ми знали про вас? <BR>
<textarea name="comment" cols=32 rows=5></textarea></p>
<p><input name="confirm" type=checkbox
    checked>Підтвердити отримання <br>
<input type=submit value="Відправити">
<input type=reset value="Відмінити"></p>
</form>
```

Після обробки браузером цей файл виглядатиме приблизно так, як показано на рисунку 2.3

Ось так створюються і виглядають HTML -формы. Вважатимемо, що ми навчилися або згадали, як їх створювати. Як ми бачимо, у формі можна вказувати метод передачі даних. Подивимося, що відбуватиметься, якщо вказати метод GET або POST, і в чому буде різниця.

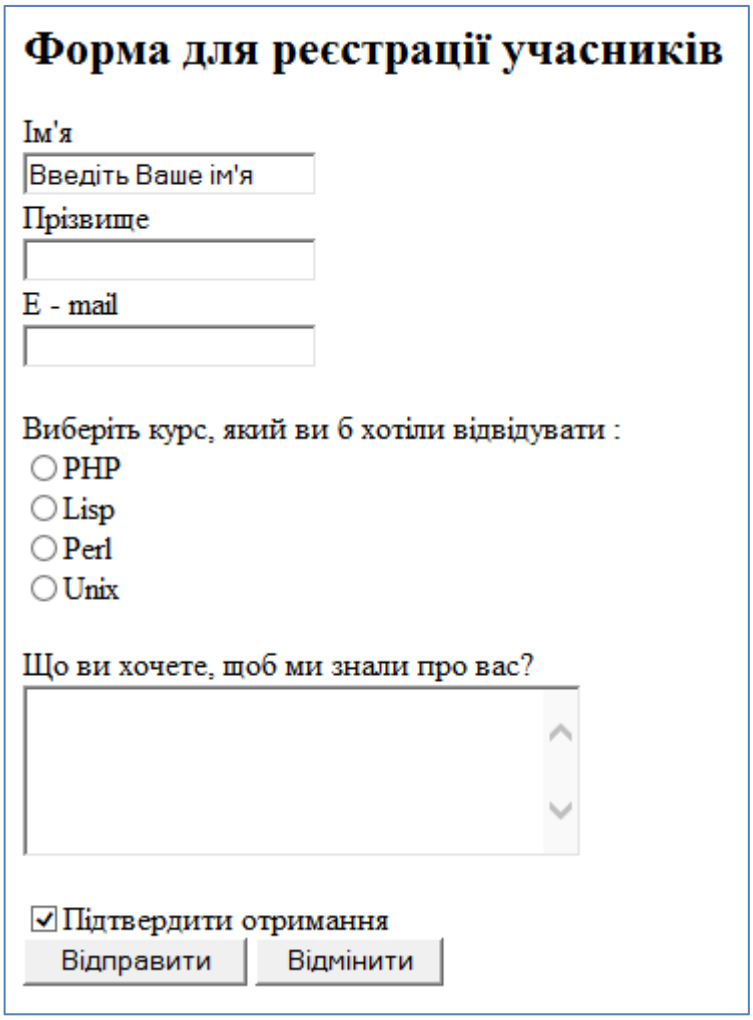


Рисунок 2.3. Приклад html -форми

Для методу GET

При відправці даних форми за допомогою методу GET вміст форми додається до URL після знаку питання у вигляді пар ім'я=значення, об'єднаних за допомогою амперсанда &:

```
action?name1=value1&name2=value2&name3=value3
```

Тут action - це URL -адрес програми, яка повинна обробляти форму (це або програма, задана в атрибуті action тега form, або сама поточна програма, якщо цей атрибут опущений). Імена name1, name2, name3 відповідають іменам елементів форми, а value1, value2, value3 - значенням цих елементів. Усі спеціальні символи, включаючи = і &, в іменах або значеннях цих параметрів будуть закодовані. Тому не варто використати в назвах або значеннях елементів форми ці символи і символи кирилиці в ідентифікаторах.

Якщо в полі для введення ввести який-небудь службовий символ, то він буде переданий в його шістнадцятиричному коді, наприклад, символ $ заміниться на %24. Так само передаються і російські букви.

Для полів введення тексту і пароля (це елементи input з атрибутом type=text і type=password), значенням буде те, що введе користувач. Якщо користувач нічого не вводить в таке поле, то в рядку запиту буде присутнім

елемент name=, де name відповідає імені цього елементу форми.

Для кнопок типу checkbox і radio button значення value визначається атрибутом VALUE у тому випадку, коли кнопка відмічена. Не відмічені кнопки при складанні рядка запиту ігноруються цілком. Декілька кнопок типу checkbox можуть мати один атрибут NAME (і різні VALUE), якщо це необхідно. Кнопки типу radio button призначені для одного з усіх запропонованих варіантів і тому повинні мати однаковий атрибут NAME і різні атрибути VALUE.

В принципі створювати HTML -форму для передачі даних методом GET не обов'язково. Можна просто додати в рядок URL потрібні змінні і їх значення.

```
http://phpbook.info/test.php?id=10&user=pit
```

У зв'язку з цим у передачі даних методом GET є один істотний недолік - будь-хто може підробити значення параметрів. Тому не радимо використати цей метод для доступу до захищених паролем сторінок, для передачі інформації, що впливає на безпеку роботи програми або сервера. Крім того, не варто застосовувати метод GET для передачі інформації, яку не дозволено змінювати користувачеві.

Незважаючи на усі ці недоліки, використати метод GET досить зручно при відладці скриптів (тоді можна бачити значення і імена передаваних змінних) і для передачі параметрів, що не впливають на безпеку.

Для методу POST

Вміст форми кодується точно так, як і для методу GET (см вищий), але замість додавання рядка до URL вміст запиту посилається блоком даних як частину операції POST . Якщо є присутнім атрибут ACTION, то значення URL, яке там знаходиться, визначає, куди посилати цей блок даних. Цей метод, як вже відзначалося, рекомендується для передачі великих за об'ємом блоків даних.

Інформація, введена користувачем і відправлена серверу за допомогою методу POST, подається на стандартне введення програмі, вказаній в атрибуті action, або потоковому скрипту, якщо цей атрибут опущень. Довжина посиланого файлу передається в змінній оточення CONTENT_LENGTH, а тип даних - в змінній CONTENT_TYPE.

Передати дані методом POST можна тільки за допомогою HTML -формы, оскільки дані передаються в тілі запиту, а не в заголовку, як в GET . Відповідно і змінити значення параметрів можна, тільки змінивши значення, введене у форму. При використанні POST користувач не бачить передавані серверу дані.

Основна перевага POST запитів - це їх велика безпека і функціональність в порівнянні з GET -запросами. Тому метод POST частіше використовують для передачі важливої інформації, а також інформації

великого об'єму. Проте не варто цілком покладатися на безпеку цього механізму, оскільки дані POST запиту також можна підробити, наприклад створивши html -файл на своїй машині і заповнивши його потрібними даними. Крім того, не усі клієнти можуть застосовувати метод POST, що обмежує варіанти його використання.

При відправці даних на сервер будь-яким методом передаються не лише самі дані, введені користувачем, але і ряд змінних, що називаються змінними оточення, характеризують клієнта, історію його роботи, шляхи до файлів і тому подібне. Ось деякі зі змінних оточення :

CONTENT_LENGTH - Довжина файлу, що надсилається;

CONTENT_TYPE - тип даних, які надсилаються;

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

REMOTE_HOST - ім 'я хоста, з якого відправлений запит;

HTTP_REFERER - адреси сторінки, що посилається на поточний скрипт;

REQUEST_METHOD - метод, який був використаний при відправці запит;

QUERY_STRING - інформація, що знаходиться в URL після знаку питання;

SCRIPT_NAME - віртуальний шлях до програми, яка повинна виконуватися;

HTTP_USER_AGENT - інформація про браузер, який використовує клієнт .

Обробка запитів за допомогою PHP

Досі ми згадували тільки, що запити клієнта обробляються на сервері за допомогою спеціальної програми. Насправді цю програму ми можемо написати самі, у тому числі і на мові PHP, і вона робитиме з отриманими даними усе, що ми захочемо. Для того, щоб написати цю програму, необхідно познайомитися з деякими правилами і інструментами, пропонованими для цих цілей PHP.

Усередині PHP -скрипта є декілька способів діставання доступу до даних, переданих клієнтом по протоколу HTTP. До версії PHP 4.1.0 доступ до таких даних здійснювався по іменах переданих змінних (нагадаємо, що дані передаються у вигляді пар "ім'я змінної, символ "=", значення змінної"). Таким чином, якщо, наприклад, було передано first_name=Nina, то усередині скрипта з'являлася змінна $first_name зі значенням Nina. Якщо вимагалося розрізняти, яким методом були передані дані, то використовувалися асоціативні масиви $HTTP_POST_VARS і $HTTP_GET_VARS, ключами яких були імена переданих змінних, а значеннями - відповідно значення цих змінних. Таким чином, якщо пара first_name = Nina передана методом GET, то $HTTP_GET_VARS["first_name"]="Nina".

Використати в програмі імена переданих змінних безпосередньо небезпечно. Тому було вирішено починаючи з PHP 4.1.0 задіяти для

звернення до змінних, переданих за допомогою HTTP -запросов, спеціальний масив - $_REQUEST . Цей масив містить дані, передані методами POST і GET, а також за допомогою HTTP cookies. Це суперглобальний асоціативний масив, тобто його значення можна отримати у будь-якому місці програми, використовуючи як ключ ім'я відповідної змінної (елементу форми).

Приклад. Припустимо, ми створили форму для реєстрації учасників заочної школи програмування, як в наведеному вище прикладі. Тоді у файлі 1.php, оброблювальному цю форму, можна написати наступне:

```
<?php
$str = "Вітаємо,
    ".$_REQUEST["first_name"]. "
    ".$_REQUEST["last_name"]".! <br>";
$str .=" Ви вибрали для вивчення курс по
    ".$_REQUEST["kurs"];
echo $str;
?>
```

Тоді, якщо у форму ми ввели ім'я "Вася", прізвище "Петров" і вибрали серед усіх курсів курс по PHP, на екрані браузеру отримаємо таке повідомлення:

```
Вітаємо, Вася Петренко!
Ви вибрали для вивчення курс по PHP
```

Після введення масиву $_REQUEST масиви $HTTP_POST_VARS і $HTTP_GET_VARS для однорідності були перейменовані в $_POST і $_GET відповідно, але самі вони з ужитку не зникли з міркувань сумісності з попередніми версіями PHP. На відміну від своїх попередників, масиви $_POST і $_GET стали суперглобальними, тобто доступними безпосередньо і усередині функцій і методів.

Наведемо приклад використання цих масивів. Припустимо, нам треба обробити форму, що містить елементи введення з іменами first_name, last_name, kurs (наприклад, форму form.html, приведену вище). Дані були передані методом POST, і дані, передані іншими методами, ми обробляти не хочемо. Це можна зробити таким чином:

```
<?
php
$str = "Вітаємо, .$_POST ["first_name"]".
    ".$_POST ["last_name"] ".! <br>";
$str .= "Ви вибрали для вивчення курс по ".
        $_POST["kurs"];
echo $str;
?>
```

Тоді на екрані браузеру, якщо ми ввели ім'я "Вася", прізвище "Петров" і вибрали серед усіх курсів курс по PHP, побачимо повідомлення, як в попередньому прикладі:

```
Вітаємо, Вася Петренко!
Ви вибрали для вивчення курс по PHP
```

Для того, щоб зберегти можливість обробки скриптів більше ранніх версій, ніж PHP 4.1.0, була введена директива register_globals, що дозволяє або забороняє доступ до змінних безпосередньо по їх іменах. Якщо у файлі налаштувань PHP параметр register_globals=On, то до змінних, переданих серверу методами GET і POST, можна звертатися просто по їх іменах (тобто можна писати $first_name). Якщо ж register_globals=Off, то треба писати $_REQUEST["first_name"] чи $_POST["first_name"], $_GET["first_name"], $HTTP_POST_VARS["first_name"], $HTTP_GET_VARS["first_name"]. З точки зору безпеки цю директиву краще відключати (тобто register_globals=Off). При включеній директиві register_globals перелічені вище масиви також міститимуть дані, передані клієнтом.

Іноді виникає необхідність упізнати значення якої-небудь змінної оточення, наприклад метод, що використався при передачі запиту або IP - адрес комп'ютера, що відправив запит. Отримати таку інформацію можна за допомогою функції getenv(). Вона повертає значення змінної оточення, ім'я якої передане їй в якості параметра.

```
<?
getenv('REQUEST_METHOD');
    // поверне використаний метод
echo getenv ('REMOTE_ADDR');
    // виведе IP -адрес користувача,
    // що надіслав запит
?>
```

Як ми вже говорили, якщо використовується метод GET, то дані передаються додаванням рядка запиту у вигляді пар "ім'я_змінної=значення до URL -адресу ресурсу". Усе, що записано в URL після знаку питання, можна отримати за допомогою команди

```
getenv('QUERY_STRING');
```

Завдяки цьому можна по методу GET передавати дані в якому-небудь іншому виді. Наприклад, вказувати тільки значення декількох параметрів через знак плюс, а в скрипті розбирати рядок запиту на частини або можна передавати значення усього одного параметра. В цьому випадку в масиві $_GET з'явиться порожній елемент з ключем, рівним цьому значенню (усьому рядку запиту), причому символ " + ", що зустрівся в рядку запиту, буде замінений на підкреслення "_".

Методом POST дані передаються тільки за допомогою форм, і користувач (клієнт) не бачить, які саме дані вирушають серверу. Щоб їх побачити, хакер повинен підмінити нашу форму своїй. Тоді сервер відправить результати обробки неправильної форми не туди, куди треба. Щоб цього уникнути, можна перевіряти адресу сторінки, з якою були послані дані. Це можна зробити знову ж таки за допомогою функції getenv() :

```
getenv('HTTP_REFERER');
```

Тепер самий час вирішити завдання, сформульоване на початку лекції.

Приклад обробки запиту за допомогою PHP

Нагадаємо, в чому полягало завдання, і уточнимо її формулювання. Треба написати форму для реєстрації учасників заочної школи програмування і після реєстрації відправити учасникові повідомлення. Ми назвали це повідомлення універсальним листом, але воно трохи відрізнятиметься від того листа, який ми склали на попередній лекції. Тут ми також не відправлятимемо що-небудь по електронній пошті, щоб не уподібнюватися спамерам, а просто згенеруємо це повідомлення і виведемо його на екран браузеру. Початковий варіант форми реєстрації ми вже приводили вище. Змінимо його так, щоб кожен той, що реєструється міг вибрати скільки завгодно курсів для відвідування, і не підтверджуватимемо отримання реєстраційної форми.

```
<h2>Форма для реєстрації студентів</h2>
<form action="1.php" method=POST>
Ім'я <br><input type=text name="first_name"
    value="Введіть Ваше ім'я"><br>
Прізвище <br><input type=text name="last_name"><br>
E - mail <br><input type=text name="email"><br>
<p> Виберіть курс, який ви б хотіли відвідувати :<br>
<input type=checkbox name='kurs[]' value='PHP'>PHP<br>
<input type=checkbox name='kurs[]' value='Lisp'>Lisp<br>
<input type=checkbox name='kurs[]' value='Perl'>Perl<br>
<input type=checkbox name='kurs[]' value='Unix'>Unix<br>
<P>Що ви хочете, щоб ми знали про вас? <BR>
<textarea name="comment" cols=32 rows=5></textarea>
<input type=submit value="Відправити">
<input type=reset value="Відмінити">
</form>
```

Тут усе досить просто і зрозуміло. Єдине, що можна відмітити, - це спосіб передання значень елементу checkbox. Коли ми пишемо в імені елементу kurs[], це означає, що перший відмічений елемент checkbox буде записаний в перший елемент масиву kurs, другий відмічений checkbox, - в другий елемент масиву і так далі. Можна, звичайно, просто дати різні імена елементам checkbox, але це ускладнить обробку даних, якщо курсів буде багато.

Скрипт, який усе це розбиратиме і оброблятиме, називається 1.php (форма посилається саме на цей файл, що записано в її атрибуті action). За умовчанням використовується для передачі метод GET, але ми вказали POST . За отриманими даними від людини, що зареєструвалася, скрипт генерує відповідне повідомлення. Якщо людина вибрала якісь курси, то йому виводиться повідомлення про час їх проведення і про лекторів, які їх читають. Якщо людина нічого не вибрала, то виводиться повідомлення про наступні збори заочної школи програмістів (ЗШП).

Скрипт 1.php, що обробляє форму form_final.html

```
<? // створимо масиви відповідностей курс-час його
// проведення і курс-лектор
$times = array("PHP"=>"14.30", "Lisp"=>"12.00",
    "Perl"=>"15.00", "Unix"=>"14.00");
$lectors = array("PHP"=>"Василь Васильович", "Lisp"=> "Іван
Іванович",    "Perl"=>"Петро    Петрович",    "Unix"=>"Семен
Семенович");
define("SIGN", "З повагою, адміністрація");
    // визначаємо підпис листа як константу
define("MEETING_TIME", "18.00");
    // задаємо час зборів студентів
$date = "12 травня"; // задаємо дату проведення лекцій
    //починаємо складати текст повідомлення
$str = "Здрастуйте, шановний ". $_POST["first_name"]
    . " " . $_POST["last_name"]".!<br>";
$str .= "<br>Повідомляємо Вас, що;
$kurses = $_POST["kurs"];   // збережемо в цій змінній
                            // список вибраних курсів
if (!isset($kurses)){ // якщо не вибраний жоден курс
    $event = "наступні збори студентів";
    $str.="$event відбудеться $date ".MEETING_TIME. "<br>";
} else { // якщо хоч би один курс вибраний
    $
event = "вибрані Вами лекції відбудуться $date <ul>";
        //функція count обчислює число елементів в масиві
    $lect = "";
    for ($i=0;$i<count($kurses);$i++){
                            // для кожного вибраного курсу
        $k = $kurses[$i]; // запам'ятовуємо назву курсу
        $lect = $lect . "<li>лекція з $k в $times[$k]";
                            // складаємо повідомлення
        $lect .= " (Ваш лектор, $lectors[$k])";
    }
    $event = $event . $lect . "</ul>";
    $str .= "$event";
}
$str .= "<br>". SIGN; // добавляем подпись
echo $str; // выводим сообщение на экран
?>
```

2.4.2 Функції в PHP

Функції, визначувані користувачем

Для чого потрібні функції? Щоб відповісти на це питання, треба зрозуміти, що взагалі є функції. У програмуванні, як і в математиці, функція є відображення безлічі її аргументів на безліч її значень. Тобто функція для кожного набору значень аргументу повертає якісь значення, що є результатом її роботи. Навіщо потрібні функції, спробуємо пояснити на

прикладі. Класичний приклад функції в програмуванні - це функція, що обчислює значення факторіалу числа. Тобто ми задаємо їй число, а вона повертає нам його факторіал. При цьому не треба для кожного числа, факторіал якого ми хочемо отримати, повторювати один і той же код - досить просто викликати функцію з аргументом, рівним цьому числу.

Функція обчислення факторіалу натурального числа

```
<?php
function fact($n){
  if ($n==0) return 1;
  else return $fact = $n * fact($n - 1);
}
echo fact(3);
   // можна було б написати echo (3*2);
   // але якщо число велике
echo fact(50);
   // те зручніше користуватися функцією
   // чим писати echo (50*49*48*...*3*2);
?>
```

Таким чином, коли ми здійснюємо дії, в яких простежується залежність від якихось даних, і при цьому, можливо, нам знадобиться виконувати такі ж дії, але з іншими початковими даними, зручно використати механізм функцій - оформити блок дій у вигляді тіла функції, а дані, що міняються, - в якості її параметрів.

Подивимося, як в загальному вигляді виглядає завдання (оголошення) функції. Функція може бути визначена за допомогою наступного синтаксису:

```
function Ім 'я_функції (параметр1, параметр2
   ... параметрN){
   Блок_команд
   return "значення, що повертає функція";
}
```

Якщо прямо так написати в php -программе, то працювати нічого не буде. По-перше, Ім'я_функції і імена параметрів функції (параметр1, параметр2 і так далі) повинні відповідати правилам найменування в PHP (і російських символів в них краще не використати). Імена функцій нечутливі до регістра. По-друге, параметри функції - це змінні мови, тому перед назвою кожного з них повинен стояти знак $. Ніяких многоточий ставити в списку параметрів не можна. По-третє, замість слів блок_дій в тілі функції повинен знаходитися будь-якою правильний PHP -код (не обов'язково залежний від параметрів). І нарешті, після ключового слова return повинно йти коректне php -выражение (що-небудь, що має значення). Крім того, у функції може і не бути параметрів, як і повертаного значення. Приклад правильного оголошення функції - функція обчислення факторіалу, приведена вище.

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

```
<?php
```

```
Ім'я_функції ("значення_для_параметра1"
  "значення_для_параметра2",...);
?>
```

Коли можна викликати функцію? Здавалося б, дивне питання. Функцію можна викликати після її визначення, тобто у будь-якому рядку програми нижче блоку function f_name(){...}. У PHP3 це було дійсно так. Але вже в PHP4 такої вимоги немає. Вся річ у тому, як інтерпретатор обробляє отримуваний код. Єдиний виняток становлять функції, визначувані умовно (усередині умовних операторів або інших функцій). Коли функція визначається таким чином, її визначення повинне передувати її виклику.

Якщо функція одного разу визначена в програмі, то перевизначити або видалити її пізніше не можна. Попри те, що імена функцій нечутливі до регістра, краще викликати функцію по тому ж імені, яким вона була задана у визначенні.

Розглянемо детальніше аргументи функцій, їх призначення і використання.

Аргументи функцій

У кожної функції може бути, як ми вже говорили, список аргументів. За допомогою цих аргументів у функцію передається різна інформація (наприклад, значення числа, факторіал якого потрібно підрахувати). Кожен аргумент є змінною або константою.

За допомогою аргументів дані у функцію можна передавати трьома різними способами. Це передача аргументів за значенням (використовується за умовчанням), по посиланню і завдання значення аргументів за умовчанням. Розглянемо ці способи детальніше.

Коли аргумент передається у функцію за значенням, зміна значення аргументу усередині функції не впливає на його значення поза функцією. Щоб дозволити функції змінювати її аргументи, їх треба передавати по посиланню. Для цього у визначенні функції перед ім'ям аргументу слід написати знак амперсанд "*&*".

```
<?php
function add_label(&$data_str){
  $data_str .= "checked";
}
$str = "<input type=radio name=article ";
// нехай є такий рядок
echo $str ".><br>";
// виведе елемент форми - не відмічену радіо кнопку
add_label($str);
// викличемо функцію
echo $str ".><br>";
// це виведе вже відмічену радіо кнопку
?>
```

У функції можна визначати значення аргументів, використовувані за умовчанням. Саме значення за умовчанням має бути константним вираженням, а не змінною і не представником класу або викликом іншої функції.

У нас є функція, що створює інформаційне повідомлення, підпис до якого міняється залежно від значення переданого їй параметра. Якщо значення параметра не задане, то використовується підпис "Оргкомітет".

```
<?php
function Message($sign="Оргкомітет".){
// тут параметр sign має за
// умовчанням значення "Оргкомітет"
  echo "Наступні збори відбудуться завтра.<br>";
  echo $sign . "<br>"; }
Message();
    // викликаємо функцію без параметра.
    // В цьому випадку підпис - це Оргкомітет
Message("З повагою, Вася".);
    // В цьому випадку підпис
    // буде "З повагою, Вася".
?>
```

Якщо у функції декілька параметрів, то ті аргументи, для яких задаються значення за умовчанням, мають бути записані після усіх інших аргументів у визначенні функції. Інакше з'явиться помилка, якщо ці аргументи будуть опущені при виклику функції.

Списки аргументів змінної довжини

У PHP4 можна створювати функції зі змінним числом аргументів. Тобто ми створюємо функцію, не знаючи заздалегідь, із скількома аргументами її викличуть. Для написання такої функції ніякого спеціального синтаксису не потрібно. Усе робиться за допомогою вбудованих функцій func_num_args(), func_get_arg(), func_get_args().

Функція func_num_args() повертає число аргументів, переданих в поточну функцію. Ця функція може використовуватися тільки усередині визначення призначеної для користувача функції. Якщо вона з'явиться поза функцією, то інтерпретатор видасть попередження.

```
<?php
function DataCheck(){
  $n = func_num_args();
  echo "Число аргументів функції $n";
}
DataCheck();
   // виведе рядок "Число аргументів функції 0"
DataCheck(1,2,3);
   // виведе рядок "Число аргументів функції 3"
?>
```

Функція func_get_arg (ціле номер_аргументу) повертає аргумент зі списку переданих у функцію аргументів, порядковий номер якого заданий параметром номер_аргументу. Аргументи функції вважаються починаючи з нуля. Як і func_num_args(), ця функція може використовуватися тільки усередині визначення якої-небудь функції.

Номер_аргументу не може перевищувати число аргументів, переданих у функцію. Інакше буде згенеровано попередження, і функція func_get_arg() поверне False.

Створимо функцію для перевірки типу цих її аргументів. Вважаємо, що перевірка пройшла успішно, якщо перший аргумент функції - ціле число, другий - рядок.

```
<?
function DataCheck(){
  $check =true;
  $n = func_num_args();
     // число аргументів переданих у функцію
      /* перевіряємо, чи являється перший
         переданий аргумент цілим числом */
  if ($n>=1) if (!is_int(func_get_arg(0)))
      $check = false;
      /* перевіряємо, чи являється другий
         переданий аргумент рядком */
  if ($n>=2)
     if (!is_string(func_get_arg(1)))
        $check = false;
return $check;
}
if (DataCheck(123, "text"))
   echo "Перевірка пройшла успішно<br>";
else echo "Дані не задовольняють
   умовам<br>";
if (DataCheck(324))
   echo "Перевірка пройшла успішно<br>";
else echo "Дані не задовольняють умовам<br>";
?>
```

Функція func_get_args() повертає масив, що складається зі списку аргументів, переданих функції. Кожен елемент масиву відповідає аргументу, переданому функції. Якщо функція використовується поза визначенням призначеної для користувача функції, то генерується попередження.

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

```
<?
function DataCheck(){
  $check =true;
```

```
  $n = func_num_args();
  // число аргументів, переданих у функцію
  $args = func_get_args();
  // масив аргументів функції
  for ($i=0;$i<$n;$i++){
    $v = $args[$i];
    if ($i % 2 == 0){
      if (!is_int($v)) $check = false;
      // перевіряємо, чи є парний аргумент цілим
    }
  }
return $check;
}
if (DataCheck("text", 324))
   echo "Перевірка пройшла успішно<br>";
else echo "Дані не задовольняють
   умовам<br>";
?>
```

Як бачимо, комбінації функцій func_num_args(), func_get_arg() і func_get_args() використовується для того, щоб функції могли мати змінний список аргументів. Ці функції були додані тільки в PHP 4. У PHP3 для того, щоб добитися подібного ефекту, можна використати в якості аргументу функції масив. Наприклад, ось так можна написати скрипт, перевіряючий, чи є кожен непарний параметр функції цілим числом:

```
<?
function DataCheck($params){
  $check =true;
  $n = count($params);
  // число аргументів переданих у функцію

  for ($i=0;$i<$n;$i++){
    $v = $params[$i];
    if ($i % 2 != 0){
      // перевіряємо, чи являється непарний
      // аргумент цілим
if (!is_int($v)) $check = false;
    }
  }
return $check;
}
if (DataCheck("text", 324))
echo "Перевірка пройшла успішно<br>";
else echo "Дані не задовольняють умовам<br>";
?>
```

Використання змінних усередині функції

Щоб використати усередині функції змінні, задані поза нею, ці змінні треба оголосити як глобальні. Для цього в тілі функції слід перерахувати їх імена після ключового слова global :

global $var1, $var2;

```
<?
$a=1;
function Test_g(){
global $a;
  $a = $a*2;
  echo 'в результаті роботи функції $a=',$a;
}
echo 'поза функцією $a=',$a';
Test_g();
echo "<br>";
echo 'поза функцією $a=',$a';
Test_g();
?>
```

В результаті роботи цього скрипта отримаємо:

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

Коли змінна оголошується як глобальна, фактично створюється посилання на глобальну змінну. Тому такий запис еквівалентний наступним (масив $GLOBALS містить усі змінні, глобальні відносно поточній зоні видимості) :

```
$var1 = &$GLOBALS["var1"];
$var2 = &$GLOBALS["var2"];
```

Це означає, наприклад, що видалення змінної $var1 не видаляє глобальної змінної $GLOBALS["var1"].

Статичні змінні

Щоб використати змінні тільки усередині функції, при цьому зберігаючи їх значення і після виходу з функції, треба оголосити ці змінні як статичні. Статичні змінні видно тільки усередині функції і не втрачають свого значення, якщо виконання програми виходить за межі функції. Оголошення таких змінних робиться за допомогою ключового слова static :

```
static $var1, $var2;
```

Статичною змінною може бути присвоєне будь-яке значення, але не посилання.

```
<?
function Test_s(){
static $a = 1;
// не можна привласнювати вираження або посилання
```

```
  $a = $a*2;
  echo $a;
}
Test_s(); // виведе 2
echo $a; // нічого не виведе, оскільки  $a доступна тільки
          // усередині функції
Test_s(); // усередині функції $a=2, тому
          // результатом роботи функції
          // буде число 4
?>
```

Значення, що повертаються

Усі функції, приведені вище в якості прикладів, виконували які-небудь дії. Окрім подібних дій, будь-яка функція може повертати як результат своєї роботи яке-небудь значення. Це робиться за допомогою затвердження return. Повертане значення може **бути будь-якого типу, включаючи списки і об'єкти**. Коли інтерпретатор зустрічає команду return в тілі функції, він негайно припиняє її виконання і переходить на той рядок, з якого була викликана функція.

Наприклад, складемо функцію, яка повертає вік людини. Якщо людина не померла, то вік вважається відносно поточного року.

```
<?php
/* якщо другий параметр обчислюється
як true, то він розглядається як дата смерті, */

function Age($birth, $is_dead){
  if ($is_dead) return $is_dead -$birth;
  else return date("Y") -$birth;
}
echo Age(1971, false); // для 2009 року виведе 38
echo Age(1971, 2001); // виведе 30
?>
```

В даному прикладі можна було і не використати функцію return, а просто замінити її функцією виведення echo. Проте якщо ми все ж робимо так, що функція повертає якесь значення (в даному випадку вік людини), то в програмі ми можемо присвоїти будь-якій змінній значення цієї функції :

$an_age = Age(1981, 2004);

В результаті роботи функції може бути повернено тільки одне значення. Декілька значень можна отримати, якщо повертати список значень (одновимірний масив).

Коли функція повертає декілька значень для їх обробки в програмі, зручно використати мовну конструкцію list(), яка дозволяє однією дією присвоїти значення відразу декільком змінним. Наприклад, в попередньому прикладі, залишивши без зміни функцію, обробити повертані їй значення можна було так:

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

Взагалі конструкцію list() можна використати для присвоєння змінним значень елементів будь-якого масиву.

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

Повернення посилання

В результаті своєї роботи функція також може повертати посилання на яку-небудь змінну. Це може згодитися, якщо вимагається використати функцію для того, щоб визначити, якою змінною має бути присвоєне посилання. Щоб отримати з функції посилання, треба при оголошенні перед її ім'ям написати знак амперсанд (&) і кожного разу при виклику функції перед її ім'ям теж писати амперсанд (&). Зазвичай функція повертає посилання на яку-небудь глобальну змінну (чи її частину - посилання на елемент глобального масиву), посилання на статичну змінну (чи її частина) або посилання на один з аргументів, якщо він був також переданий по посиланню.

```
<?
$a = 3; $b = 2;
function & ref($par){
global $a, $b;
  if ($par % 2 == 0) return $b;
  else return $a;
}
$var =& ref(4);
echo $var, " і ", $b"<br>";    //виведе 2 і 2
$b = 10;
echo $var, " і ", $b"<br>";    // виведе 10 і 10
?>
```

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

Змінні функції

PHP підтримує концепцію змінних функцій. Це означає, що якщо ім'я змінної закінчується круглими дужками, то PHP шукає функцію з таким же ім'ям і намагається її виконати.

```
<?
/* створимо дві прості функції:
Add_sign - додає підпис до рядка і
Show_text - виводить рядок тексту */
function Add_sign($string
   $sign="З повагою, Петро"){
echo $string ". ".$sign;
}
function Show_text(){
  echo "Відправити повідомлення поштою<br>";
}
$func = "Show_text";
   // створюємо змінну зі значенням
   // рівним імені функції Show_text
$func();
   // це викличе функцію Show_text
$func = "Add_sign";
   // створюємо змінну зі значенням
   // рівним імені функції Add_sign
$func("Привіт всім <br>");
   // це викличе функцію
   // Add_sign з параметром "Привіт всім"
?>
```

В даному прикладі функція Show_text просто виводить рядок тексту. Здавалося б, навіщо для цього створювати окрему функцію, якщо існує спеціальна функція echo(). Річ у тому, що такі функції, як echo(), print(), unset(), include() і тому подібне не можна використати в якості змінних функцій. Тобто якщо ми напишемо:

```
<?
$func = "echo ";
$func("TEXT");
?>
```

то інтерпретатор виведе помилку:

```
Fatal error: Call to undefined function:
echo() in
c:\users\nina\tasks\func\var_f.php on line 2
```

Тому для того, щоб використати будь-яку з перелічених вище функцій як змінну функцію, треба створити власну функцію, що ми і зробили в попередньому прикладі.

Внутрішні (вбудовані) функції

Говорячи про функції, визначувані користувачем, все ж не можна не сказати пару слів про вбудовані функції. З деякими зі вбудованих функцій, такими як echo(), print(), date(), include(), ми вже познайомилися. Насправді усі перераховані функції, окрім date(), є мовними конструкціями. Вони входять в ядро PHP і не вимагають ніяких додаткових налаштувань і модулів. Функція date() теж входить до складу ядра PHP і не вимагає налаштувань. Але є і функції, для роботи з якими треба встановити різні бібліотеки і підключити відповідний модуль. Наприклад, для використання функцій роботи з базою даних MySql слід скомпілювати PHP з підтримкою цього розширення. Останнім часом найбільш поширені розширення і відповідно до їх функції спочатку включають до складу PHP так, щоб з ними можна працювати без яких би то не було додаткових налаштувань інтерпретатора.

Рішення задачі

Нагадаємо, в чому полягає завдання. Ми хочемо написати інтерфейс, який дозволяв би створювати html -формы. Користувач вибирає, які елементи і в якій кількості треба створити, придумує їм назви, а наша програма сама генерує необхідну форму.

Розіб'ємо завдання на декілька підзадач: вибір типів елементів введення і їх кількості, створення назв елементів введення і обробка отриманих даних, тобто безпосередньо генерація форми. Перше завдання досить просте: треба написати відповідну форму, наприклад подібну до приведеної нижче () :

```
<form action="ask_names.php">
Створити елемент "рядок введення тексту" : <input
  type=checkbox name=types[]
  value=string><br>
Кількість елементів : <input type=text
  name=numbers[string]
  size=3><br>
<br>
Створити елемент "текстова область": <input
  type=checkbox
  name=types[] value=text><br>
Кількість елементів : <input type=text
  name=numbers[text]
  size=3><br>
<input type=submit value="Створити">
</form>
```

Коли ми пишемо в імені елементу форми, наприклад types[], це означає, що його ім'я - наступний елемент масиву types. Тобто у нас перший елемент форми ("рядок введення тексту") матиме ім'я types[0], а другий (текстова область) - types[1]. У браузері task_form.html виглядатиме приблизно так:

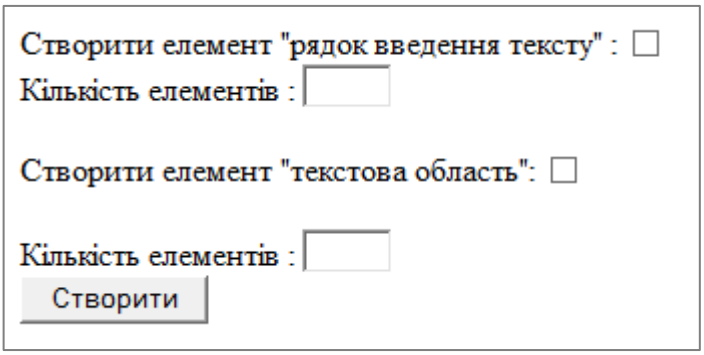


Рисунок 2.4. Форма для вибору створюваних елементів і їх кількості

Після відправки даних цієї форми ми отримаємо інформацію про те, які елементи і скільки елементів кожного типу треба створити. Наступний скрипт ask_names.php просить назви для цих елементів:

```
<?
$file = "task.php";
/*  файл, який оброблятиме
    згенеровану цим скриптом форму */
function Ask_names(){
   // функція генерує форму для
   // введення назв елементів введення
global $file;
   //оголошуємо, що хочемо використати цю
   // змінну, задану поза функцією
if (isset($_GET["types"])){
  $st = '<form action="'.$file.'">';
  foreach ($_GET["types"] as $k => $type){
/* перебираємо усі типи елементів,
    які треба створити */

  $num = $_GET["numbers"][$type];
     // скільки елементів кожного типу потрібні
  for ($i=1;$i<=$num;$i++){
     // створюємо $num рядків для введення
   $st.="Введіть ім'я $i-го елементу типу $type: ";
   $st.="<input type=text name=names[$type][]><br>";
  }
  // зберігаємо тип і число необхідних
  // елементів введення цього типу
$st.="<input type=hidden name=types[] value=$type>";
$st.=  "<input  type=hidden  name=numbers[]  value=
    $num><br>";
}
$st.="<input  type=submit  name=send  value=send>
</form>";
return $st;
```

```
    // у змінній $st міститься код форми
    // для запиту імен
} else echo "Select type";
}
echo Ask_names();
    // викликаємо функцію і виводимо
    // результати її роботи
?>
```

Лістинг 5.13.

Припустимо, треба створити два елементи типу "текстовий рядок" і один елемент типу "текстова область", як і відмічено у формі вище. Тоді скрипт ask_names.php обробить її таким чином, що ми отримаємо таку форму:

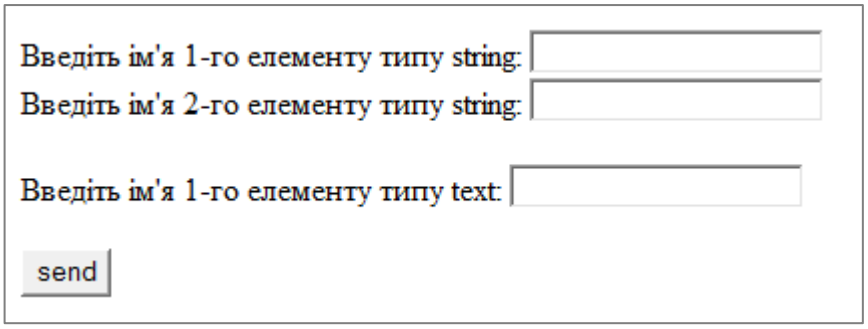


Рисунок 2.5 Форма для введення назв створюваних елементів

Введемо в цю форму, наприклад, рядка "Назва", "Автор" і "Короткий зміст". Ці дані оброблятиме скрипт task.php:

```
<?
$show_file = "task_show.php";
/* файл, який оброблятиме дані
   створеною цим файлом форми */
function Create_element($type,$name){
    // функція створює елемент введення
    // за типом і назвою
  $str="";
  switch($type){
  case "string":
  $str.="$name:<input    type=text    name=string[]>
     <br>";     break;
  case "text":
  $str.="$name:   <textarea   name=text[]></textarea>
     <br>";     break;
   }
return $str;
}
function Create_form(){
    // функція створює форму
```

```
    // з потрібними елементами
global $show_file;
  $str = '<form action="'.$show_file.'">';
  foreach ($_GET["types"] as $k => $type){
  // перебираємо типи елементів
    $num = $_GET["numbers"][$k];
    // число елементів цього типу
    for ($i=1;$i<=$num;$i++){
      $arr = $_GET["names"][$type][$i - 1];
      // ім'я створюваного елементу
      $str.=Create_element($type,$arr);
      // викликаємо функцію для
      // створення елементу
    }
  }
  $str .= "<input type=submit value=send></form>";
  echo $str;
}
$crt = "Create_form";
$crt(); // викликаємо функцію створення
        // форми Create_form
?>
```

Результатом роботи цього скрипта з вхідними даними, приведеними вище, буде наступна форма:

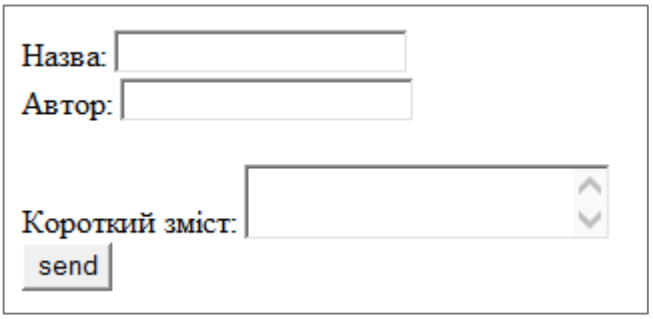


Рисунок 2.6 Приклад форми, що генерується програмою

Тема 2.5. Створення web-додатків засобами PHP

2.5.1 Сеанси і сесії в PHP

Змінна сеансу є спеціальним типом змінної, значення якої зберігається при переході між послідовними сторінками Web. С допомогою змінних сеансу специфічні дані користувача можна зберігати при переході із сторінки на сторінку, надаючи йому індивідуалізований контент при взаємодії користувача з додатком Web. Змінні сеансу існують зазвичай, поки не виконується одна з наступних подій :

- користувач закриває вікно браузеру;
- завершення заданого на сервері максимального інтервалу часу сеансу;
- використання функцій PHP, таких, як session_destroy(), щоб звільнити усі існуючі в даний момент змінні сеансу.

Проблема, яку повинні вирішити змінні сеансу, полягає в тому, що протокол HTTP, використовуваний в Web, не має стану. Кожен запит сторінки абсолютно незалежний від попередніх запитів, тому, якщо вимагається, щоб подальші сторінки "запам'ятовували" ім'я користувача, яке він вводить на першій сторінці, то цю інформацію потрібно десь зберегти.

Система PHP має ряд засобів для підтримки сеансів. Це керівництво розпочинається з введення глобальної змінної $_SESSION[]. Змінну $_SESSION[] рекомендується використати для поліпшення безпеки і легкості для читання коду.

Також вводяться функції сеансу session_start() і session_destroy(). Кожна з них визначена нижче:

$_SESSION[] - суперглобальний масив PHP, який містить зареєстровані в даний момент змінні сеансу сценарію. У нім, власне, і знаходяться змінні, які ми ходимо зробити доступними в різних скриптах. Щоб помістити змінну в сесію, досить присвоїти її елементу масиву $_SESSION.

Щоб отримати її значення - досить звернутися до того ж елементу.

session_start() - ініціалізація даних сеансу. Ця функція викликається перед створенням нової змінної сеансу за допомогою $_ SESSION[].

session_destroy() - руйнує усі дані, зареєстровані для потокового сеансу сценарію

Використовується також функція PHP header("Location".). Хоча це функція HTTP, а не функція сеансу, вона зазвичай використовується для перенаправлення користувачів під час існуючих сеансів. Ця функція визначена нижче.

header("Location: http://www.domain.com") - функція header використовується для перенаправлення браузеру на сторінку, визначену параметром Location.

Наступний блок коду демонструє, як ініціалізувалася змінна сеансу.

```
<?php
session_start();
if ($_SESSION['count'] == "")
{
	$_SESSION['count'] = 1;
}
else
{
	$_SESSION['count'] = $_SESSION['count'] + 1;
	}
?>
```

В даному прикладі спочатку викликається функція session_start() для ініціалізації даних сеансу. Функція session_start() повинна викликатися перед створенням і привласненням значень змінним сеансу. Потім використовується оператор if для перевірки значення змінної сеансу "count". Якщо сеанс буде null або не містить значення, то він ініціалізувався як 1, інакше значення змінної сеансу збільшується на 1. В цьому випадку змінна сеансу count використовується для підрахунку числа відвідувачів. Змінні сеансу надають ідеальний спосіб створення лічильників сторінок, оскільки кожному користувачеві надається унікальний сеанс.

Змінна сеансу може створюватися також після успішного введення користувачем свого імені і пароля, які потрібні для доступу до конфіденційного сайту. В цьому випадку змінна сеансу містить значення, яке передається із сторінки на сторінку, вказуючи, що користувач має право доступу до будь-яких ресурсів, пов'язаних з сайтом. Коли користувач виходить або вирішує "завершити", змінна сеансу повинна повертатися в початковий стан. Це виконується за допомогою функції session_destroy().

login.php

```
<?php
if ($_POST['submit'] == "Login")
{
     // Сценарій перевірки імені користувача
     // і пароля повинен кодуватися тут
     // Якщо аутентифікація пройшла успішно
     session_start();
     $_SESSION['access'] = "yes";
     header("Location:access.php");
}
if ($_POST['submit'] == "Log Out")
{
     //якщо користувач вирішує вийти
     session_destroy();
}
?>
```

Файл access.php

```
<?php
// якщо користувач звертається до цієї сторінки
// перевірити, що він був
// аутентифікований в login.php
if ($_SESSION['access'] != "yes")
{
    header(Location:login.php);
}
?>
```

Приклад вище складається з двох сторінок — login.php і access.php. Перш ніж користувач зможе побачити вміст access.php, він повинен пройти login.php. Створюється сеанс, щоб гарантувати, що користувач був аутентифицирован. Після того, як користувач вводить дійсне ім'я користувача і пароль, виконується клацання на кнопці "Login". Сценарій PHP перевіряє, що пароль і ім'я користувача допустимі. Потім змінною сеансу привласнюється значення "yes" і доступ надається. Користувач перенаправляється на сторінку access.php. Сценарій на access.php перевіряє, що існує сеанс. Якщо сеанс не існує, застосовується функція header(), щоб перенаправити користувача назад на сторінку login.php і не дозволити доступ до вмісту сторінки access.php.

Сторінка login.php містить другий сценарій, який використовується для знищення змінної сеансу, викликаючи функцію session_destroy(). Сценарій виконується після того, як користувач клацає на кнопці "Log Out".

Додатки Web, які використовують дані сеансу, можуть бути доступні одночасно безлічі користувачів. Щоб кожен користувач мав свій власний сеанс, з кожним сеансом необхідно зв'язати унікальне значення id. У PHP це унікальне значення сеансу id можна витягнути за допомогою функції session_id(). Унікальне значення сеансу session_id() підтримується для кожного користувача і зберігається в підкаталозі PHP/sessiondata, розташованому на сервері Web.

Оскільки значення session_id() є унікальним для кожного користувача, його можна застосовувати для ідентифікації користувачів, не створюючи при цьому індивідуальних імен користувачів і паролів. Функція session_id() визначена нижче:

session_id() - використовується для отримання значення id потокового сеансу.

Файли Cookies

Файл cookie є повідомленням від браузеру Web -серверу. Браузер зберігає повідомлення в текстовому файлі. Це повідомлення посилається потім назад на сервер кожного разу, коли браузер просить сторінку з сервера.

Основне призначення cookies полягає в ідентифікації користувачів і можливій підготовці спеціально налагодженої для них сторінки Web. При відвідуванні сайту Web, використовуючого cookies, на сайті може бути

запропоновано заповнити форму, щоб надати таку інформацію, як своє ім'я і можливі інтереси. Ця інформація упаковується в cookie і посилається браузеру Web, який зберігає її для подальшого використання. Коли ви наступного разу відвідуєте той же самий сайт Web, браузер пошле cookie серверу Web. Сервер може використати цю інформацію, щоб створити індивідуалізовані сторінки Web. Тому, наприклад, замість звичайної вітальної сторінки можна побачити вітальну сторінку зі своїм ім'ям.

У PHP файли cookies створюють за допомогою функції setcookie(). Усі дані cookie зберігаються в глобальній змінній PHP $_COOKIE і доступні для подальших сторінок.

setcookie(name, value, expiration, path, domain, security) - визначає файл cookie, який посилається разом з іншими заголовками HTTP. Як і інші заголовки, файли cookie повинні посилатися до якого-небудь виведення роботи сценарію(це обмеження протоколу). Тому вимагається, щоб звернення до функції було поміщене до будь-якого виводу, включаючи теги і будь-які символи роздільники. Якщо вивід відбувається до звернення до цієї функції, то setcookie() не виконається і поверне FALSE. Якщо setcookie() виконується успішно, то повертається TRUE. Це не вказує на ті, що користувач прийняв cookie.

Параметри setcookie() пояснюються в таблиці 2.15.

Таблиця 2.15 Параметри setcookie

Параметр	**Опис**
name	Ім'я cookie. Цей ідентифікатор зберігається в глобальній змінній $_COOKIE і доступний в подальших сценаріях
value	Значення cookie. Значення, пов'язане з ідентифікатором cookie. Зберігається на комп'ютері користувача, тому не повинно містити секретної інформації
expiration	Час, коли витікає значення cookie або стає більше недоступним. Цей час можна задати за допомогою функції time(). Файли cookie без заданого значення часу витікання завершують своє існування при закритті браузеру
path	Вказує шляху доступу на сервері, для яких cookie дійсний або доступний. Пряма коса риска "/" говорить, що cookie доступний в усіх теках
domain	Домен, в якому доступний cookie. Якщо домен не визначений, за умовчанням використовується хост, на якому створений cookie. Значення domain повинні містити в рядку як мінімум дві точки ""., щоб бути допустимими
security	Вказує, чи буде cookie передаватися через HTTPS. Значення 1 означає, що cookie передається через захищене з'єднання. Значення 0 обоз1начает стандартну передачу HTTP

2.5.2 Взаємодія PHP і MySQL

У дистрибутив PHP входить розширення, що містить вбудовані функції для роботи з базою даних MySQL. У цій лекції ми познайомимося з деякими основними функціями для роботи з MySQL, які знадобляться для вирішення завдань побудови web -интерфейсов з метою відображення і наповнення бази даних. Виникає питання, навіщо будувати такі інтерфейси? Для того, щоб вносити інформацію у базу даних і переглядати її вміст могли люди, не знайомі з мовою запитів SQL. При роботі з web -интерфейсом для додавання інформації у базу даних людині треба просто ввести ці дані в html -форму і відправити їх на сервер, а наш скрипт зробить усе інше. А для перегляду вмісту таблиць досить просто клацнути по посиланню і зайти на потрібну сторінку.

Для наочності будуватимемо ці інтерфейси для деякої таблиці Artifacts, в якій міститься інформація про експонати віртуального музею інформатики. Кожен експонат в колекції Artifacts описується за допомогою наступних характеристик:

- назва (title);
- автор (author);
- опис (description);
- альтернативна назва (alternative);
- зображення (photo).

Назва і альтернативна назва є рядками менш ніж 255 символів завдовжки (тобто мають тип VARCHAR(255)), опис - текстове поле (має тип TEXT), а в полях "автор" і "зображення" містяться ідентифікатори автора з колекції Persons і зображення експоната з колекції Images відповідно.

Отже, у нас є якась таблиця у базі даних. Щоб побудувати інтерфейс для додавання інформації в цю таблицю, треба її структуру (тобто набір її полів) відобразити в html -форму.

Розіб'ємо це завдання на наступні підзадачі:

- установка з' єднання з БД ;
- вибір робочої БД ;
- отримання списку полів таблиці ;
- відображення полів в html - форму.

Після цього дані, введені у форму, треба записати у базу даних. Розглянемо усі ці завдання по порядку.

Отже, перше, що треба зробити, - це встановити з' єднання з базою даних. Скористаємося функцією mysql_connect.

Синтаксис mysql_connect

```
ресурс mysql_connect ([рядок server
    [, рядок username [, рядок password
    [, логічне new_link [, ціле client_flags]]]])
```

Ця функція встановлює з' єднання з сервером MySQL і повертає

покажчик на це з'єднання або FALSE у разі невдачі. Для відсутніх параметрів встановлюються наступні значення за умовчанням:

- server = 'localhost: 3306'
- username = ім 'я користувача власника процесу сервера
- password = порожній пароль

Якщо функція викликається двічі з одними і тими ж параметрами, то нове з'єднання не встановлюється, а повертається посилання на старе з'єднання. Щоб цього уникнути, використовують параметр new_link, який примушує у будь-якому випадку відкрити ще одне з'єднання.

Параметр client_flags - це комбінація наступних констант: MYSQL_CLIENT_COMPRESS (використати протокол стискування), MYSQL_CLIENT_IGNORE_SPACE(дозволяє вставляти пропуски після імен функцій), MYSQL_CLIENT_INTERACTIVE (чекати interactive_timeout секунд - замість wait_timeout - до закриття з' єднання).

Параметр new_link з'явився в PHP 4.2.0, а параметр client_flags - в PHP 4.3.0.

З' єднання з сервером закривається при завершенні виконання скрипта, якщо воно до цього не було закрите за допомогою функції mysql_close().

Отже, встановлюємо з' єднання з базою даних на локальному сервері для користувача nina з паролем "123":

```
<?
$conn = mysql_connect("localhost", "nina", "123")
or die("Неможливо встановити
    з'єднання: ". mysql_error());
echo "З'єднання встановлене";
mysql_close($conn);
?>
```

Дія mysql_connect рівносильно команді:

```
shell>mysql -u nina -p123
```

Після установки з'єднання треба вибрати базу даних, з якою працюватимемо. Наші дані зберігаються у базі даних book. У MySQL вибір бази даних здійснюється за допомогою команди use :

```
mysql>use book;
```

У PHP для цього існує функція mysql_select_db.

Синтаксис mysql_select_db :

```
логічне mysql_select_db ( рядок database_name
    [, ресурс link_identifier])
```

Ця функція повертає TRUE у разі успішного вибору бази даних і FALSE - інакше.

Зробимо базу даних book робочою:

```
<?
$conn = mysql_connect(
    "localhost", "nina", "123")
or die("Неможливо встановити
        з' єднання: ". mysql_error());
```

```
echo "З' єднання встановлено";
mysql_select_db("book");
?>
```

Тепер можна зайнятися власне рішенням задачі. Як отримати список полів таблиці? Дуже просто. У PHP і на цей випадок є своя команда - mysql_list_fields.

Синтаксис mysql_list_fields

```
ресурс mysql_list_fields ( рядок database_name,
    рядок table_name [, ресурс link_identifier])
```

Ця функція повертає список полів в таблиці table_name у базі даних database_name. Виходить, що вибирати базу даних нам було необов'язкове, але це згодиться пізніше. Як можна помітити, результат роботи цієї функції - змінна типу ресурс. Тобто це не зовсім те, що ми хотіли отримати. Це посилання, яке можна використати для отримання інформації про поля таблиці, включаючи їх назви, типи і прапори.

Функція mysql_field_name повертає ім'я поля, отриманого в результаті виконання запиту. Функція mysql_field_len повертає довжину поля. Функція mysql_field_type повертає тип поля, а функція mysql_field_flags повертає список прапорів поля, записаних через пропуск. Типи поля можуть бути int, real, string, blob і так далі. Прапори можуть бути not_null, primary_key, unique_key, blob, auto_increment і так далі

Синтаксис у усіх цих команд однаковий:

```
рядок mysql_field_name (ресурс result, ціле field_offset)
рядок mysql_field_type (ресурс result, ціле field_offset)
рядок mysql_field_flags(ресурс result, ціле field_offset)
рядок mysql_field_len (ресурс result, ціле field_offset)
```

Тут result - це ідентифікатор результату запиту (наприклад, запиту, відправленого функціями mysql_list_fields або mysql_query (про неї буде розказано пізніше)), а field_offset - порядковий номер поля в результаті.

Взагалі кажучи, те, що повертають функції типу mysql_list_fields або mysql_query, є таблицею, а точніше, покажчик на неї. Щоб отримати з цієї таблиці конкретні значення, треба задіяти спеціальні функції, які відрядковий читає цю таблицю. До таких функцій і відносяться mysql_field_name і тому подібне. Щоб перебрати усі рядки в таблиці результату виконання запиту, треба знати число рядків в цій таблиці. Команда mysql_num_rows (ресурс result) повертає число рядків у безлічі результатів result.

А тепер спробуємо отримати список полів таблиці Artifacts (колекція експонатів).

```
<?
$conn = mysql_connect(
    "localhost", "nina", "123")
or die("Неможливо встановити
        з'єднання: ". mysql_error());
echo "З'єднання встановлене";
mysql_select_db("book");
$list_f = mysql_list_fields (
```

```
  "book", "Artifacts",$conn);
  $n = mysql_num_fields($list_f);
for($i=0;$i<$n; $i++){
  $type = mysql_field_type($list_f, $i);
  $name_f = mysql_field_name($list_f,$i);
  $len = mysql_field_len($list_f, $i);
  $flags_str =  mysql_field_flags (
              $list_f, $i);
echo "<br>Ім'я поля : ". $name_f;
echo "<br>Тип поля : ". $type;
echo "<br>Довжина поля : ". $len;
echo "<br>Рядок прапорів поля : ".
    $flags_str . "<hr>";
}
?>
```

В результаті повинно вийти приблизно ось що (якщо в таблиці всього два поля, звичайно) :

Ім'я поля : id
Тип поля : int
Довжина поля : 11
Рядок прапорів поля :
not_null primary_key auto_increment
Ім'я поля : title
Тип поля : string
Довжина поля : 255
Рядок прапорів поля :

Тепер трішки підкоригуємо попередній приклад. Не просто виводитимемо інформацію про поле, а відображати його у відповідний елемент html -формы. Так, елементи типу BLOB переведемо в textarea (помітимо, що поле description, яке ми створювали з типом TEXT, відображається як що має тип BLOB), числа і рядки відобразимо в текстові рядки введення <input type=text>, а елемент, що має мітку автоінкремента, взагалі не відображатимемо, оскільки його значення встановлюється автоматично.

Усе це вирішується досить просто, за винятком виділення зі списку прапорів прапора auto_increment. Для цього треба скористатися функцією explode.

Синтаксис explode :

```
масив explode( рядок separator, рядок string [, int limit])
```

Ця функція розбиває рядок string на частини за допомогою роздільника separator і повертає масив отриманих рядків.

У нашому випадку в якості роздільника треба узяти пропуск " ", а в якості початкового рядка для розбиття - рядок прапорів поля.

Отже, створимо форму для введення даних в таблицю Artifacts:

```
<?
$conn=mysql_connect("localhost", "nina", "123");
// встановлюємо з' єднання
$database = "book";
$table_name = "Artifacts";
// вибираємо базу даних для роботи
mysql_select_db($database);
// отримуємо список полів в таблиці
$list_f = mysql_list_fields($database,$table_name);
// число рядків в результаті попереднього
//запиту (тобто скільки всього  полів в таблиці
$n = mysql_num_fields($list_f);
// створюємо форму для введення даних
echo "<form method=post action=insert.php>";
echo " <TABLE BORDER=0 CELLSPACING=0 width=50% ><tr>
<TD  BGCOLOR='#005533' align=center>
    <font color='#FFFFFF'> <b> Add new row in $table_name
    </b></font></td></tr><tr><td></td></tr></TABLE>";
echo  "<table  border=0  CELLSPACING=1  cellpadding=0
width=50% >";
// для кожного поля отримуємо його ім'я, тип, довжину і
прапорці
for($i=0;$i<$n; $i++){
    $type = mysql_field_type($list_f, $i);
    $name_f = mysql_field_name ($list_f,$i);
    $len = mysql_field_len($list_f, $i);
    $flags_str = mysql_field_flags ($list_f, $i);
    // з рядка прапорців робимо масив, де кожен елемент
    //масиву - прапор поля
        $flags = explode(" "flags_str);
        foreach ($flags as $f){
            if ($f == 'auto_increment') $key = $name_f;
        }
    /* для кожного поля, що не є автоінкрементом, в
    залежності від його типу виводимо
    відповідний елемент форми */
    if ($key <> $name_f){
    echo "<tr><td align=right bgcolor='#C2E3B6'><font
        size=2><b> ". $name_f ".</b></font></td>";
    switch ($type){
        case "string":
          $w = $len/5;
          echo "<td><input type=text name=\"$name_f\"
                    size = $w ></td>";
          break;
```

```
            case "int":
            $w =  $len/4;
            echo "<td><input type=text name=\"$name_f\"
                      size = $w ></td>";
            break;
            case "blob":
            echo "<td><textarea rows=6 cols=60
              name=\"$name_f\"></textarea></td>";
            break;
              }
          }
          echo "</tr>";
      }
      echo "</table>";
      echo "<input type=submit name='add' value='Add'>";
      echo "</form>";
      ?>
```

Отже, форма створена. Тепер треба зробити найголовніше - відправити дані з цієї форми в нашу базу даних. Як ви вже знаєте, для того, щоб записати дані в таблицю, використовується команда INSERT мови SQL. Наприклад:

```
mysql> INSERT INTO Artifacts
            SET title='Петренко';
```

Виникає питання, як можна скористатися такою командою (чи будь-якою іншою командою SQL) в PHP скрипті. Для цього існує функція mysql_query().

Синтаксис mysql_query

```
ресурс mysql_query (рядок query
     [,ресурс link_identifier])
```

mysql_query() посилає SQL -запрос активній базі даних MySQL сервера, який визначається за допомогою покажчика link_identifier (це посилання на якесь з'єднання з сервером MySQL). Якщо параметр link_identifier опущений, використовується останнє відкрите з'єднання. Якщо відкриті з'єднання відсутні, функція намагається з'єднатися з СУБД, аналогічно функції mysql_connect() без параметрів. Результат запиту буферизируется.

Зауваження: рядок запиту НЕ повинен закінчуватися крапкою з комою.

Тільки для запитів SELECT, SHOW, EXPLAIN, DESCRIBE, mysql_query() повертає покажчик на результат запиту, або FALSE, якщо запит не був виконаний. У інших випадках mysql_query() повертає TRUE, якщо запит виконаний успішно, і FALSE - у разі помилки. Значення, не рівне FALSE, говорить про те, що запит був виконаний успішно. Воно не говорить про кількість рядів, що торкнулися або повернених. Цілком можлива ситуація, коли успішний запит не торкнеться жодного ряду. mysql_query()

також вважається помилковим і поверне FALSE, якщо у користувача недостатньо прав для роботи з вказаною в запиті таблицею.

Отже, тепер ми знаємо, як відправити запит на вставку рядків у базу даних. Помітимо, що в попередньому прикладі елементи форми ми назвали іменами полів таблиці. Тому вони будуть доступні в скрипті insert.php, оброблювальному ці форми, як змінні виду $_POST['ім'я_поля']:

```
<?
$conn=mysql_connect("localhost", "nina", "123");
$database = "book";
$table_name = "Artifacts";
mysql_select_db($database);
$list_f = mysql_list_fields($database,$table_name);
$n = mysql_num_fields($list_f);
// складемо один запитий відразу для
// усіх полів таблиці
// починаємо створювати запит,
$sql = "INSERT INTO $table_name SET ";
//  перебираємо усі поля таблиці
for($i=0;$i<$n; $i++){
  // обчислюємо ім 'я поля
  $name_f = mysql_field_name($list_f,$i);
  // обчислюємо значення поля
  $value = $_POST[$name_f];
    $j = $i + 1;
  // дописуємо в рядок $sql пару ім'я=значення
  $sql = $sql.$name_f".='$value'";
  // якщо поле не останнє в списку, то ставимо кому
  if($j <> $n)  $sql = $sql"., "; }
  // перш ніж записувати щось у базу
  // можна подивитися, який запитий вийшов
  //echo $sql;
// відправляємо запит
$result = mysql_query($sql,$conn);
// виводимо повідомлення чи успішно
// виконаний запит
if (!$result) echo " Can't add ($table_name) ";
    else echo "Success!<br>";
?>
```

Отже, завдання додавання даних за допомогою web -интерфейса ми вирішили. Проте тут є одна тонкість. При рішенні ми не враховували той факт, що значення деяких полів (author, photo) повинні братися з інших таблиць (Persons, Images). Оскільки MySQL із зовнішніми ключами не працює (вже працює - прим. експерта), цей момент залишається на совісті розробників системи, тобто на нашій совісті. Треба дописати програму так,

щоб була можливість вводити в такі поля правильні значення. Але ми робити цього не будемо, оскільки завдання лекції полягає в тому, щоб познайомити читача з елементами технології, а не в тому, щоб створити працюючу систему. Крім того, наявних у читача знань цілком достатньо, щоб розв'язати цю проблему самостійно. Ми ж звернемося до іншого завдання - відображення даних, що зберігаються у базі цих СУБД MySQL.

Щоб відобразити якісь дані у браузері за допомогою PHP, треба спочатку отримати ці дані у вигляді змінних PHP. При роботі з MySQL без посередника (такого, як PHP) вибірка даних робиться за допомогою команди SELECT мови SQL :

mysql> SELECT * FROM Artifacts;

У попередній главі ми говорили, що будь-який запит, у тому числі і на вибірку, можна відправити на сервер за допомогою функції mysql_query(); Там у нас стояло трохи інше завдання - отримати дані з форми і відправити їх за допомогою запиту на вставку у базу даних. Результатом роботи mysql_query() там могло бути тільки один з виразів, TRUE або FALSE. Тепер же вимагається відправити запит на вибір усіх полів, а результат відобразити у браузері. І тут результат - це ціла таблиця значень, а точніше, покажчик на цю таблицю. Так що потрібні якісь аналоги функції mysql_field_name(), тільки щоб вони витягали з результату запиту не ім'я, а значення поля. Таких функцій в PHP декілька. Найбільш популярні - mysql_result() і mysql_fetch_array().

Синтаксис mysql_result

```
змішане mysql_result (ресурс result,
    ціле row [, змішане field])
```

mysql_result() повертає значення одного осередку результату запиту. Аргумент field може бути порядковим номером поля в результаті, ім'ям поля або ім'ям поля з ім'ям таблиці через точку tablename.fieldname. Якщо для імені поля в запиті застосовувався аліас ('select foo as bar from...'), використайте його замість реального імені поля.

Працюючи з великими результатами запитів, слід задіяти одну з функцій, оброблювальних відразу цілий ряд результату (наприклад, mysql_fetch_row(), mysql_fetch_array() і так далі). Оскільки ці функції повертають значення декількох осередків відразу, вони НАБАГАТО швидше mysql_result(). Крім того, треба врахувати, що вказівка чисельного зміщення (номери поля) працює набагато швидше, ніж вказівка колонки або колонки і таблиці через точку.

Виклики функції mysql_result() не повинні змішуватися з іншими функціями, працюючими з результатом запиту.

Синтаксис mysql_fetch_array

```
масив mysql_fetch_array ( ресурс result
    [, ціле result_type])
```

Ця функція обробляє ряд результату запиту, повертаючи масив (асоціативний, чисельний або обоє) з обробленим рядом результату запиту,

або FALSE, якщо рядів більше немає.

mysql_fetch_array() - це розширена версія функції mysql_fetch_row(). Окрім зберігання значень в масиві з чисельними індексами, функція повертає значення в масиві з індексами по назві колонок.

Якщо декілька колонок в результаті матимуть однакові назви, буде повернена остання колонка. Щоб отримати доступ до перших, слід використати чисельні індекси масиву або аліаси в запиті. У разі аліасів саме їх ви не зможете використати в іменах колонок, як, наприклад, не зможете використати "photo" в описаному нижче прикладі.

```
select Artifacts.photo as art_image,
    Persons.photo as pers_image
    from Artifacts, Persons
```

Важливо помітити, що mysql_fetch_array() працює НЕ повільніше, ніж mysql_fetch_row(), і надає зручніший доступ до даних.

Другий опциональный аргумент result_type у функції mysql_fetch_array() є константою і може набувати наступних значень: MYSQL_ASSOC, MYSQL_NUM і MYSQL_BOTH. Ця можливість додана в PHP 3.0.7. Значенням за умовчанням є: MYSQL_BOTH.

Використовуючи MYSQL_BOTH, отримаємо масив, що складається як з асоціативних індексів, так і з чисельних. MYSQL_ASSOC поверне тільки асоціативні відповідності, а MYSQL_NUM - тільки чисельні.

Зауваження: імена полів, повертані цією функцією, регистрозависимы.

Тепер відобразимо дані з Artifacts у вигляді таблиці у браузері:

```
<?
/* спочатку робимо ті ж, що і раніше: встановлюємо
з'єднання, вибираємо базу і отримуємо список і
число полів в таблиці Artifacts */
$conn=mysql_connect("localhost", "nina", "123");
$database = "book";
$table_name = "Artifacts";
mysql_select_db($database);
$list_f = mysql_list_fields($database,$table_name);
$n1 = mysql_num_fields($list_f);
// збережемо імена полів в масиві $names
for($j=0;$j<$n1; $j++){
    $names[] = mysql_field_name ($list_f,$j);
}
// створюємо SQL запит
$sql = "SELECT * FROM $table_name";
// відправляємо  запит на сервер
$q = mysql_query($sql,$conn) or die();
// отримуємо число рядків результату
$n = mysql_num_rows($q);
//малюємо HTML - таблицю
echo " <TABLE BORDER=0 CELLSPACING=0 width=90%
```

```
        align=center><tr>
    <TD BGCOLOR='#005533' align=center>
    <font color='#FFFFFF'><b>$table_name</b></font> </td>
    </tr></TABLE>";
    echo "<table cellspacing=0 cellpadding=1 border=1
            width=90% align=center>";
    // відображаємо назви полів
    echo "<tr>";
    foreach ($names as $val){
       echo "<th ALIGN=CENTER BGCOLOR='#C2E3B6'>
             <font size=2>$val</font></th>";
        }
     // відображаємо значення полів
     echo "</tr>";
     for($i=0;$i<$n; $i++){ // перебираємо усі рядки в
         // результаті запиту на вибірку
       echo "<tr>";
       foreach  ($names as $val){ // перебираємо всі
                  // імена полів
       $value = mysql_result($q,$i,$val); // отримуємо
                  // значення поля
       echo "<td><font size=2>  $value</font></td>";
                 // виводимо значення поля
            }
        echo "</tr>";
        }
    echo "</table>";
```

Зробимо те ж саме за допомогою mysql_fetch_array() :

```
<?
/* ... початок  аналогічний, як
    і в попередньому прикладі */
// відображаємо значення полів
// надаємо значення поля у вигляді асоціативного масиву
while($row = mysql_fetch_array($q, MYSQL_ASSOC)){
    echo "<tr>";
    foreach  ($names as $val){
        echo "<td><font size=2> 
             $row[$val]</font></td>";
            // виводимо значення поля
            }
        echo "</tr>";
        }
    echo "</table>";
    ?>
```

Контрольні питання до розділу 2

1. Правила запису функцій в мові JavaScript.
2. Що таке об’єкт в мові JavaScript?
3. Як визначити об’єкт користувача в JavaScript?
4. Стандартні об’єкти JavaScript.
5. Як реалізована інтерактивність в JavaScript?
6. Стандартні функції JavaScript.
7. Методи та властивості об’єкту Array.
8. Методи та властивості об’єкту Date.
9. Методи та властивості об’єкту Math.
10. Методи та властивості об’єкту String.
11. Методи та властивості об’єкту Window?
12. Як за допомогою JavaScript відкрити нове вікно браузеру ?
13. Як за допомогою JavaScript запрограмувати виконання деяких команд після закінчення встановленого терміну часу?
14. Методи та властивості об’єкту Document.
15. Методи та властивості об’єкту Location.
16. Що таке PHP?
17. В чому, власне, програмувати PHP-скрипти?
18. Що потрібно, щоб мої скрипти на PHP працювали?
19. Як взаємодіють PHP і HTML?
20. Як оголосити константу в PHP?
21. Як додавати коментар в текст скрипта?
22. Як здійснити простий вивід на PHP?
23. Як оголосити змінну в PHP?
24. Як розбити рядок на слова?
25. Як перевірити, чи була натиснута submit-кнопка, чи відвідувач щойно прийшов на сторінку?
26. Як відбувається ініціалізація масиву в PHP?
27. Що таке регулярні вирази?
28. Як перенаправити користувача на інший URL в PHP-скрипті?
29. Як надіслати e-mail за допомогою PHP?
30. Є форма, в яку вноситься текст з перенесенням рядків, при виведенні тексту на сторінку рядки не переносяться, що робити?
31. Як мені отримати вчорашню дату?
32. Як перевірити, чи дана змінна визначена?

РОЗДІЛ 3. ПІДТРИМКА ТА ПРОСУВАННЯ ВЕБ-ПРОЕКТІВ

Тема 3.1. Оптимізація структури веб-проекту

3.1.1 Методи оптимізації веб-проектів

Як відомо, критичними параметрами Web-середовища є час завантаження і швидкість відображення сторінки, які тісно пов'язані один з одним. Розглядаючи ці параметри як критичні, ми враховуємо те, що робота Web -системи відбувається набагато повільніше Windows, що компілюють, -додатків за інших рівних умов, навіть якщо Web -додаток працює локально.

Одне з основних завдань Web-швидке отримання і обробка інформації, тому питання збільшення продуктивності Web-системи буде актуальним постійно, оскільки потужніші системи дозволяють вирішувати і складніші завдання, тобто с ростом продуктивності Web-системи відбуватиметься і ускладнення вирішуваних за допомогою її завдань. Простий приклад. Перші обчислювальні машини виконували прості арифметичні дії. На сучасних системах проводять складні наукові розрахунки, і було б, щонайменше, дивно використати їх там, де цілком вистачить функціональності і продуктивності звичайного калькулятора. По цьому шляху розвивається і Інтернет, але повільніше, хоча володарі високошвидкісного каналу вже можуть насолодитися красивою анімацією і тривимірними віртуальними світами.

Порівнюючи можливості Windows -додатків і Web -додатків легко помітити, що перші на декілька порядків перевершують другі. При цьому, як ми з'ясували вище, функціональність застосування залежить від швидкості його роботи. Типовий приклад - виведення таблиці даних, коли відображення таблиці вже з 300 записами є для браузеру дуже непростим завданням, тоді як звичайне застосування без зусиль може завантажити декілька десятків тисяч записів. Звичайно, поява таких баз даних як MySQL відкрило перед розробниками нові можливості для створення потужних баз даних в Інтернеті, але, проте, реалізувати цілий ряд необхідних користувачам механізмів, наприклад, інкрементного пошуку і замовлення товару в реальному масштабі часу, представляється для Web дуже непростим завданням. Деякі читачі можуть запитати, а чи так потрібні в Інтернеті можливості традиційних застосувань, адже дуже багато користувачів використовують його в основному для пошуку інформації і окрім поля пошуку, в яке треба ввести запит, ним більше нічого і не треба? Для пошуку інформації, можливо, і не треба, проте для вирішення великого класу прикладних завдань, наприклад, замовлення товарів в інтернет-магазині, простого пошуку буде явно недостатньо, оскільки окрім пошуку інформації потрібно ще і її швидку обробку. Припустимо, що у нас є два застосування, але за допомогою першого застосування виконання певної дії проводиться на 10 хвилин швидше. Якщо розглянути одноразову втрату 10 хвилин, то це,

можливо, і некритично. Але якщо щодня на кожному етапі роботи кожен співробітник організації втратить усього лише по 10 хвилин, то за робочий тиждень це може скласти вже дуже відчутні втрати часу. Звідси низька ефективність роботи співробітників і їх швидка стомлюваність, втрата клієнтів, темпів розвитку організації і мн. ін. Яку роль в цьому грає Інтернет? Не найменшу, якщо врахувати інтенсивність роботи з ним і можливості, які він може запропонувати (чи не запропонувати) користувачам. Тут як в спорті, якщо хочеш бути першим, то необхідно керуватися не лише основними моментам в підготовці, але і враховувати найдрібніші деталі, які самі по собі непомітні, але в сумі призводять до загального успіху. Чи можна у рамках обмежень Web -середовища добитися збільшення ефективності як роботи самого Web -додатку, так і роботи з ним користувача на якісному рівні? Можна, проте доведеться чимось жертвувати для оптимізації проекту під конкретне програмне забезпечення, яке зможе забезпечити необхідну продуктивність і функціональність застосування. Браузер IE вже з версії 5.0 може працювати з модальними і безрежимными вікнами, без яких не обходиться практично жодне Windows -додаток, тоді як NN 6.0 не підтримує їх. Хтось може помітити, що можна обійтися і без модальних вікон, і без деяких можливостей DHTML. Можна, але за рахунок збільшення часу завантаження інформації і її об'єму, зменшення функціональності ресурсу, можливості оперативної роботи з ним і так далі (хоча на сьогоднішній момент у користувачів є дві альтернативи: працювати більш менш ефективно з тими ресурсами, які вже існують в мережі або не працювати з ними, вирішуючи завдання за допомогою, наприклад, Excel або 1С). Звичайно, користувачі не використовуватимуть у своїй роботі складний Web -додаток на усе 100%, проте більш менш функціональне застосування дозволить їм виконати необхідні дії набагато швидше, ніж просте, пропонуюче тільки ряд стандартних функцій. Узяти приміром можливість складання замовлення в інтернет-магазині декількох сотень найменувань за декілька хвилин, що у разі стандартного рішення займе у користувача декілька годин (днів) з витратою пристойної суми за доступ в Інтернет. Чи варто користувачам витратити 10-15 хвилин на вивчення цієї можливості, щоб потім заощадити місяці роботи?

Думаю, що це невелике введення якщо і не переконало Вас в необхідності включення процесу оптимізації в планування і реалізацію Web -додатку, то хоч би дало привід для повернення до цього питання в майбутньому. Ми ж продовжимо вивчення теми с оптимізації статичних html -сторінок, перейшовши потім до оптимізації динамічних html -сторінок і Web -додатків. Відмітимо відразу, що питання оптимізації розглядатиметься з технічного боку виходячи з проблемної орієнтації Web -ресурсу. Останнє зауваження дуже часто упускають з виду, що відразу перекладає питання оптимізації на деякий абстрактний технічний рівень, який слабо пов'язаний з самим Web -ресурсом і вирішуваними ним завданнями. Річ у тому, що в Web - ще дуже молодий напрям, який знаходиться у стадії пошуку оптимальних рішень різного роду прикладних завдань. З цієї причини нині існує лише

невелика кількість типових схем побудови інформаційних ресурсів. Але, як відомо, типові варіанти далеко не завжди підходять для якісного вирішення специфічних для цієї області діяльності питань. У традиційному програмуванні ми можемо використати величезну кількість прикладних рішень або мов програмування, оскільки продуктивність Windows -додатку в основному залежить від апаратного забезпечення, тобто продуктивність апаратного забезпечення може компенсувати неоптимальний вибір програмної технології для вирішення практичних завдань. Таким чином, розробник може створювати велику кількість різних застосувань за допомогою однієї мови програмування. У Web ситуація інша. З цієї причини при розгляді питання оптимізації слід відштовхуватися саме від проблемної орієнтації вирішуваних завдань, виходячи з якої необхідно зробити вибір оптимальної схеми побудови Web -додатку, і тільки після цього перейти до оптимізації конкретних програмних елементів, тобто оптимізація Web - додатку відбувається в два етапи, які, можливо, доведеться повторювати не один раз для отримання необхідного ефекту.

3.1.2 Способи оптимізації Web-додатків

Існує два основні способи оптимізації Web-додатків:

- Загальна оптимізація
- Оптимізація під конкретний браузер

У літературі в основному описуються загальні прийоми оптимізації, які підходять для великого числа браузерів. Оптимізація під конкретний браузер - тонший спосіб, який частенько змахує на "трюкацтво", проте ці "трюки" дозволяють часто значно зменшити розмір html -сторінок і прискорити її завантаження в порівнянні із загальною оптимізацією.

При рішенні практичних завдань необхідно розглянути можливість реалізації двоступінчатої оптимізації, коли спочатку проводиться загальна оптимізація, а після неї оптимізація під конкретний браузер. Останнє не застосовно, якщо поставлено завдання розробити Web-додаток який повинен адекватно відображатися на як можна більшій кількості браузерів, що може істотно обмежити можливості по його оптимізації, та як молодші версії браузерів не підтримую або некоректно працюють з новими програмними технологіями і конструкціями. В деяких випадках використовують декілька варіантів коду під кожен браузер, проте при цьому відбувається значне збільшення програмного коду, та і усунути за допомогою коду помилки різних браузерів представляється непростим заняттям. Ось ще одна причина, по якій практичний розробляти Web-ресурси тільки для лінійки версій одного (найчастіше використовуваного) браузеру, наприклад, IE 5.0 і вище (приклад загальної оптимізації і оптимізації під конкретний браузер буде розглянутий далі).

Оптимізація статичних html -сторінок

Якщо подивитися на код типовою статичною html -сторінок, то можна побачити, що він складається з наступних елементів:

- Інформація - видимі на сторінці текст і графіка
- HTML -розмітка - теги HTML, їх властивості і параметри

Допоміжна інформація - видимі на сторінці елементи графічного і текстового оформлення і невидимі при її перегляді текст і графіка : метаінформація, коментарі та ін.

У загальному випадку, обов'язковими елементами на сторінці є інформація і HTML -разметка. Допоміжна інформація може бути відсутньою.

Як ви знаєте, одним їх основних критичних параметрів Web - середовища являється швидкість завантаження сторінки, на яку безпосередньо впливає її розмір і розмір усіх графічних файлів, на які є посилання в її коді. Чим більше розмір сторінки, тим довше вона завантажується і відображається у браузері. Іншим важливим параметром є швидкість відображення сторінки, яка залежить не лише від її розміру, але і від інших параметрів, і зокрема від html -разметки. Виходячи з цього ми можемо визначити наступні елементи сторінки, які нас цікавлять з точки зору поліпшення параметрів :

- Текст
- Графіка
- HTML -разметка

Чи можна виділити в цьому списку елемент, важливіший з точки зору оптимізації, чим інші або ні? Думаю, більшість читачів дадуть відповідь, що це графіка. У загальному випадку це вірно, оскільки навіть відносно невелике графічне зображення задовільної якості може "важити" більше, ніж уся корисна інформація на сторінці разом з html -разметкой. З іншого боку, на сторінках часто можна побачити як надмірний текст, так і об'ємну розмітку, створену візуальними редакторами html, яка містить велику кількість зайвих тегів. Тому необхідно уважно подивитися на код сторінки і спробувати виділити ті її елементи, які вносять найбільший вклад в загальну "вагу" сторінки. Для такої оцінки може знадобитися певна кількість часу, проте без цього можна довго намагатися оптимізувати другорядні елементи, внаслідок чого загальний ефект буде дуже незначним. Мало сенсу приступати до тонкої оптимізації HTML -структуры, коли на сторінці є присутніми два необроблені зображення по 40Кб.

Чим менше вага сторінки, тим більше важливими з точки зору оптимізації стають окремі програмні блоки коду. Типовий приклад - лічильники, об'єм коду яких може бути рівним 1Kb і більше. Звичайно, якщо розмір сторінки дорівнює 100Kb, то це не так істотно, але якщо розмір сторінки дорівнює 20Kb, то код лічильника може займати вже 5-7% від об'єму сторінки, що може бути порівнянне з ефектом від застосування деяких способів оптимізації.

Окремо хочеться зупинитися на форматуванні коду сторінки

візуальними редакторами, наприклад, Dreamweaver. Багато розробників мало звертають увагу на те, що зліва від коду часто без всякого на те підстави розташовуються відступи, а довгі рядки коду розбиваються на декілька коротких та ін. Такі "дурнички" в масштабі усієї сторінки можуть істотно збільшити її об'єм. Як приклад був вибраний абзац тексту на цій сторінці, що розпочинається з фрази чи "Можна виділити в цьому списку.". і подивився його розмір з цими відступами і без них. З відступами розмір тексту склав 1441 байт, а без відступів - 1103 байти, що приблизно на 33% менше первинного розміру. Є над чим замислитися, чи не так? Виходів з цієї ситуації може бути декілька: редагувати код в іншому редакторові (HomeSite та ін.), прибирати відступи за допомогою програмного фільтру або вручну після остаточного редагування тексту.

Оптимізація тексту

Оптимізація тексту здійснюється досить просто: необхідно зменшити його об'єм і (чи) розмістити на декількох сторінках з посиланнями на них.

Зменшення об'єму тексту, як підкреслюють фахівці з юзабіліті, є в деяких випадках одним з основних прийомів оптимізації, оскільки, багато сайтів переобтяжені текстом, що важко сприймається при читанні з екрану. В цьому випадку цікаво подивитися як приклад на сайт новин, де на першій сторінці дається короткий огляд якої-небудь події, а у кінці огляду посилання на сторінку з його детальним описом. Непоганим варіантом буде використання одного рядка для анонса події, як це зроблено на початковій сторінці Yandex. Користувачі набагато більш охоче прочитають текст декількох помітних однорядкових посилань, чим учитуватимуться у великі масиви тексту.

Істотно зменшити об'єм завантажуваного тексту можна здійснити за допомогою фреймів, якщо завантажувати в нього сторінки, що містять тільки форматований текст без інших елементів - шапки сторінки з банерами, елементів навігації, допоміжної інформації та ін., які в деяких випадках "важать" набагато більше основного тексту сторінки.

Оптимізація графіки

Графіка - це один з найпідступніших елементів сторінки, на який частенько починають звертати увагу тільки тоді, коли розмір сторінки вже зашкалює далеко за 100Kb, тому необхідно постійно контролювати розмір і об'єм графіки на сторінці. Особливо це характерно для випадку, коли кодування сторінки проводиться по її затвердженому красивому графічному ескізу з великою кількістю заливок і різних спецефектів. Типовий приклад - використання широкої шапки сторінки з великим логотипом, графічних посилань для навігації та ін. Без графіки не обійтися при побудові ігрових і мультимедійних застосувань, проте для оформлення ділових ресурсів слід використати її в розумних межах, оскільки тут важливо швидкість і зручність

роботи з ним. Зверніть увагу на інтерфейс більшості Windows -додатків, який виглядає спокійно і по-діловому.

Оптимізація графіки здійснюється різними способами:

- Зменшення фізичного розміру графічного зображення. Чому саме фізичного розміру, а не просто розміру? Припустимо, що для анонса деякої події на сторінці потрібне зображення розміром 400x300px, а в наявність є зображення 1024x768px. Зрозуміло, що для отримання фотографії необхідного розміру треба в графічному редакторові зменшити початкове зображення і зберегти його для Web з урахуванням відношення якість/розмір. Мені доводилося стикатися з тим, що цю операцію здійснюють за допомогою тега <image>, задавши необхідну ширину за допомогою властивості тега width, забуваючи про те, що ця властивість впливає на відображення вже завантаженої фотографії і, отже, ніяк не може впливати на розмір її файлу;
- Попереднє виведення невеликого зображення з посиланням на зображення великого розміру, що часто використовується при виведенні розділу каталогу товарів в інтернет-магазинах;
- Повторне використання графічних елементів. Коли браузер уперше завантажує зображення, він тимчасово зберігає його копію у своїй кеш-пам'яті, тому при повторному запиті на відображення цього зображення воно буде завантажено з кеш-пам'яті, а не з сервера;
- Розрізання великого зображення на декілька зображень із застосуванням різного коефіцієнта стискування або графічного формату. Цей прийом часто використовується при створенні графічної шапки сторінки, коли в ній є присутніми різні графічні елементи: логотип організації, фотографії, текст, суцільні області кольору, заливки та ін. Однорідні області кольору і текст зберігають в GIF -формате, а фотографії і заливки в JPEG – форматі;
- Використання для фотозображень JPEG-формата з підбором коефіцієнта стискування і згладжування;
- Виключення згладжування для тексту з маленьким розміром шрифту і застосування спеціальних фільтрів;
- Зменшення кількості кольорів в зображенні (для GIF і PNG форматів);
- Заміна градієнтних переходів одноколірними областями (для GIF намагаються створювати одноколірні області в горизонтальному напрямі, оскільки в цьому випадку відбувається краще стискування. PNG проводить стискування по обох осях);
- Застосування чересстрочного показу (для GIF і PNG форматів), коли зображення відображається в декілька проходів з поступовим збільшенням чіткості і кількості деталей в зображенні, що виводиться;
- Застосування прогресивного формату JPEG (подібно до чересстрочному показу GIF і PNG файлів);
- Використання FLASH;
- Векторизація растрового зображення.

- Використання псевдосимволів замість графічних зображень;
- Заміна графічних елементів текстовими та ін.

Оптимізація html -розмітки

Залежно від конкретного завдання оптимізація html -разметки може здійснюватися наступними способами:

- Чищення коду після операцій експорту/імпорту інформації у формат html і після роботи різних візуальних редакторів, які готові при кожній слушній нагоді "допомогти" розробникові додати на сторінку пару-трійку "правильних" з точки зору алгоритму їх роботи тегів
- Заміна одній html -структуры на іншу (наприклад, таблиці на список, або нумерований список на список з розривами рядків та ін.). Це комбінований прийом, який дозволяє не лише зменшити розмір сторінки, але і збільшити швидкість її відображення у браузері
- Оптимізація під браузер - видалення необов'язкових символів, наприклад, лапок в значеннях параметрів тегів та ін.
- Стискування коду шляхом видалення символів переведення каретки і нового рядка
- Розбиття великої html -таблицы на декілька маленьких
- Використання фреймів
- Застосування CSS та ін.

Давайте розглянемо один з прикладів оптимізації html -разметки під IE (варіант оптимізації під конкретний браузер). Припустимо, що для виведення товарів використовується наступна таблиця:

```
<table border="0" cellspacing="0" cellpadding="8" width="100%">
<tr>
<td align="center">Код товару</td>
<td align="center">Назва товару</td>
<td align="center">Ед.</td>
<td align="center">Ціна</td>
</tr>
<tr>
<td align="center">123</td>
<td>Телевізор</td>
<td align="center">шт.</td>
<td align="right">9000</td>
</tr>
        * * *
</table>
```

у якій знаходяться 100 записів (увесь запис виділений жирним шрифтом). Очевидно, що розмір таблиці визначається розміром усіх записів, тому необхідно як можна компактніше провести кодування кожного запису.

У нашому прикладі розмір html -разметки для одного запису дорівнює

100 байтам (цікаве те, що корисна інформація "важить" всього 19 байт, тобто в 5 разів менше).

Давайте приберемо переклади рядків і складемо запис в один рядок:

<tr><td align="center">123</td><td>Телевізор</td><td align= "center"> шт.</td><td align="right">9000</td></tr>

Такий варіант розмітки важитиме 93 байти, тобто ми зменшили розмір запису на 7%. Браузери IE дозволяють не ставити лапки в параметрах властивостей, тому їх можна також прибрати. Отримаємо розмір запису 87 байтів (13% зменшення розміру). Подальше зменшення розміру можна здійснити двома шляхами: видалити деякі або усі властивості align з корекцією ширини стовпців, яке приведе до зміни і навіть спотворення форматування тексту, що не завжди допустимо або використати стилі CSS. У останньому випадку запис може виглядати так:

<tr><td id=c>123</td><td>Телевізор</td><td id=c>шт.</td><td id=r> 9000</td></tr>

Отримаємо розмір розмітки 60 байт, що на 40% менше початкового варіанту. Зробивши поправку на додатковий код опису стилю, ми бачимо, що за допомогою оптимізації html -разметки вдалося зменшити її розмір на 35-38%. Цікаво відмітити, що останній варіант оптимізації відбувається чисто на технічному рівні без спотворень інформації і її форматування.

Оптимізація статичного Web -додатку

При оптимізації статичного Web -додатку необхідно розглянути роботу усієї Web -системи Web -сервер, канал передачі даних між Web -сервером у браузером, браузер і апаратне забезпечення клієнта. В цьому випадку механізм роботи Web -системи буде таким:

- Відправка запиту серверу на отримання html –сторінок;
- Отримання запиту сервером, обробка його і відправка html -сторінок браузеру;
- Передача html -сторінок через канал передачі даних;
- Відображення браузером отриманої html –сторінок.

Ми бачимо, що механізм роботи системи полягає в передачі і відображенні прошеною html -сторінок, тому основним критичним параметром є розмір статичної html -сторінок. У більшості випадків ми не можемо змінювати параметри каналу передачі даних і настроювати Web -сервер, тому ці елементи опускаємо з розгляду, як і апаратне забезпечення клієнта. Крім того, ми розглядаємо оптимізацію самих html -сторінок (статичного Web -додатку), а не Web -систему в цілому.

Оптимізація динамічних html -сторінок

Основна відмінність динамічної html -сторінки від статичної полягає в тому, що в ній є присутнім програмний код, який транслюється браузером. З цієї причини останні дуже чутливо відносяться до побудови тієї або іншої

програмної конструкції, особливо при роботі з різними об'єктами, циклами і елементами форми. Ця проблема посилюється тим, що довідники і керівництво по DHTML призначені в першу чергу для демонстрації застосування тій або іншій функції, і приклади, що наводяться для цього, при певних обставинах здатні на 20% і більше знижувати продуктивність програмного коду, в яких вони використовуються при практичному їх використанні. Тому розробникам доводиться напрацьовувати досвід в цій області здебільшого методом проб і помилок.

Оптимізація динамічних сторінок може проводитися двома шляхами: зменшення розміру сторінки і збільшення швидкості роботи програмних блоків. Перш ніж перейти до оптимізації програмного коду, давайте подивимося, як зміниться процес оптимізації тексту, графіки і HTML - разметки в порівнянні із статичними сторінками.

Оптимізація тексту

Для зменшення об'єму передаваного тексту використовується кодування, що можна застосувати при передачі у браузер, наприклад, каталогу товарів, в якому одне і теж назва товару зустрічається кілька разів. Як приклад можна на сервері створити довідковий масив з усіма назвами, що повторюються, і здійснити його передачу клієнтові разом з даними. При отриманні даних з сервера здійснюється створення довідкового масиву, і відновлення назви товару за його кодом (цей метод дозволив мені в одному проекті понизити об'єм завантажуваного каталогу товарів на 30%)

У ряді випадків для відображення текстової інформації можна скористатися тимчасовим вікном, створеним на JavaScript за допомогою методу window.open(). Це можна використати для завантаження, наприклад, опису товару, що може зменшити розмір сторінки на 50 і навіть більше відсотків, якщо взяти до уваги той факт, що розмітка сторінки з навігацією і іншими елементами часто перевищує об'єм основного текстового блоку. Незважаючи на очевидну перевагу, дуже мало ресурсів використовують цей спосіб для зменшення мережевого трафіку і прискорення завантаження інформації. Чому? Мені здається, що це відбувається з наступних причин:

Відбувається як би "випадання" відвідувача з контексту сайту. Це можна усунути, якщо спрощено оформити інформацію в тимчасовому вікні під дизайн сайту.

Проблеми рекламний-маркетингово плану. Дійсно, пошуковим системам більше "подобається" повноцінні сторінки, та і багато користувачів, що потрапив на сторінку, яка містить текст, що тільки відформатував, можуть на ній і закінчити своє знайомство з ресурсом. Цього можна уникнути, якщо оформити сторінку з описом товару під загальний дизайн сайту.

Протиріччя основної концепції зв'язку гіпертексту, коли користувач може повернутися на попередню сторінку при клацанні на кнопці браузеру "Назад" ("Back"). У нашому випадку він може повернутися на попередню

сторінку, після закриття тимчасового вікна або клацнувши в зоні головного вікна.

Користувач може заплутатися у великій кількості відкритих тимчасових вікон (дуже актуальне зауваження, мені доводилося самому чути здивування співрозмовника кількості відкритих на моєму робочому столі застосувань). Ця проблема легко вирішується за допомогою модальних вікон (NN не підтримує), хоча це нетипово для Інтернету і користувач може не відразу зрозуміти, чому він не може перейти в головне вікно сайту.

Тим Web -розробникам, які після прочитання цієї інформації все одно категорично відмовляться від застосування тимчасових вікон, я можу порадити знайти Windows-додаток, в якому опис товару (а не анотація) відкривається в головному вікні програми і при цьому працювати з програмою і описом (копіювати його, роздруковувати, зберігати та ін.) зручніше, ніж в застосуванні, де воно відкривається в окремому вікні (підказка: це можна зробити за допомогою багатосторінкових панелей, проте власний досвід показав, що це не дуже зручно і практично).

Оптимізація графіки

Для прискорення завантаження браузером графічних зображень часто використовують метод попереднього завантаження зображень в кеш браузеру за допомогою об'єкту image. Це дещо збільшує час завантаження сторінки, але коли потрібно буде відобразити зображення станеться завантаження локальної копії зображення з кеш-пам'яті браузеру, що відбувається значно швидше за завантаження зображення з сервера.

Ще один неочевидний прийом полягає в тому, щоб завантажувати з сервера за запитом зображення без HTML -разметки і допоміжній інформації. Типовий приклад - відображення фотографії товару. У багатьох випадках при цьому відкривається нова html -сторінока з повним текстовим і графічним оформленням - шапка сторінки з логотипом і банером, навігація, різні сервіси, допоміжний текст та ін., що в сумі може "важити" більше, ніж прошена фотографія. Для завантаження тільки фотографії можна скористатися фреймами або створити за допомогою JavaScript тимчасове вікно за допомогою методу window.open().

Оптимізація HTML -розмітки

Ефект зменшення розміру html -разметки за допомогою програмних алгоритмів можна наочно показати на прикладі, який був розглянутий при оптимізації html -разметки статичної сторінки, де зменшення html -разметки склало 35-38%.

На JavaScript алгоритм виведення записів може бути таким:

```
function DrawTable()
{
for(i=0;i<100;i++){
```

```
document.write("<tr><td
id=c>"+D[i].Code+"</td><td>"+D[i].Name+"</td><td
id=c>"+D[i].Ed+"</td><td
id=c>"+D[i].Cost+"</td></tr>");
}}
```

Зробивши поправку на програмний код функції і структуру оформлення даних, можна помітити, що "вага" розмітки зменшилася десь на 80-90%, тобто програмне рішення в нашому випадку дозволило добитися зменшення "ваги" розмітки в 2 рази, в порівнянні із статичною оптимізацією. "Але це ж прекрасно" - вигукне інший читач, що зацікавився, дістаючи з полиці книгу, що запорошила, з JavaScript, - "тепер я зможу значно прискорити роботу свого сайту". Почекайте, не усе так просто. Вище ми розглянули теоретичну модель, а на практиці цей виграш частково або повністю йде на реалізацію необхідною для роботи з даними функціональність. Саме у цьому і полягає якісна відмінність між оптимізацією статичного Web -додатку від оптимізації динамічного Web -додатку, коли в останньому випадку з'являється побічний ефект - оптимізація Web -додатку призводить до збільшення його функціональності і (чи) її розширення. У даному контексті відбувається розділення поняття функціональності на дві складові - базова функціональність і додаткова функціональність, яка призначена для реалізації необхідних для ефективнішої роботи користувачів функцій. Для розуміння відмінностей між цими поняттями давайте розглянемо інтернет-магазин. Як відомо, він складається з декількох програмних блоків, які здійснюють роботу з масивом даних, виведення інформації, роботу із замовленням, здійснюють різні перевірки даних, що вводяться, обробку подій, розрахунки і мн. ін. Неважко помітити, що ці функції є базовими для побудови будь-якого интрнет-магазина, тобто існує деякий мінімальний набір функцій, без якого неможливо побудувати інтернет-магазин, - функції для роботи з каталогом товарів, замовленням, формою відправки замовлення та ін. Додаткова функціональність, як випливає з її назви, повинна надати покупцям додатковий сервіс при роботі з інтернет-магазином, іншими словами, базова функціональність відповідає за роботу самого Web -додатку і можливість роботи з ним користувача, тоді як розширена функціональність відповідає за зручність робота користувач з він. Це необхідно розуміти, оскільки виграш від оптимізації Web -додатку повинен піти більшою мірою на реалізацію додаткової функціональності, оскільки саме вона визначає, чи зможе користувач вирішити завдання за допомогою Web -ресурса за 10 хвилин або йому доведеться витратити на це 1 годину. Ми дійшли ще одного цікавого висновку: ефективність роботи користувача з Web -додаткомм залежить не лише від ефективності роботи останнього, але і від його додаткової функціональності, за рахунок якої повільніше за технічними параметрами застосування може обійти по практичності використання швидше. При цьому додаткова функціональність повинна розроблятися в контексті роботи цього Web -додатку. Типовий приклад - розширений пошук, який має бути не лише

як формальний доважок до простого пошуку.

Оптимізація Web -додатків

У попередньому параграфі ми розглянули процес оптимізації динамічних html -сторінок, який, в загальному випадку, впливає на ефективність роботи лише окремих частин Web -додатку. Візьмемо, приміром, ефективний механізм пошуку товарів в каталозі інтернет-магазину. Чи означає це, що покупцям зручно працювати з таким ресурсом? Звичайно немає, оскільки окрім пошукової системи необхідно грамотно розробити і інші функціональні частини інтернет-магазину - каталог товарів, систему замовлення товарів, реєстрацію покупців і відправку замовлень та ін. Зрозуміло, що необхідно оптимізувати кожну з цих складових. А для цього треба паралельно розглядати специфіку роботи майбутнього Web -додатку і можливості різних програмних технологій для реалізації необхідних функцій. Інакше висока вірогідність вибору неефективного програмного рішення і окрім втрати часу оптимізація окремих частин ресурсу нічого практично не дасть.

Давайте згадаємо спрощену структуру Web -системи:

Web -сервер - канал передачі даних - браузер користувача.

З схеми видно, що програмний код може виконуватися як на сервері, так і у браузері користувача :

Програмний код виконується тільки у браузері - програмний код JavaScript або ін.

Програмний код виконується тільки на Web -сервері - серверні скрипти PHP, Perl та ін.

Програмний код виконується як у браузері, так і на Web -сервері.

Перш ніж ми перейдемо до детального розгляду цих варіантів, необхідно звернути увагу на те, що програмний код може виступати як код управління або як допоміжний код. У першому випадку ми вважатимемо, що за допомогою програмного коду здійснюється управління усім Web -ресурсом (наприклад, інтернет-магазин), тоді як другою відноситься до допоміжних операцій, наприклад, перевірка правильності заповнення полів форми, підписка на новини, робота з форумом (якщо він є частиною ресурсу) та ін. Основна різниця між ними полягає в тому, що оптимізація коду управління впливає на увесь Web -ресурс, а оптимізація допоміжного коду тільки на окремі його частини. Це необхідно враховувати, оскільки якісна робота окремих частин Web -проекта ще не гарантує високу продуктивність усього Web-проекта і зручність роботи з ним користувачів.

Програмний код виконується тільки у браузері

Програмні сценарії JavaScript, JScript або VBScript розміщуються в коді html -сторінок і призначені для реалізації різні динамічних ефектів (зміна зображень, анімація, управління кольором та ін.) і допоміжних функції (перевірка правильності заповнення полів форми та ін.).

До переваг цього варіанту в порівнянні з серверним відносяться:

- Простота створення типових сценаріїв, оскільки для цього, в загальному випадку, не вимагаються знання в області програмування, на відміну від побудови серверних скриптів;
- Швидкість виконання простих функцій, оскільки в даному випадку не відбувається звернення до Web -серверу, що істотно розвантажує не лише його, але і канал передачі даних.

Ще раз підкреслю, що приведені вище перевага даного варіанту відносяться тільки до простих сценаріїв. Складні сценарії швидше виконуються на сервері. Порівняйте пошук товару в каталозі тільки по його назві, або пошук товару в каталозі по назві, ціні, товарній категорії і тому подібне, що ефективніше здійснюється за допомогою SQL -запросов до бази даних. Незважаючи на це, за допомогою сценаріїв можна реалізувати оперативне управління невеликим Web -проектом і навіть інтернет-магазином з декількома тисячами записів (на практиці до 2000-3000, залежно від загального розміру бази даних).

Основним недоліком цього варіанту є неможливість роботи з великими базами даних і виконання більш менш складних сценаріїв.

Програмний код виконується тільки на Web -сервері

В цьому випадку Web -ресурс посилає запити на сервер для виконання певних дій, наприклад, формування вибірки даних за заданими критеріями або збереження вибраних покупцем товарів в кошику замовлення, розрахунок вартості усього замовлення та ін.

Цей варіант дозволяє працювати з великими базами даних і здійснити серверне управління Web -додатком, проте в чистому вигляді його застосування украй обмежене: для виконання більшості запитів потрібно час на їх передачу Web -серверу, обробку і повернення результатів (останнє у ряді випадків може бути відсутнім). Це призводить до великих втрат часу на очікування відповіді, істотно навантажує Web -сервер і канал передачі даних. Усе це, кінець кінцем, призводить до різкого зниження ефективності роботи такої системи і зручності роботи користувачів з нею.

Програмний код виконується як у браузері, так і на Web -сервері

Цей варіант є найбільш практичним як з точки зору роботи Web -системи, так і з точки зору зручності роботи з нею користувачів, оскільки тут можна провести тонкий розподіл функцій (і навантаження відповідно) між браузером і Web -сервером. Як приклад часто приводять механізм відправки форми, коли на клієнтові здійснюється попередня перевірка правильності заповнення полів форми, а сервер здійснює вже тоншу перевірку і обробку отриманих даних. Такий розподіл функцій дозволяє здійснити оперативне управління великими об'ємами даних, що являється в наше "ділове століття" одного з основних завдань при побудові швидких і якісних Web -додатків.

Як видно з розглянутих варіантів, оптимізація Web -додатку полягає не лише у виборі ефективних програмних алгоритмів, але і від розподілу функцій в Web -системе. Це особливо видно на прикладі другого варіанту, коли максимально можлива оптимізація коду на сервері все одно не дозволить добитися того ефекту, який можна отримати при використанні

навіть не оптимізованих функцій на JavaScript. Тому треба чітко розуміти відмінність між оптимізацією схеми побудови Web -додатку і оптимізацією програмних блоків коду.

Якщо подивитися на початковий код деяких Windows -додатків, то неважко помітити, що критичні за часом виконання фрагменти коду переписуються на мовах CI, assembler та ін., тобто збільшення продуктивності досягається за рахунок використання ефективніших мов програмування. У Web такий спосіб оптимізації можна використати тільки на серверній стороні, тому істотно збільшити швидкість роботи Web -додатку у браузері можна тільки за рахунок вибору такою, що відповідає завданням схеми його побудови, що вимагає постійного пошуку нестандартного використання різних програмних об'єктів.

Давайте узагальнимо отриману інформацію і проведемо порівняння можливостей по оптимізації різних Web -проектів за допомогою таблиці 3.1.

Таблиці 3.1. Порівняння можливостей оптимізації різних Web-проектів

Питання	**статична html -сторінка**	**динамічна html -сторінка**	**динамічний Web -додаток**
Елементи для оптимізації	1. Текст 2. Графіка 3. HTML – розмітка	1. Текст 2. Графіка 3. HTML -розмітка 4. Програмний код для браузеру	1. Схема побудови 2. Програмний код для браузеру і Web -сервера 3. Текст 4. Графіка 5. HTML -розмітка
Рівень оптимізації	Кількісний	Кількісний - основний якісний - допоміжний	Якісний - основний кількісний - допоміжний
Елементи сторінки	1. Інформація 2. HTML -разметка 3. Допоміжна інформація	1. Інформація 2. HTML -разметка 3. Допоміжна інформація 4. Програмний код	1. Інформація 2. HTML -разметка 3. Допоміжна інформація 4. Програмний код
Технічна мета оптимізації	Зменшення розміру сторінки	Зменшення розміру сторінки і збільшення ефективності роботи програмних функцій	Збільшення ефективності роботи Web -додатку
Практична мета оптимізації	Збільшення зручності роботи з ресурсом		

Рівень складності реалізації оптимізації	простий	середній	складний і дуже складний
Навички програмування для виконання оптимізації	не вимагаються	вимагаються	потрібно глибокі спеціальні знання

З таблиці видно, що оптимізація статичних html -сторінок здійснюється в основному на кількісному рівні, тобто за рахунок зменшення розміру html - сторінок. Провести оптимізацію на якісному рівні можна тільки при використанні програмних рішень, використовуваних при побудові динамічних сторінок і Web -додатків.

Отриманий від статичної оптимізації ефект призводить до збільшення швидкості завантаження окремих сторінок застосування, а ефект від динамічної оптимізації йде на збільшення функціональності застосування.

Тема 3.2. Просування сайту та пошукова оптимізація

3.1.1 Методи просування і оптимізації сайту

SEO - це абревіатура фрази «Search Engine Optimization», що означає: «Пошукове Просування і Оптимізація».

Пошукова оптимізація сайту припускає велику кількість текстів на сайті, що містять ключові запити. Наприклад, якщо сайт будівельної тематики (продаж будматеріалів), то, грубо кажучи, йому потрібно 10 текстів по 1500 знаків, розбитих на 4 абзаци кожен, ключів типу, що містять по 8-10 входжень, «продаж будматеріалів», «будматеріали оптом», «дешеві будматеріали». Входження ключа - вживання ключового слова в сегменті тексту. Зазвичай ключі розташовують чимдалі один від одного, щоб їх щільність в тексті була оптимальною. Також потрібний текст на головну сторінку сайту - 2-3 тисячі знаків, що містить ключові запити по максимуму, тобто в повному об'ємі. Ключові запити бувають низькочастотними, середньочастотними або високочастотними. Окрім подібних статей, що містять велику кількість входжень ключів (т.з. сео-контент), для пошукової оптимізації потрібні статті, що рекламують сайт, створюють позитивний імідж (вони до сео-контенту не відносяться). Сео-контент буде мертвим вантажем без «живих» рекламно-інформаційних текстів.

Пошукова оптимізація припускає створення посилань між сторінками сайту для активного переходу, також вона зачіпає навігацію сайту (створення вдалого інтерфейсу, спонукаючого відвідувача активно досліджувати сайт) плюс впливає на внутрішній код сторінок (html -код, тобто набір тегів і слів і php -код, тобто скриптовый мова програмування).

Завдання пошукової оптимізації полягає у формуванні такого ядра запитів (ключів), які були б однаково зручні для даної пошуковою системи (обробка запитів пошуковими машинами) і користувачів інтернет, які ці запити вводять.

Будь-який сайт містить ключі першого рівня і ключі другого рівня - завдання пошукової оптимізації полягає ще і в тому, щоб вивести сайт в лідери в рядках запитів, як для перших, так і для других.

Отже, пошукова оптимізація сайту (Search Engine Optimization, SEO) - це робота по його розкручуванню, виведенню на перші позиції в рядках пошукових запитів, вдосконалення з цією метою його архітектури.

Робота по пошуковій оптимізації сайту

Перед початком роботи вам слід врахувати, під яку пошукову систему ви формуватимете ключові запити - Google, Yandex, Rambler або якийсь інший. Успішна пошукова оптимізація сайту дозволить вам при найменших фінансових витратах отримати максимальний приплив клієнтів на ваш сайт. Більшість недосвідчених власників сайтів прибігають тільки до методу наповнення текстів сайту ключовими запитами, - нині з'явилася безліч

методів куди дієвішої роботи по просуванню сайту.

На самому початку формується т.з. семантичне ядро сайту. Семантичне ядро - це перелік пошукових запитів (пошукові запити і переходи на ваш сайт фіксуються такими незалежними системами інтернет-статистики як LiveInternet, Google Analytics). Робота користувача в інтернет здійснюється через пошукові запити, які він вводить рядок пошуку. Наприклад, «Христофор Колумб» або «хитрощі імені Олександра Петренка». Те, що користувач виходить на той або інший сайт - не випадкове попадання. Розташування посилань, що з'явилися, - результат строгої закономірності. Для того, щоб не помилитися з ключовими запитами, можна досліджувати ядро запитів пошукової системи і проаналізувати на наявність ключових входжень сайти ваших конкурентів, якщо ви просуваєте якийсь товар або послугу. Статистика по запитах пошукової системи надається спеціальними обчислювальними програмами. Результат цієї роботи такий: у вас з'являється ядро запитів, спираючись на які ви здійснюєте подальше розкручування сайту (якщо йдеться про сайт з продажу будматеріалів, то конкурентів у вас буде дуже багато і поборотися хоч би за 7-10 позицій в рядках пошукової системи - важке завдання).

Стежте за тим, щоб пошукова оптимізація сайту проводилася «білими», тобто законними методами, інакше ви ризикуєте потрапити під санкції пошукової системи.

Методи просування сайту

Більшість великих компаній по просуванню сайтів пропонують декілька способів оптимізації сайту. Основні наступні: формування ядра ключових запитів, сео-аудит, написання сео-ориентированных текстів, технічний аудит і внесення додаткових змін.

Що таке сео-аудит? Це дослідження сайту по відповідності критеріям ранжирування пошукових систем (тобто виявлення причин, по яких оптимізація сайту не просувається успішно). Одночасно перевіряється індексація сайту в пошукових системах (тобто те, як результати пошукових запитів, ключів сайту відображаються в цій пошуковій системі). Сео-аудит припускає повне вивчення ключових запитів цієї пошукової системи, тобто запитів, необхідних конкретно вам. Сео-аудит детально досліджує і навігацію сайту, зручність інтерфейсу для клієнтів. Для цієї пошукової системи досліджуються його фільтри, тобто механізми, що перешкоджають просуванню конкретних запитів, - ці «підводні камені» треба уникати. Аналогічно можна сказати і про спам. Якщо узагальнити, то робота сео-аудита зводиться до наступного: перевірка на наявність сайту і ключових запитів сайту в пошуковій системі, перевірка помилок в html - і php -коде, перевірка сайту на «чорні», тобто заборонені методи просування (ніби спам-рассылок, забороненої реклами), проводиться оцінка того, наскільки унікальний контент, тобто текстова наповненість сайту (аж до 98-100%, використовуючи такі програми як Etxt Антиплагіат і Advego Plagiatus).

Технічний аудит відрізняється від сео-аудита напрямом діяльності. Технічний аудит досліджує не наповненість тексту ключами, а гіпертекст, тобто правильність html -кода, досліджується робота движка сайту (1С-Битрикс, NetCat та ін.), вивчаються і коригуються заголовки на сайті.

Подібні методи - сео-аудит і технічний аудит - можуть здійснювати тільки спеціальні компанії, працюючі з пошуковою оптимізацією сайтів. Пункт під назвою «внесення додаткових змін» припускає роботу із вже «відредагованим» сайтом - на цій стадії перевіряється його відповідність вибраній пошуковій системі.

Просування по словах і по трафіку

Розкручування сайту «по словах» - це виведення сайту на перші місця в результатах пошукових запитів. Т. е. той самий сео-контент, ті самі входження ключа, про які говорилося вище. Іноді цей метод не спрацьовує. Наприклад, у тому випадку, якщо ви власник футбольного сайту, освітлюючого події футбольної Європи. Притягнути відвідувачів по ключових запитах буде дуже важко - маса футбольних сайтів-конкурентів містить приблизно однакові ключові запити. В цьому випадку вам доведеться звернутися до просування сайту по трафіку (якщо ви власник інтернет-магазину, то тільки це вас і зацікавить).

Припустимо, у вас є перспективний інтернет-проект і ви шукаєте способи його розкручування? SEO -оптимизация у наш час - справа непроста і відповідальна і, звичайно ж, для виконання цього завдання можна спробувати знайти талановитого самоучку-фрилансера або адекватного штатного оптимізатора. Але досвід показує, що найбільш надійним є варіант, коли клієнт звертається в перевірену SEO -компанию. Різнобічна експертиза, знання останніх тенденцій галузі і відповідальний підхід окупляться сповна.

Отже, ми визначилися з виконавцем, тепер подумаємо про те, як оцінювати і оплачувати отриманий результат. По суті, на сьогодні існує всього два варіанти:

Просування за ключовими словами - платимо за дні знаходження запиту в Топі пошукових систем.

Просування сайту по трафіку - плата за кожного притягненого відвідувача на сайт.

В Україні перший варіант спочатку завоював набагато більшу популярність, багато в чому завдяки своїй простоті. При вибраних ключових запитах навіть школяр зуміє зняти поточні позиції і виставити рахунок. Та і турбот менше - вивів запити в ТОП і формально свої зобов'язання виконав, а надання чогось більшого вже залежить від сумлінності оптимізатора. В цей час клієнт може справно оплачувати рахунки, з трепетом стежити за коливаннями позицій найбільш високочастотного запиту і не підозрювати, що він упускає потенційних відвідувачів, які просто набрали не той ключовий запит.

Переваги трафікового просування

По суті, що найважливіше для власника сайту, що продає? Щоб були

покупці. Трафікове просування має величезну перевагу в тому, що прив'язка до невеликого набору ключових слів відсутня. Можна комплексно задіяти просування величезного пулу низькочастотних запитів, можна притягати трафік просуванням в соціальних мережах, у вертикальному пошуку по картинках, відео, і так далі. Фокус оптимізатора зміщується з позицій запитів до того, щоб перетворити користувачів на відвідувачів, а власникові сайту залишається лише перетворити відвідувачів на покупців.

Для клієнтів просування по трафіку немає сенсу хвилюватися із-за падіння окремих запитів, тому що трафік високочастотного запиту, що вилетів з Топа, може з легкістю компенсувати сотня низькочастотних запитів, що піднялися. Апдейти пошукових систем вже не так серйозно міняють відвідуваність, і завдяки просуванню по трафіку результат набагато стабільніший.

У оптимізатора хороша мотивація постійно допрацьовувати сайт, підбирати нові відповідні ключові слова, тим самим збільшуючи трафік - а додатковий трафік, у свою чергу, обіцяє додатковий доход. Причому доход отримує як клієнт, так і оптимізатор.

Трафікове просування ідеально підходить для інтернет-магазинів, оскільки у них постійно міняється товарна номенклатура: деяких товарів тимчасово немає в наявності, деякі моделі застарівають. Для просування по трафіку ця ситуація не є проблемою, оскільки ключові запити гнучко адаптуються під каталог, що змінився, і гроші не будуть витрачені даремно.

Підводні камені просування сайту по трафіку.

При усіх очевидних перевагах, у трафікового просування є декілька вузьких місць, а саме:

- Висока трудомісткість. Для того, щоб забезпечити якісне просування по трафіку, SEO - компаніям необхідно розробити систему підбору ключових слів, збору позицій по тисячах запитів, технологію автоматизованої внутрішньої оптимізації і визначення цільової сторінки. Також потрібно високий рівень професіоналізму оптимізатора, який повинен мати навички комплексного просування сайту. Тому, необхідно з усією відповідальністю підійти до вибору підрядника.
- Нецільовий трафік. Відвідувачі на сайті можуть бути цільові і нецільові. Буває що, притягаючи величезну кількість трафіку на свій ресурс, власник сайту практично не отримує покупців. Студенти навряд чи купуватимуть складське устаткування, а домогосподарки замовляти автозапчастині. Для розважальних порталів, новинних сайтів цільовим є практично увесь трафік, для інтернет-магазинів цільовим є той, який пов'язаний з товарними найменуваннями. Чи платити за увесь трафік? Це залежить від домовленості з організацією, що надає послуги з просування сайту. Деякі компанії пропонують оплачувати увесь трафік, але за меншою ціною за відвідувача, інші - тільки цільовий, але тоді вартість притягненого відвідувача стає більше. Рішення завжди залишається за клієнтом.
- Сума до оплати. Ні клієнт, ні SEO -компания не знають заздалегідь, яка сума буде нарахована у кінці місяця і цілком вірогідні сюрпризи. Для

того, щоб внести більше визначеності, в договір іноді вносяться умови, що регламентують максимальну суму платежу трафікового просування, яка з періодичністю в 3-4 місяці переглядається.

Як уникнути санкцій пошукових систем при пошуковій оптимізації?

Прийнявши рішення про просування сайту в пошукових системах, необхідно замислитися про вибір безпечних методів досягнення мети. Це дуже важливо, адже невміла оптимізація може привести до негативних наслідків. Пошукові системи постійно відстежують сайти, що користуються забороненими методами, і застосовують санкції. Якщо Ваш сайт потрапить в чорний список пошукових систем, то він буде знижений у видачі (тобто втратить позиції) або може бути взагалі виключений з результатів пошуку.

Як же уникнути санкцій пошукових систем? Відповідь проста: не використати заборонені методи оптимізації. Тому якщо Вам обіцяють просунути сайт в ТОП-10 за дуже короткий термін або дуже дешево, то можете бути упевнені, що працювати оптимізатори будуть «чорними» методами.

Якщо продажі через інтернет важливі для клієнтів компанії, то не потрібно використати небезпечні методи просування, такі як:

- Приховані (чи важко читані) тексти, у тому числі тексти у вікнах прокрутки. Недобросовісні оптимізатори розміщують на сайтах, що просуваються, текст, який невидно користувачеві, але добре сприймається пошуковим роботом. Для цього текст маскується, наприклад, за рахунок однакового кольору з фоном, може бути набраний дуже маленьким шрифтом, розташований під картинкою і так далі
- Клоакинг. Це показ роботам пошукових систем зміненого контенту сайту. Т. е. користувачеві надається одна версія сайту, а роботові - спеціально для нього створені сторінки.
- Системи автоматичного обміну посиланнями (статтями, новинами). На сайтах розміщуються приховані посилання, які дають тимчасовий приріст позицій.
- Різке, разове нарощування кількості посилань. Для того, щоб отримати результат в максимально оперативні терміни, недобросовісні оптимізатори купують відразу велику кількість посилань. Різке нарощування посилальної маси - неприродний процес для сайту, тому пошукові системи досить швидко відстежують це і знижують позиції.
- Розміщення зовнішніх посилань на сайті. На сайтах клієнтів розміщуються посилання на інші сайти.
- Недобросовісні методи просування забезпечують тимчасовий результат. Пам'ятайте, що навіть якщо пошукові системи не виявлять «чорну» оптимізацію в автоматичному режимі, то конкуренти не упустять шанс і «поскаржаться» на Ваш сайт. Краще не ризикувати і звернутися в професійну компанію.

Оптимізація сайту : що далі

Посилання на ваш сайт мають бути присутніми на ретельно вибраних плацдармах в мережі інтернет. Наприклад, якщо ваш сайт - сайт внз, то посилання на нього бажано розмістити на інших учбових ресурсах, на інших сайтах внз, на будь-яких навчально-методичних сайтах. На сайті бажано розміщувати «рекламу у відповідь», тобто за домовленістю рекламувати інший сайт у відповідь на те, що прорекламували вас. Просування через групи в соціальних мережах зараз набирає оберти. Тим більше це не вимагає ніяких грошових вкладень - ви самі можете вести групу в соц. мережі «Вконтакте», присвячену вашій компанії, також ви можете розміщувати безкоштовні оголошення про вашу компанію в «Вконтакте» або на дошках безкоштовних оголошень. Можна створити групу вашого сайту і в мережі «Фейсбук», але очевидно, що вона матиме меншу популярність порівняно з групою в мережі «Вконтакте» (із-за не розробленості «Фейсбук» в країнах СНД, правда, з появою «Фейсбук» на росіянинові ця проблема поступово стала вирішуватися). Цікавими майданчиками в даному випадку є також Myspace і Livejournal.

Якщо ви ведете пошукову оптимізацію сайту своїми силами, то правильно їх розрахуйте. Вам також бажано володіти мовами програмування (PHP, MySQL, Java, мовою гіпертексту html ви повинні володіти досконало), представляти, що таке робота з движком сайту, формувати самостійно сео-тексты і щільність ключів, причому підбирати ключі, що задовольняють цій пошуковій системі (а їх набір постійно міняється). Для розміщення контекстної реклами, тобто створення в інтернеті майданчиків з посиланнями на ваш сайт, краще всього звернутися по допомогу до спеціальних агентств, цим розміщенням, що займається. Зараз 25 мільйонів росіян є регулярними користувачами інтернету, тому найефективніша реклама на просторах СНД - це реклама в інтернеті. Існує багато майданчиків для розміщення новин, статей, посилань - біржа Sape, наприклад.

3.1.2 Показники SEO

SEO - це дії, спрямовані на підвищення позицій сайту у видачі пошукових систем, що веде до збільшення відвідуваності з пошукових систем.

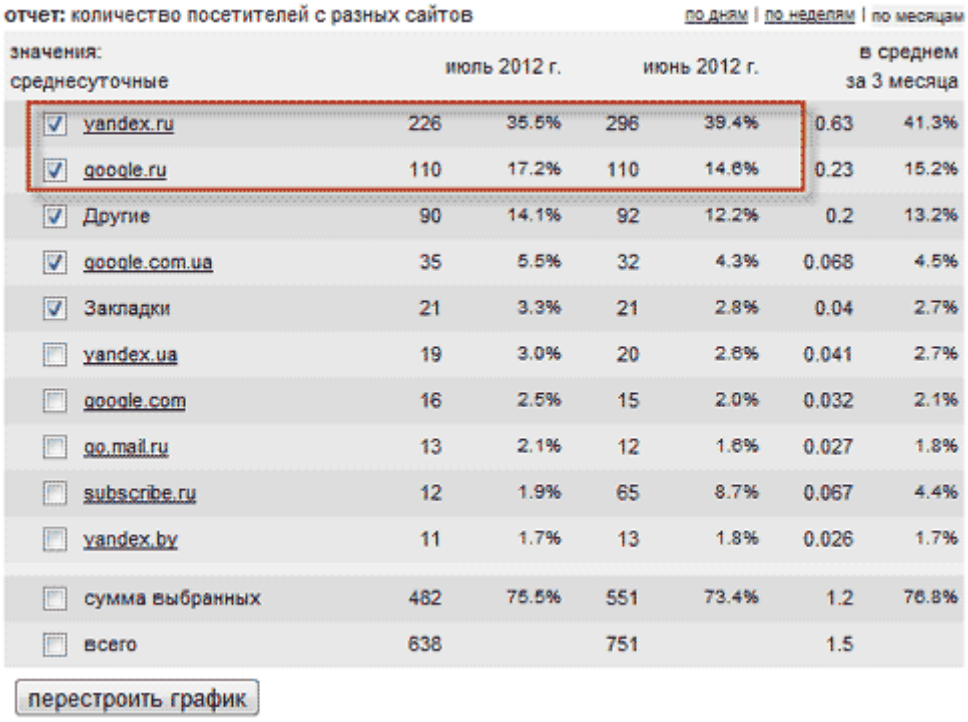
До SEO можна також віднести роботи над сайтом, пов'язані з підвищенням таких показників, як:

- тематичний індекс цитування Яндекса (ТІЦ) — технологія пошукової машини «Яндекс», що полягає у визначенні авторитетності інтернет-ресурсів з урахуванням якісної характеристики — посилань на них з інших сайтів. ТИЦ розраховується по спеціально розробленому алгоритму, в якому особливе значення надається тематичній близькості ресурсу і сайтів, що посилаються на нього. Цей показник в першу чергу використовується для визначення порядку розташування ресурсів в рубриках каталогу «Яндекса».

Усі сайти, що посилаються, мають бути обов'язково проиндексированы Яндексом.

- Page Rank Google (PR) — це числова величина, що характеризує «важливість» веб— сторінок. Чим більше посилань на сторінку, тим вона «важливіша». Крім того, «вага» сторінки А визначається вагою посилання, що передається сторінкою В. Таким чином, PageRank — це метод обчислення ваги сторінки шляхом підрахунку важливості посилань на неї.
- траст сайтатраст сайту (довіра, Trust) - рівень довіри з боку пошукової системи до цього сайту.

На сьогодні пошукові системи - це практично самі відвідувані сайти у світі. Пошукові системи створені спеціально для пошуку необхідної інформації на різних сайтах з подальшим наданням релевантних сторінок людям, які шукають інформацію. Саме пошукові системи сьогодні приносять найбільший трафік більшості сайтів (рисунок 3.1)



отчет: количество посетителей с разных сайтов

по дням | по неделям | по месяцам

значения: среднесуточные	июль 2012 г.		июнь 2012 г.		в среднем за 3 месяца	
yandex.ru	226	35.5%	296	39.4%	0.63	41.3%
google.ru	110	17.2%	110	14.6%	0.23	15.2%
Другие	90	14.1%	92	12.2%	0.2	13.2%
google.com.ua	35	5.5%	32	4.3%	0.068	4.5%
Закладки	21	3.3%	21	2.8%	0.04	2.7%
yandex.ua	19	3.0%	20	2.6%	0.041	2.7%
google.com	16	2.5%	15	2.0%	0.032	2.1%
go.mail.ru	13	2.1%	12	1.6%	0.027	1.8%
subscribe.ru	12	1.9%	65	8.7%	0.067	4.4%
yandex.by	11	1.7%	13	1.8%	0.026	1.7%
сумма выбранных	482	76.5%	551	73.4%	1.2	76.8%
всего	638		751		1.5	

перестроить график

Рисунок 3.1 Оцінка трафіку сайтів

Для кращого розуміння необхідності пошукової оптимізації, звернемося до пошукової видачі (SERP) Яндекса (Рисунок 3.2) .

За запитом «як розкрутити сайт самостійно», на першій сторінці SERP Яндекса ми спостерігаємо:

— рекламні оголошення Яндекс.Директ (у самому верху і справа);

— ТОП-10 сайтів.

Приблизно 60-70% трафіку по цьому запиту доводиться на сайти, розташовані в ТОП-3 пошукової видачі. Інша доля переходів доводиться на контекстну рекламу (туди можна потрапити тільки заплативши Яндексу за розміщення реклами Яндекс.Директ), ну і трохи переходів отримують інші сайти.

Рисунок 3.2 SERP пошукової системи Яндекс

Види SEO

Search Engine Optimization прийнято ділити на два види.

- Внутрішня пошукова оптимізація
- Зовнішня пошукова оптимізація

Внутрішня пошукова оптимізація - це дії, спрямовані на поліпшення свого сайту, поліпшення чинників ранжирування, які роблять безпосередній вплив на позиції в пошуковій видачі.

Внутрішня пошукова оптимізація - це сама трудомістка робота, проте, цю роботу необхідно проводити постійно, вона дуже важлива. Саме від

внутрішньої оптимізації в цілому залежить успіх Вашого проекту.

До внутрішньої оптимізації можна віднести наступні заходи:

- складання семантичного ядра сайту (підбір ключових слів, по яких планується просування ресурсу);
- робота над внутрішньою структурою ресурсу (Robots.txt, карта сайту);
- усунення технічних помилок (дублі сторінок, биті посилання, прискорення сайту);
- постійне вдосконалення і поліпшення юзабіліті ресурсу (зручність для користувачів);
- робота над тестом (підбір ключових слів, релевантность сторінок, оптимізація зображень);
- інші заходи, спрямовані на зручність роботи з сайтом (наприклад, дизайн, форматування тексту, перевірка орфографії тексту і тому подібне).

Сьогодні усі пошукові системи постійно прагнуть поліпшити якість пошукової видачі, тому внутрішній оптимізації необхідно приділити найпильнішу увагу.

Ваш сайт має бути якісним, швидким, зручним для користувачів і повною мірою надавати користувачам ту інформацію, за якою вони до Вас прийшли.

Інакше про високу відвідуваність з пошукових систем не може бути мови.

Зовнішня оптимізація це, в основному, дії, спрямовані на отримання посилальної маси на сайт. Є ще і інші заходи, наприклад аналіз сайтів конкурентів.

Кілька років тому пошукова оптимізація сайту будувалася, в основному, на придбанні посилань з необхідними анкорами, в яких містилися ключові слова, по яких просувалася сторінка. Оптимізатору треба було закупити посилань більше ніж у конкурента і з авторитетніших сайтів і ресурс, що просувається, опинявся в топі.

Сьогодні ці методи вже не працюють, а посилання вже не мають такої магічної сили, як раніше. Нині необхідно більшою мірою займатися внутрішньою оптимізацією, проте забувати про зовнішнє SEO теж не варто.

Посилальну масу можна нарощувати як безкоштовними методами, так і платними способами, наприклад, купуючи посилання у біржах gogetlinks або rotapost. Посилання бувають тимчасовими і вічними.

Новачкам рекомендується розпочати зовнішню оптимізацію з безкоштовних методів, наприклад, реєстрація в профілях трастових сайтів, анонси статей на інших ресурсах, обмін постовими, гостьові пости і іншими способами отримання посилальної маси.

Окрім росту позицій у видачі пошукових систем і збільшення відвідуваності, посилання допомагають збільшити такі показники, як тИЦ і PR, а також траст сайту.

Посилальну масу необхідно нарощувати поступово, не допускаючи різкого збільшення кількості посилань.

У SEO існують такі поняття, як природні і неприродні (SEO) посилання, які можуть визначити пошукові системи, тому, слід дуже ретельно підходити до питання придбання посилань. Перш ніж купити посилання, необхідно ретельно проаналізувати сайт-донор.

Контрольні питання до розділу 3

1. Які задачі вирішує оптимізація веб-проекту?
2. Порівняйте два основні способи оптимізації веб-додатків.
3. Які пункти передбачає оптимізація HTML-сторінок?
4. Суть оптимізації статичного веб–додатку.
5. Заходи з оптимізації динамічних html -сторінок .
6. Що включено в поняття SEO?
7. Які основні завдання пошукової оптимізації?
8. Що таке семантичне ядро сайту?
9. Які є найвідоміші методи просування сайту?
10. Назвіть переваги і недоліки трафікового просування.
11. Які способи просування не рекомендується використовувати і чому?
12. Назвіть відомі показники SEO.
13. Які є основні види SEO?
14. В яких випадках рекомендують застосовувати внутрішню оптимізацію?
15. Коли доцільніше використати зовнішню пошукову оптимізацію?
16. Яка роль пошукових серверів у «розкрутці» сайту?

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

1. Юринець В.Є. Комп'ютерні мережі. Інтернет: навч. посіб. Львів: ВЦ ЛНУ, 2006.

2. Возняк, Л.С. Комп'ютерний практикум. Формування навичок роботи із сервісами мережі Інтернет. І.-Ф.: ВДВ ЦІТ, 2006.

3. Глинський, Я. М. Інтернет. Сервіси, HTML і web-дизайн: Навч. посіб.- 3-є вид. Львів : Деол, СПД Глинський, 2005.

4. Татенбаум Э. Компьютерные сети. СПб. : Питер, 2008.

5. Глушаков, С.В. Работа в сети Internet. Фолио, 2002.

6. Иванов В. Интернет для начинающих. Самоучитель. К.: Питер; BHV, 2005.

7. Самсонов В. В. Методи та засоби Інтернет-технологій. Харків: Компанія СМІТ, 2008.

8. Юринець В.Є. Комп'ютерний практикум. Формування навичок роботи із сервісами мережі Інтернет. Львів: ВЦ ЛНУ, 2006

9. Гаевский А.Ю. Самоучитель по созданию Web-сторінок HTML JavaScript Dynamic HTML.К.: А.С.К., 2002.

10. Матвієнко О.В. Internet-технології: проектування Web-сторінки: Навч.посібник. К. : ЦУЛ, 2004.

ІНФОРМАЦІЙНІ РЕСУРСИ

1. Архитектура web-додатків. [Електронний ресурс]. – Режим доступу: http://edu.dvgups.ru/metdoc/GDTRAN/YAT/ITIS/INT_PR/METOD/MU_KR/frame/1.htm

2. Введение в стандарты Web. [Електронний ресурс]. – Режим доступу: http://www.intuit.ru/department/internet/operawebst/

3. Практикум по программированию на JavaScript. [Електронний ресурс]. – Режим доступу: http://www.intuit.ru/studies/courses/119/119/info

4. Основы программирования на JavaScript. [Електронний ресурс]. – Режим доступу: http://www.intuit.ru/studies/courses/1093/132/info

5. Язык программирования PHP. [Електронний ресурс]. – Режим доступу: http://www.intuit.ru/studies/courses/42/42/info

6. Введение в программирование на PHP5. [Електронний ресурс]. – Режим доступу: http://www.intuit.ru/studies/courses/1025/166/info

7. Поисковая оптимизация. [Електронний ресурс]. – Режим доступу: http://ru.wikipedia.org/wiki/Поисковая_оптимизация