# Зміст

***1.Основи мови ДжаваСкріпт. Типи даних. Сталі. Ідентифікатори. Змінні. Вирази. Операції. Підключення коду.***

Для **включения JavaScript-сценариев** в HTML-документы существует несколько способов. Можно определить встроенный сценарий, содержимое которого является частью документа. Также можно определить внешний сценарий, JavaScript-код которого хранится в отдельном файле, тогда как HTML-документ лишь ссылается на него посредством URL-адреса

```
<head>
<title>ripnMep</title>
 <script type="text/javascript"> , console.log("Привет");
</script>
</head>
```

У сценаріях JavaScript можна використовувати змінні, звертаючись до них за назвою. Змінні можуть бути глобальними або локальними. Глобальні змінні досяжні з довільного місця сценарію. Область дії локальних змінних обмежено кодом функції, всередині якого оголошено ці змінні. Мова JavaScript містить шість типів даних: Undefined (непроникний), Null (нульовий), Boolean (логічний), String (рядковий), Number (числовий) і Object (об'єктний). Це відносно незначна кількість типів дозволяє, тим не менше, створювати повноцінні сценарії для виконання багатьох функцій.

У Javascript є і об'єктні типи даних і елементарні, які можна інтерпретувати як об'єкти.

**Ідентифікатор**

Послідовність символів в коді, яка у недвозначний спосіб позначає
(ідентифікує) змінну, функцію або властивість, називається ідентифікатором.

У JavaScript, ідентифікатори можуть містити лише алфавітно-цифрові символи (а також "$" або "_"), і не можуть починатися з цифри. Ідентифікатор відрізняється від рядків тим, що рядок являє собою дані, натомість ідентифікатор є частиною коду. В JavaScript, немає можливості перетворити ідентифікатори в рядки, але іноді є можливість перетворитирядки у ідентифікатори.

**Змінні**

Назва змінної:

має починатися начинаться з літери латиниці або символів "_" чи "$", містити лише літери латиниці, цифри і символи "_" та "$";

не може збігатися з жодним з таких зарезервованих слів JavaScript:

```
break     case      catch    class     const    continue
debugger  default   delete   do        else     enum
export    extends   false    finally   for      function
if        import    in       new       null     return
super     switch    this     throw     true     try
typeof    var       void     while     with
```

Надання змінним величини здійснюють за допомогою оператора "=".

Var Hello; Hello = “privet”;
При цьому у довільному місці програми змінній Hello можна надати числельну величину, наприклад, так:
Hello=4;
Після здійснення такої операції тип змінної буде змінено, але у процесі інтерпретації сценарію не буде

зроблено жодного попереджувального повідомлення. Змінній можна надати спеціальну величину null. У цьому випадку змінній не призначено жоден з типів.

**Регулярні вирази** це шаблони, що використовуються для пошуку збігу,співпадіння в тексті чи строках. В JavaScript регулярні вирази також є об’єктами. Ці шаблони використовуются для пошуку збігу у тексті.

Ви можете створювати регулярні вирази двома способами:

Використовуючи літерал регулярного виразу, який складається з шаблону між символами "/" слеш:

Var re = /ac+c/;

Цей спосіб підійде для статичних, простих виразів, використовуйте його для покращення продуктивності.

Або через створення об’єкта RegExp:

var re = new RegExp("ab+c");

В такому випадку створюється об'єкт з методами exec та test класу RegExp, та методами з String классу, а саме match, replace, search, та split.

Використовуйте цей спосіб у випадках, коли вам відомо, що шаблон регулярного виразу буде змінюватись. Наприклад для очікування дій користувача з подальшим виконанням якоїсь функції тощо, або для побудови URL маршутів.

**Арифметичні операції** беруть в якості операндів числові значення і повертають результат у вигляді одного числового значення. Стандартними арифметичними операціями є додавання (+), віднімання (-), множення (*) і ділення (/).

Також є такі операції як: Остача від ділення (%), Зведення в ступінь (**), Інкремент (++), Декремент (-), Унарний мінус (-), Унарний плюс (+)

*2.Оператори умови (if switch) та циклічні (while, for, …)*

**if ... else - умовний оператор.**

if(умова)
{ блок if }
[else { блок else }]
Параметри:
умова - умова яка перевіряється. Вираз типу Boolen.
блок if - блок коду який виконується якщо умова дорівнює true.
блок else - не обов'язковий блок. Виконується.у випадку якщо умова дорівнює false.
Опис:
Умовний оператор if використовується коли необхідно виконати певний блок коду якщо виконується певна умова.
Умова виконується якщо сама умова дорівнює true (логічний тип даних Boolean), тоді виконується блок if в іншому випадку виконується блок else так як умова дорівнює false.

```
if(true){alert('істина');}
else{alert('не істина (хиба)');}
```

Список значень які являються хибою(false, не істиною.)
False,undefined,NaN,0,null,порожній рядок ("")
Все інше рівняється як true.

**switch - умовний оператор, оператор вибору.**

Синтаксис:

```
switch (значення) {
case ідентифікатор:
 дія
break;
case ідентифікатор :
 дія
break;
default :
 дія
break;
}
```

Параметри:
значення - значення за яким виконується вибір.
ідентифікатор - ідентифікатор з яким порівнюється саме значення.
дія - блок коду(оператори) який виконується якщо значення спів падає з ідентифікатором.
Опис:
switch оператор вибірки. Виконує пошук case з потрібним ідентифікатором (міткою). Якщо потрібний ідентифікатор знайдений виконується код до оператора break.
Оператор break перериває виконання коду. У випадку якщо оператора break немає код виконується до наступного операторa break або до кінця оператора switch.
Цю можливість можна використовувати якщо потрібно один і такий код при кількох ідентифікаторах або до наступного оператора break.
default - використовується якщо потрібно виконати певний код коли потрібного ідентифікатора не знайдено.
У JavaScript не має можливості у switch поставити ідентифікатор типу від 1 до 10 (1..10) тощо. Таку можливість можна досягнути якщо для значення передати true а умови ставити у ідентифікаторі. Тоді switch буде шукати істину умову:

**while**

while - цикл який виконується доки умова дорівнює true.
Синтаксис:

```
while(умова) код
```

Параметри:
умова - умова яка перевіряється.

код - код циклу який виконується.
Опис:
Цикл while виконується доти доки умова дорівнює true.
Спочатку перевіряється умова і якщо умова дорівнює true тоді виконується код циклу, протилежно діє цикл do...while.

**for - цикл, виконує код доки умова дорівнює true.**
Синтаксис:
for(ініціалізація;умова;крок)
Параметри:
ініціалізація - виконує лише один раз, оголошується змінна (змінні) яка необхідна для умови і кроку. Для оголошення використовується var або let.
умова - умова яка перевіряється при кожному циклі. Якщо умова дорівнює true то виконується код і змінна збільшується на вказаний крок. У випадку коли умова дорівнює false цикл зупиняється.
крок - виконується кожного разу при повторному виконанні циклу,вказується на скільки збільшується зміна(яка вказується при ініціалізації) після кожного циклу.
Якщо не вказати параметри циклу то отримаємо вічний цикл for(;;) :

```
for(;;){
 alert('Ой, Ви запустили вічний цикл! :-(');
}
```

Опис:
Цикл використовується якщо необхідно виконати певний код потрібну кількість разів, пройти масив.
Також цикл можна зупинити за допомогою оператора break або перейти до наступного проходу цикла за допомогою оператора continue.
При ініціалізація оголошується змінні за допомогою var або let. Різниця полягає в тому що let оголошує локальну змінну яка доступна лише у області циклу, тоді як за допомогою var змінна доступна за межами циклу.

```
for(var a=1;a<3;a++){
alert('змінна a в циклі: '+a);
}
```

Якщо оголошувати змінні в ініціалізації циклу без var то може виникнути ситуація що коли викликається інший цикл з оголошеною такоюж назвою зміної то значення зміної зміниться і перший цикл перевіривши вказану умову завершиться швидше ніж очікувалося:

***3.Функції мови JS. Об`єкти мови JS***

Конструктор **function** створює новий об'єкт Function. В JavaScript кожна функція є об'єктом Function.
Синтаксис:
function functionName([argname1  [, ...[, argnameN]]])
{
  functionCode
}
functionName = new Function( [argname1,  [... argnameN,]] functionCode);
Синтаксис стрілкової функції:
(argname1, [..., argnameN]) => { functioncode }
Параметри:
FunctionName - Обов'язкове. Ім'я новоствореної функції.
argname1 ... argnameN - Необов'язково. Список аргументів функції які передаються або іншими словами параметри функції. Кожне ім'я повинне мати іншу назву.
functionCode - Необов'язково. Рядок, що містить блок коду JavaScript який буде виконуватися при виконанні функції.
Опис:
Синтаксис є стандартним способом для створення нових функцій в JavaScript. Синтаксис 2 є альтернативною формою використовується для створення об'єктів функцій в явному вигляді. Також появився у ES6 спрощений синтаксис функції - стрілкові функції.
На відмінно від деяких мов програмування(Delphi, Pascal) JavaScript не має процедур а лише функції. Різниця між процедурою і функцією полягає в тому що процедура не повертає результат а функція повертає результат. Функція у JavaScript завжди повертає результат, результатом може бути будь який тип даних. Повертається результат за допомогою оператора return. Якщо під час виконання функції результат не був повернутий тоді JavaScript автоматично повертає undefined.
Також є функції-конструктори і стрілкові функції.

**Конструктор Object** створює оболонку об'єкта.
Синтаксис
// Ініціалізатор об'єктів або літерал
{ [ *nameValuePair1*[, *nameValuePair2*[, ...*nameValuePairN*] ] ] }

// Викликається як конструктор
new Object([*value*])
Параметри
nameValuePair1, nameValuePair2, ... nameValuePair*N*
Пари імен (рядків) і значення (будь-яке значення), де ім'я відділяється від значення двокрапкою.
value
Будь-яке значення.
Опис
Конструктор Object створює оболонку об'єкта для заданного значення. Якщо значення є null або undefined, конструктор створює і повертає порожній об'єкт, в іншому випадку, він створює об'єкт Типу, що відповідає заданному значенню. Якщо значення вже є об'єктом , то конструктор поверне значення.
Коли визваний не в контексті конструктора, Object поводиться ідентично до new Object().

***4.JS.Масиви. Об`єкт String***

Глобальний об'єкт **JavaScript Array** - це конструктор для масивів, які на високому рівні є спископодібними об'єктами.
Синтаксис:
arrayObj = new Array();
arrayObj = new Array([size]);
arrayObj = [];
arrayObj = [element0, ..., elementN];
Параметри:
arrayObj - Ім'я змінної, якій присвоюється об'єкт Array.
size - Необов'язковий параметр. Розмір масиву. Оскільки масиви відраховуються від нуля, створеним елементам присвоюються індекси від нуля до size -1.
element0, ..., elementN - Необов'язковий. Елементи, які потрібно помістити в масив. Створює масив з числом елементів n + 1 і значенням length, рівним n + 1. При використанні даного синтаксису необхідно вказувати більше одного елемента.
Опис:
Масиви є спископодібними об'єктами, чиї прототипи містять методи для операцій обходу змінних масиву. Ні розмір JavaScript-масиву, ні типи його елементів не є фіксованими. Оскільки розмір масиву може збільшуватися і зменшуватися в будь-який час, то немає гарантії, що масив виявиться щільним. Тобто, при роботі з масивом може виникнути ситуація, що елемент масиву, до якого ви звернетеся, буде порожнім і поверне undefined. В цілому, це зручна характеристика але якщо ця особливість масиву не бажана в вашому специфічному випадку, ви можете розглянути можливість використання типізованих масивів.
Для доступу до окремих елементів створеного масиву використовуються квадратні дужки "[]". Масиви в JavaScript індексуються з нуля: перший елемент масиву має індекс, що дорівнює 0, а індекс останнього елемента дорівнює значенню length-1.

**Двохвимірний масив** це масив елементи якого містять значення масив. Звертатися до елементу двохвимірного масиву необхідно за вказанням двох індексів array[index1][index2].
За принципом присвоєнню значенню елементу масиву інший масив у JavaScript можна створювати багатовимірний масив. Це дозволяє створювати масив двохвимірний, трьохвимірний, чотирьохвимірний і т. д.

**String - текстовий рядок.** Рядки у JavaScript це послідовність символів Unicode. Кількість символів може бути від 0 і більше. Символи включають в себе букви, цифри, знаки пунктуації, спеціальні символи і пропуски. Рядки повинні бути поміщені в подвійні або одинарні лапки (апострофи):
var s = "text";
var s2 = 'text 2';
var s3 = String('створюємо строку через String');
var s4 = new String('створюємо строку через String');

Рядки вкладені в подвійні лапки можуть містити символи одинарних лапок і навпаки. Для того, щоб в рядку, укладеному в подвійні лапки, можна було використовувати подвійні лапки, потрібно їх екранувати за допомогою зворотного слеша , теж саме стосується і рядка вкладеного в одинарні лапки:
Екранізувати можна будь який символ, для екранізації самого зворотнього слеша використовують зворотній слеш
У JavaScript рядки можна об'єднувати за допомогою символу плюс "+".
var s= "Привіт", s2="JavaScript";

str=s+" "+s2+" :-)"; // "Привіт JavaScript :-)"

У Javascript не можна змінити рядок, тобто змінити певний символ. Створений рядок є таким назавжди, його лише можна замінити новим рядком.

***5.Регулярні вирази. Розробка сценаріїв пошуку текстової інформації за допомогою регулярних виразів***

Регулярні вирази це шаблони, що використовуються для пошуку збігу,співпадіння в тексті чи строках. В JavaScript регулярні вирази також є об’єктами. Ці шаблони використовуються для пошуку збігу у тексті.
Ви можете створювати регулярні вирази двома способами:
Використовуючи літерал регулярного виразу, який складається з шаблону між символами "/" слеш:
Var re = /ac+c/;
Цей спосіб підійде для статичних, простих виразів, використовуйте його для покращення продуктивності.
Або через створення об’єкта RegExp:
RegExp - об'єкт регулярного виразу для співставлення з шаблонами.
Синтаксис:
new RegExp( pattern [, flags])
Параметри:
pattern - текст регулярного виразу.
flags - не обов'язково. Доступні прапорці:
g - глобальний пошук.
i - ігнорувати регістр символів.
m - багаторядковий пошук.
y - шукає співставлення у властивостях lastIndex регулярного виразу і не шукає в більш пізніших індексах.
u - включає функції Unicode.
Опис:
Регулярний вираз це спеціальні коди для пошуку шаблонів у рядках.
Регулярні вираз забезпечує швидкий і легкий спосіб співставлення рядків в шаблон.
Регулярні вирази застосовують:
для перевірки коректності вводу тексту (email, номер тел. і т.п.).
для заміни одного тексту на інший, для пошуку тексту

За замовчуванням регулярні вирази в JavaScript чутливі до регістру і пошук відбувається тільки до першого співпадіння в будь-якому заданому рядку. Додаючи g (для глобального), і i (для ігнорування регістра) модифікатори після другого /, ви можете зробити пошук по регулярному виразу для всіх збігів в рядку і ігнорувати регістр відповідно. Ось кілька прикладів їх використання. Для кожного варіанту вказано, яка частина рядку «test1 Test2 TEST3» буде відповідати запиту:
/Test[0-9]+/ тільки “Test2”
/Test[0-9]+/i тільки “test1”
/Test[0-9]+/gi “test1”, “Test2” і “TEST3”

Символи ?, + та * також мають особливе значення. Зокрема, ? означає "попередній символ не є обов'язковим", + означає "один або більше з попереднього символу", а * означає "нуль або більше попереднього символу".

```
bana?na     // Дорівнює "banana" і "banna",
            // не дорівнює "banaana".
bana+na     // Дорівнює "banana" і "banaana",
            // не дорівнює "banna".
bana*na     // Дорівнює "banna", "banana", і "banaaana",
            // не дорівнює "bnana".
```

***6.Структура ДОМ. Об`єкти. Основні примітиви. Вбудовані об`єкти. Об`єкт документ***

Об’єктна модель документа (**DOM**) – це API, через який JavaScript взаємодіє з контентом на сайті. JavaScript і DOM зазвичай розглядаються як єдине ціле, оскільки для взаємодії з контентом в Інтернеті найчастіше використовується саме JavaScript. DOM API використовується для доступу, перегляду та маніпулювання HTML і XML документами.
DOM створює ієрархію, що відповідає структурі кожного веб-документа. Ця ієрархія складається з вузлів. Існує кілька різних типів DOM-вузлів, найбільш важливими з яких є «Елемент», «Текст» і «Документ»:
Вузол «Елемент» представляє елемент на сторінці. Таким чином, якщо у вас є елемент абзацу (‘<p>‘), то до нього можна отримати доступ через DOM як до вузла.
Вузол «Текст» являє собою весь текст (всередині елементів) на сторінці. Тому, якщо у вашому абзаці є трохи тексту, до нього можна напряму звернутися через DOM.
**Вузол «Документ»** представляє собою весь документ (кореневий вузол ієрархії/дерева DOM).
Атрибути елемента теж є вузлами DOM.
В різних браузерах по-різному відбувається механізм компонування DOM.
**JavaScript існує два основних типів значень:**
**Примітиви**: рядки, числа, булеві значення, undefined та null.
**Об'єкти**: масиви, функції, дати.
Конструктор Object створює оболонку об'єкта.
Синтаксис
// Ініціалізатор об'єктів або літерал
{ [ nameValuePair1[, nameValuePair2[, ...nameValuePairN] ] ] }

// Викликається як конструктор
new Object([value])
Параметри
nameValuePair1, nameValuePair2, ... nameValuePair*N*
Пари імен (рядків) і значення (будь-яке значення), де ім'я відділяється від значення двокрапкою.
value
Будь-яке значення.
Опис
Конструктор **Object** створює оболонку об'єкта для заданного значення. Якщо значення є null або undefined, конструктор створює і повертає порожній об'єкт, в іншому випадку, він створює об'єкт Типу, що відповідає заданному значенню. Якщо значення вже є об'єктом , то конструктор поверне значення.
Коли визваний не в контексті конструктора, Object поводиться ідентично до new Object().
JavaScript містить декілька вбудованих об'єктів: Global, Object, Error, Function, Array, String, Boolean, Number, Math, Date, RegExp. Крім того, JavaScript містить набір вбудованих операцій, які, грубо кажучи, не обов'язково є функціями або методами, а також набір вбудованих операторів, що управляють логікою виконання програм.
**Модель документу**
Після аналізу структурованого документа, будується його представлення у вигляді дерева. Дерево, в моделі DOM, складається із множини зв'язних вузлів (Node) різних типів. Як правило, розрізняють вузли наступних типів:
Документ (Document) — корінь дерева, представляє цілий документ.
Фрагмент документа (DocumentFragment) — вузол, який є коренем піддерева основного документа.
Елемент (Element) — представляє окремий елемент HTML або XML документа.
Атрибут (Attr) — представляє атрибут елемента.
Текст (Text) — представляє текстові дані, які містяться в елементі або атрибуті.
Стандартом визначаються і деякі інші типи вузлів у моделі документа.
Вузли деяких типів можуть мати гілки, інші ж можуть бути лише листами дерева. Спеціальні методи об'єктів вузлів дають можливість обходу дерева.

***7.Об`єкти: Windows, Frameset, Navigator. Модель об`єктів браузера***

Об’єкт **window** є глобальним об’єктом, доступ до нього можна отримати, просто набравши «window». Саме всередині цього об’єкту виконується весь код JavaScript. Як і у всіх об’єктів, у нього є властивості і методи:
Властивість – це змінна, що зберігається в об’єкті. Всі змінні, створені на веб-сторінці, автоматично стають властивостями об’єкта window.
Метод – це функція, що зберігається в об’єкті. Оскільки всі функції зберігаються в об’єкті window, всі вони можуть називатися «методами».
Щоб зробити перегляд документів на Web-сторінці зручнішим, можна скористатися багатовіконний інтерфейс, реалізованим за допомогою фреймів. В цьому випадку можна завантажити відразу кілька документів (Web-сторінок) і працювати одночасно з усіма ними.

**frameset**
На відміну від звичайного HTML-документа в документі з описом фреймів немає тега-контейнера <BODY>. . . </ BODY>. Замість нього використовується тег-контейнер <FRAMESET>. . . </ FRAMESET>, який ділить екран на кілька горизонтальних частин (вікон), або на декілька вертикальних вікон. Кожне з вікон описується в вигляді фрейму за допомогою тега <FRAME>. Тег <FRAMESET> містить наступні параметри:

COLS - вказує через кому ширину вертикальних вікон в пікселях або в% від ширини екрана (якщо задається *, то цього вікна відводиться решта екрану);
ROWS - вказує через кому висоту горизонтальних вікон в пікселях або в% від висоти екрана (якщо задається *, то цього вікна відводиться решта екрану);
FRAMEBORDER:
1 - фрейми мають рамку;
0 - фрейми не мають рамку.
FRAMESPACING - вказує відстань між фреймами в пікселях.
Наприклад, тег

<FRAMESET COLS = "20%, 30%, *">
<FRAME. . . >
<FRAME. . . >
<FRAME. . . >
</ FRAMESET>

ділить екран на три вертикальних вікна, які займають соответсвенно 20%, 30% і решту (50%) екрана. На місці кожного з тегів <FRAME> може бути вказаний тег <FRAMESET>. . . </ FRAMESET>. Це дозволяє формувати на екрані складну багатовіконну структуру.

**navigator**
navigator - об'єкт який містить інформацію про браузер.
Синтаксис:
window.navigator
Параметри:
window - не обовязково вказувати. Об'єкт window.
Опис:
navigator дочірний об'єкт об'єкта window який містить інформацію про браузер, пристрій на якому запущений браузер, тип зєднання інтернету, стан заряду батареї (акумулятора).

Уся інформація про браузер є лише для читання.

**Структура мови**
Структурно JavaScript можна представити у вигляді об'єднання трьох частин, що чітко різняться одна від одної :
- ядро (ECMAScript ),
- об'єктна модель браузера (Browser Object Model або BOM),
- об'єктна модель документа (Document Object Model або DOM).

*Об'єктна модель браузера*
Об'єктна модель браузера - браузероспеціфічна частина мови, яка являється прошарком між ядром і об'єктною моделлю документа. Основне призначення об'єктної моделі браузера - керування вікнами браузера і забезпечення їх взаємодії. Кожне з вікон браузера представляється об'єктом window, центральним об'єктом BOM. Об'єктна модель браузера на даний момент не стандартизована, проте специфікація знаходиться в розробці WHATWG та W3C. Крім управління вікнами, в рамках об'єктної моделі браузера.
*Об'єктна модель документа*
Об'єктна модель документа - інтерфейс програмування додатків для HTML і XML-документів. Згідно DOM документом можна поставити у відповідність дерево об'єктів, які мають ряд властивостей, які дозволяють робити з ним різні маніпуляції.

***8. Основи технології CGI. Виникнення та призначення.***

CGI (від англ. Common Gateway Interface - «загальний інтерфейс шлюзу») - стандарт інтерфейсу, використовуваного для зв'язку зовнішньої програми з веб-сервером. Програму, яка працює за таким інтерфейсом спільно з веб-сервером, прийнято називати шлюзом, хоча багато хто воліє називати «скрипт» (сценарій) або «CGI-програма».

Оскільки гіпертекст є статичним за своєю природою, веб-сторінка не може безпосередньо взаємодіяти з користувачем. До появи JavaScript, не було іншої можливості відреагувати на дії користувача, крім як передати введені ним дані на веб-сервер для подальшої обробки. У разі CGI ця обробка здійснюється за допомогою зовнішніх програм і скриптів, звернення до яких виконується через стандартизований інтерфейс - загальний шлюз.

Сам інтерфейс розроблений таким чином, щоб можна було використовувати будь-яку мову програмування, яка може працювати зі стандартними пристроями введення-виведення. Такими можливостями володіють навіть скрипти для вбудованих командних інтерпретаторів операційних систем, тому в простих випадках можуть використовуватися навіть командні скрипти.

Як працює CGI?

Узагальнений алгоритм роботи через CGI можна представити в наступному вигляді:

1. Клієнт запитує CGI-додаток за його URI.
2. Веб-сервер приймає запит і встановлює змінні оточення, через них з додатком передаються дані та службова інформація.
3. Веб-сервер перенаправляє запити через стандартний потік введення (stdin) на вхід спричиненої програми.
4. CGI-додаток виконує всі необхідні операції і формує результати у вигляді HTML.
5. Сформований гіпертекст повертається веб-сервера через стандартний потік виводу (stdout). Повідомлення про помилки передаються через stderr.
6. Веб-сервер передає результати запиту клієнта.

Області застосування CGI

Найбільш часта задача, для вирішення якої застосовується CGI - створення інтерактивних сторінок, зміст яких залежить від дій користувача. Типовими прикладами таких веб-сторінок є форма реєстрації на сайті або форма для відправки коментаря. Інша область застосування CGI, що залишається за лаштунками взаємодії з користувачем, пов'язана зі збором та обробкою інформації про клієнта: установка і читання «печенюшок» - cookies; отримання даних про браузер і операційну систему; підрахунок кількості відвідувань веб-сторінки; моніторинг веб-трафіку і т.п.

Це забезпечується можливістю підключення CGI-скрипта до бази даних, а також можливістю звертатися до файлової системи сервера. Таким чином CGI-скрипт може зберігати інформацію в таблицях БД або файлах і отримувати її звідти за запитом, чого не можна зробити засобами HTML.

***9. Технологія CGI(дивись вище питання 8). Застосування технології клієнт-сервер для роботи із гіпертекстовими документами. Розвиток технологій CGI. SSI - програмування.***

Технологія клієнт сервер CGI - Common Gateway Interface є стандартом інтерфейсу (зв'язку) зовнішньої прикладної програми з інформаційним сервером типу HTTP, Web сервер.

Велика кількість World Wide Web додатків засноване на використанні зовнішніх програм, керованих Web сервером. Використання даних програм дозволяє будувати Web додатка з динамічно оновлюваною інформацією, що зберігається в базах даних або генерується в залежності від бізнес-правил вирішуваних завдань. Для зв'язку між Web сервером і викликаються програмами широко використовується Common Gateway Interface (CGI), що має реалізації, як для Windows-орієнтованих програм, так і для додатків, що функціонують в середовищі Unix.

Зазвичай гіпертекстові документи, які добувають із WWW серверів, містять статичні дані. За допомогою CGI можна створювати CGI-програми, звані шлюзами, які у взаємодії з такими прикладними системами, як система управління базою даних, електронна таблиця, ділова графіка і ін., Зможуть видати на екран користувача динамічну інформацію.

Інтернет взагалі і WWW зокрема працює за технологією «клієнт-сервер», тобто все програмне забезпечення поділяється на клієнтську і на серверну частини. Також між ними розділені і функціональні обов'язки. Для взаємодії цих частин розроблений спеціальний протокол (в окремому випадку - протокол HTTP), і все взаємодія між клієнтом і сервером здійснюється виключно в рамках даного протоколу.

Етапи з'єднання по протоколу HTTP:

1. Формування запиту клієнтом. (Браузер формує запит з URL, набраного користувачем, з клацання на посиланні або з даних форми).

2. Встановлення з'єднання з сервером (якщо з'єднання втрачено, то на цьому HTTP-транзакція закінчиться і клієнт видасть користувачеві повідомлення про помилку).

3. Здійснення запиту і очікування відповіді від сервера.

4. Сервер приймає запит.

5. Сервер обробляє запит.

6. Генерація відповіді.

7. Прийом відповіді клієнтом.

8. Розрив з'єднання.

9. Обробка даних клієнтом (висновок або збереження даних) .

Зазвичай під запитом до сервера розуміється URL (це уніфікована форма «замовлення» даних на сервері). До власне URL можуть ще «додаватися» деякі дані, найчастіше це дані форм. При установці з'єднання з сервером спочатку відбувається трансляція символьного доменного імені, такого, як www.siemens.com, в IP-адресу, а потім здійснюється безпосередньо створення TCP / IP-з'єднання з даними IP. Коли дані HTTP-запиту послані серверу, клієнт просто чекає, поки не прийде відповідь. Поки немає звернень від клієнтів, сам HTTP-сервер просто «спить» в очікуванні запитів. Коли клієнт встановлює з'єднання, сервер «прокидається» і, прийнявши дані запиту, приступає до їх обробці. Що саме сервер робить із запитом - відомо тільки самому сервера. Єдиний результат всіх хитрих маніпуляцій - це видача відповіді, якого і очікує клієнт.

Після того як сервер видав відповідь, він розриває з'єднання і знову «занурюється в сон». Природно відзначити, що в разі виникнення помилки HTTP-транзакція може закінчитися на будь-якому з цих етапів.

Якщо документ не знайдений або якщо для доступу до нього у вас немає прав, то видається код помилки. А якщо все нормально, то інформація, що міститься в документі, включаючи супутні дані про його типі, видається у вигляді відповіді.

Server Side Includes (SSI) — мова для динамічної збірки веб-сторінок на сервері з окремих складових частин і видачі клієнтові отриманого (зібраного) HTML-документа. Використання SSI дає можливість включати один і той же фрагмент одночасно у велику кількість веб-сторінок на сервері. При зміні інформації у файлі, що включається, вона одночасно міняється відразу на всіх сторінках.

Синтаксис SSI дозволяє включати в текст сторінки інші SSI-сторінки, викликати зовнішні CGI-скрипти, реалізовувати умовні операції (if/else), працювати зі змінними тощо. Завдяки крайній простоті мови, збірка SSI-сторінок відбувається дуже швидко, проте багато можливостей повноцінних мов програмування, наприклад, робота з файлами, в SSI відсутні.

***10. Характеристика сучасних Web – серверів. Пакет програм xampp (Apache, MySQL,PHP).***

Веб-се́рвер (англ. Web Server) — це сервер, що приймає HTTP-запити від клієнтів, зазвичай веб-браузерів, видає їм HTTP-відповіді, зазвичай разом з HTML-сторінкою, зображенням, файлом, медіа-потоком або іншими даними. Веб-сервер — одна із основ Всесвітньої павутини.

Веб-сервером називають як програмне забезпечення, що виконує функції веб-сервера, так і комп'ютер, на якому це програмне забезпечення працює.

Клієнти дістаються веб-сервера за URL-адресою потрібної їм веб-сторінки або іншого ресурсу.

Додатковими функціями багатьох веб-серверів є:

- Ведення журналу серверу про звернення користувачів до ресурсів
- Автентифікація користувачів
- Підтримка сторінок, що динамічно генеруються
- Підтримка HTTPS для захищених з'єднань з клієнтам

Як клієнти для звернення до веб-серверів можуть використовуватися абсолютно різні пристрої:

- Веб-браузер — найпоширеніший спосіб
- Спеціальне програмне забезпечення може самостійно звертатися до веб-серверів для отримання оновлень або іншої інформації
- Мобільний телефон може дістатися до ресурсів веб-сервера за допомогою протоколу WAP або HTTP
- Інші інтелектуальні пристрої або побутова техніка

XAMPP — **X**(будь-яка з ОС: Linux, Windows, Mac OS, Solaris), **A**pache, **M**ySQL, **P**HP, **P**erl.

XAMPP — безкоштовна багатоплатформова збірка веб-сервера з відкритим початковим кодом, що містить HTTP-сервер Apache, базу данних MariaDB, MySQL й інтерпретатори скриптів для мов програмування PHP та Perl, а також додаткові бібліотеки, що дозволяють запустити повноцінний веб-сервер.

Для встановлення XAMPP необхідно завантажити всього один файл формату zip, tar або exe, а компоненти програми не вимагають настройки. Програма регулярно оновлюється, для включення до складу новітніх версій Apache / MySQL / PHP та Perl. Також XAMPP йде з безліччю інших модулів, включаючи OpenSSL та phpMyAdmin.

XAMPP іноді використовується і у всесвітній павутині. Також програма підтримує створення і керування базами даних MySQL та SQLite.

XAMPP можна використовувати для установки власної копії Вікіпедії на комп'ютер.

***11. Протокол передачі веб-документів HTTP. Структура протоколу. HTTP-запит та відповідь.***

HTTP — протокол передачі даних, що використовується в комп'ютерних мережах. Назва скорочена від Hyper Text Transfer Protocol, протокол передачі гіпер-текстових документів

HTTP належить до протоколів моделі OSI.

Основним призначенням протоколу HTTP є передача веб-сторінок (текстових файлів з розміткою HTML), хоча за допомогою нього успішно передаються і інші файли, які пов'язані з веб-сторінками (зображення і застосунки), так і не пов'язані з ними (у цьому HTTP конкурує з складнішим FTP).

HTTP припускає, що клієнтська програма — веб-браузер — здатна відображати гіпертекстові веб-сторінки та файли інших типів у зручній для користувача формі. Для правильного відображення HTTP дозволяє клієнтові дізнатися мову та кодування символів веб-сторінки й/або запитати версію сторінки в потрібних мові/кодуванні, використовуючи позначення із стандарту MIME.

HTTP — протокол прикладного рівня, схожими на нього є FTP і SMTP. Обмін повідомленнями йде за звичайною схемою «запит-відповідь». Для ідентифікації ресурсів HTTP використовує глобальні URI. На відміну від багатьох інших протоколів, HTTP не зберігає свого стану. Це означає відсутність збереження проміжного стану між парами «запит-відповідь». Компоненти, що використовують HTTP, можуть самостійно здійснювати збереження інформації про стан, пов'язаний з останніми запитами та відповідями. Браузер, котрий посилає запити, може відстежувати затримки відповідей. Сервер може зберігати IP-адреси та заголовки запитів останніх клієнтів. Проте, згідно з протоколом, клієнт та сервер не мають бути обізнаними з попередніми запитами та відповідями, у протоколі не передбачена внутрішня підтримка стану й він не ставить таких вимог до клієнта та сервера.

Кожен запит/відповідь складається з трьох частин:

- стартовий рядок;
- заголовки;
- тіло повідомлення, що містить дані запиту, запитаний ресурс або опис проблеми, якщо запит не виконано.

Обов'язковим мінімумом запиту є стартовий рядок. Починаючи з HTTP/1.1 обов'язковим став заголовок Host: (щоб розрізнити кілька доменів, які мають одну й ту ж IP-адресу).

Запит

Стартові рядки розрізняються для запиту й відповіді. Рядок запиту виглядає так:

‹Метод› ‹URI› HTTP/‹Версія›

де ‹Метод› можливо:

OPTIONS Повертає методи HTTP, які підтримуються сервером. Цей метод може служити для визначення можливостей веб-сервера.

GET Запитує вміст вказаного ресурсу.

HEAD Аналогічний методу GET, за винятком того, що у відповіді сервера відсутнє тіло.

POST Передає призначені для користувача дані (наприклад, з HTML-форми) заданому ресурсу.

PUT Завантажує вказаний ресурс на сервер.

PATCH Завантажує певну частину ресурсу на сервер.

DELETE Видаляє вказаний ресурс.

TRACE Повертає отриманий запит так, що клієнт може побачити, що проміжні сервери додають або змінюють в запиті.

CONNECT Для використання разом з проксі-серверами, які можуть динамічно перемикатися в тунельний режим SSL.

Відповідь сервера

Перший рядок відповіді виглядає так:

HTTP/‹Версія› ‹Код статусу› ‹Опис статусу›

Коди статусу:

1xx — інформаційний: запит прийнятий, продовжуй процес.

2xx — успіх: дія була успішно передана, зрозуміла, та прийнята.

3xx — перенаправлення: наступні дії мають бути успішно виконані для реалізації запиту.

4xx — помилка клієнта: запит містить синтаксичні помилки або не може бути виконаний.

5xx — помилка сервера: сервер не зміг виконати правильно сформований запит.

Найбільш поширені статуси:

200 OK — запит виконано успішно.

301 Moved Permanently — ресурс переміщено.

403 Forbidden — доступ до запитаного ресурсу заборонений.

404 Not Found — ресурс не знайдений.

503 Service Unavailable - сервіс недоступний.

Заголовки

Заголовки HTTP — це рядки, кожен з яких складається з імені параметра, за яким слідує двокрапка і його значення. Вони несуть інформацію для браузера або для серверних програм (таких, як CGI-застосунки). Між заголовками і тілом обов'язково повинен бути порожній рядок.

### *12. Технологія CGI та її використання. Методи Get I Post*

CGI (від англ. Common Gateway Interface - «загальний інтерфейс шлюзу») - стандарт інтерфейсу, використовуваного для зв'язку зовнішньої програми з веб-сервером. Програму, яка працює за таким інтерфейсом спільно з веб-сервером, прийнято називати шлюзом, хоча багато хто воліє називати «скрипт» (сценарій) або «CGI-програма».

Оскільки гіпертекст є статичним за своєю природою, веб-сторінка не може безпосередньо взаємодіяти з користувачем. До появи JavaScript, не було іншої можливості відреагувати на дії користувача, крім як передати введені ним дані на веб-сервер для подальшої обробки. У разі CGI ця обробка здійснюється за допомогою зовнішніх програм і скриптів, звернення до яких виконується через стандартизований інтерфейс - загальний шлюз.

Сам інтерфейс розроблений таким чином, щоб можна було використовувати будь-яку мову програмування, яка може працювати зі стандартними пристроями введення-виведення. Такими можливостями володіють навіть скрипти для вбудованих командних інтерпретаторів операційних систем, тому в простих випадках можуть використовуватися навіть командні скрипти.

Як працює CGI?

Узагальнений алгоритм роботи через CGI можна представити в наступному вигляді:

1. Клієнт запитує CGI-додаток за його URI.

2. Веб-сервер приймає запит і встановлює змінні оточення, через них з додатком передаються дані та службова інформація.

3. Веб-сервер перенаправляє запити через стандартний потік введення (stdin) на вхід спричиненої програми.

4. CGI-додаток виконує всі необхідні операції і формує результати у вигляді HTML.

5. Сформований гіпертекст повертається веб-сервера через стандартний потік виводу (stdout). Повідомлення про помилки передаються через stderr.

6. Веб-сервер передає результати запиту клієнта.

Області застосування CGI

Найбільш часта задача, для вирішення якої застосовується CGI - створення інтерактивних сторінок, зміст яких залежить від дій користувача. Типовими прикладами таких веб-сторінок є форма реєстрації на сайті або форма для відправки коментаря. Інша область застосування CGI, що залишається за лаштунками взаємодії з користувачем, пов'язана зі збором та обробкою інформації про клієнта: установка і читання «печенюшок» -

cookies; отримання даних про браузер і операційну систему; підрахунок кількості відвідувань веб-сторінки; моніторинг веб-трафіку і т.п.

Це забезпечується можливістю підключення CGI-скрипта до бази даних, а також можливістю звертатися до файлової системи сервера. Таким чином CGI-скрипт може зберігати інформацію в таблицях БД або файлах і отримувати її звідти за запитом, чого не можна зробити засобами HTML.

Запит

Стартові рядки розрізняються для запиту й відповіді. Рядок запиту виглядає так:

‹Метод› ‹URI› HTTP/‹Версія›

де ‹Метод› можливо:

GET

Запитує вміст вказаного ресурсу. Запитаний ресурс може приймати параметри (наприклад, пошукова система може приймати як параметр шуканий рядок). Вони передаються в рядку URI (наприклад: http://www.example.net/resource?param1=value1&param2=value2). Згідно зі стандартом HTTP, запити типу GET вважаються ідемпотентними — багатократне повторення одного і того ж запиту GET повинне приводити до однакових результатів (за умови, що сам ресурс не змінився за час між запитами). Це дозволяє кешувати відповіді на запити GET. Якщо назва ресурсу не вказана (у URI наявні лише схема та доменне ім'я), то веб-сервер повертає індекс директорії веб-сервера.

POST

Передає призначені для користувача дані (наприклад, з HTML-форми) заданому ресурсу. Наприклад, в блогах відвідувачі зазвичай можуть вводити свої коментарі до записів в HTML-форму, після чого вони передаються серверу методом POST, і він поміщає їх на сторінку. При цьому передані дані (у прикладі з блогами — текст коментаря) включаються в тіло запиту. На відміну від методу GET, метод POST не вважається ідемпотентним, тобто багатократне повторення одних і тих же запитів POST може повертати різні результати (наприклад, після кожного відправлення коментаря з'являтиметься одна копія цього коментаря).

***13. Змінні середовища Web – сервера. Загальна характеристика.***

При запуску CGI-скрипта веб-сервер передає йому цілий ряд параметрів у змінні середовища. Частина змінних середовища генерується веб-сервером, а інша частина формується з полів HTTP-запиту.
В "Специфікації CGI/1.1" описано стандартний набір змінних середовища CGI, що формуються веб-сервером, а також спосіб передачі CGI-скрипту полів HTTP-запиту в змінних середовища. Крім цього, ряд веб-серверів (включаючи Apache) окрім стандартних формують свої змінні, в котрих передають скрипту додаткові ("нестандартні") параметри (E-Mail адміністратора сервера и т.п.).

**Змінні середовища CGI на конкретному хостингу.**
Щоб побачити всі доступні CGI-скрипту змінні середовища на конкретному сервері, можна запустити на ньому CGI-скрипт, що виведе імена і значення всіх своїх змінних середовища:

```
#!/usr/bin/perl
print "Content-Type: text/html\n\n";
print "<HTML>\n<HEAD></HEAD><BODY>
<TABLE width=100% border=1 bordercolor=#00007F cellspacing=0>
<TR><TD align=center>Змінна</TD><TD align=center>Значення</TD></TR>\n";

foreach $itm(keys %ENV)
{print "<TR><TD>$itm</TD><TD>$ENV{$itm}</TD></TR>\n";}
print "</TABLE></BODY></HTML>\n";
```

Залежно від умов виклику (HTTP-метод, прямий або через SSI...) набір змінних може бути різним.
**Змінні середовища CGI, що формуються веб-сервером.**
**QUERY_STRING** - рядок параметрів виклику (всі символи, записані в URL після знаку '?').
**REQUEST_METHOD** - метод HTTP, за допомогою котрого викликаний скрипт. Частіше за все цей метод GET або POST, хоча можуть бути й інші (PUT, DELETE и т.п.).
**GATEWAY_INTERFACE** - версія інтерфейсу CGI у вигляді CGI/x.y. Наприклад, CGI/1.1
**REMOTE_ADDR** - містить IP-адресу комп'ютера, з котрого було здійснено звернення до веб-сервера (адреса клієнта або останнього проксі-сервера).
**REMOTE_PORT** - TCP-порт віддаленого комп'ютера, з котрого йде запит.
**REMOTE_HOST** - доменне ім'я віддаленого комп'ютера, з котрого йде запит (визначається веб-сервером через DNS по значенню REMOTE_ADDR, якщо це дозволено його конфігурацією).
**SERVER_NAME** - доменне ім'я сервера.
**SERVER_PORT** - номер TCP-порта веб-сервера.
**SERVER_ADDR** - IP-адреса сервера.
**SERVER_PROTOCOL** - версія HTTP-протоколу, що використовується для даного HTTP-запиту. Наприклад, HTTP/1.1.
**SERVER_SOFTWARE** - програмне забезпечення сервера.
**SCRIPT_NAME** - HTTP-шлях до скрипту.
**SCRIPT_FILENAME** - фізичний повний шлях до скрипту в файловій системі сервера.
**PATH_INFO** - HTTP-шлях до скрипту.
**PATH_TRANSLATED** - повний фізичний шлях до скрипту.

Якщо HTTP-метод, що використовується для методу, передбачає передачу вмісту в тілі запиту, то передаються наступні змінні:
**CONTENT_TYPE** - тип вмісту (MIME).
**CONTENT_LENGTH** - довжина вмісту
Якщо стався аутентифікований запит (із вказуванням імені користувача і пароля), то передаються змінні:
**AUTH_TYPE** - тип аутентифікації (аутентифікаційна схема, що використовується). Частіше за все - 'Basic'.
**REMOTE_USER** – ім’я користувача, що пройшов аутентифікацію.
Якщо скрипт розрахований на обслуговування декількох користувачів, за параметром **REMOTE_USER** він може їх розрізняти. Варто мати на увазі, що коли CGI-скрипт уже запущений і йому передано параметр REMOTE_USER, то користувач ВЖЕ успішно пройшов аутентифікацію на рівні веб-сервера.

### *14. Система управління базами даних (СУБД). Характеристика сучасних СУБД. Характеристика MySQL.*

База даних (БД) - упорядкований набір логічно взаємопов'язаних даних, що використовується спільно, та призначений для задоволення інформаційних потреб користувачів. У технічному розумінні включно й система управління БД.

Система управління базами даних (СУБД) - це комплекс програмних і мовних засобів, необхідних для створення баз даних, підтримання їх в актуальному стані та організації пошуку в них необхідної інформації.

Централізований характер управління даними в базі даних передбачає необхідність існування деякої особи (групи осіб), на яку покладаються функції адміністрування даними, що зберігаються в базі.

Головним завданням БД є гарантоване збереження значних обсягів інформації та надання доступу до неї користувачеві або ж прикладній програмі. Таким чином БД складається з двох частин: збереженої інформації та системи управління нею. З метою забезпечення ефективності доступу записи даних організовують як множину фактів (елемент даних).

Існує величезна кількість різновидів баз даних, що відрізняються за критеріями (наприклад, в Енциклопедії технологій баз даних [21] визначаються понад 50 видів БД).

Відзначимо тільки основні класифікації.

Класифікація БД *за моделлю даних*:

- ієрархічні,
- мережеві,
- реляційні,
- об'єктні,
- об'єктно-орієнтовані,
- об'єктно-реляційні.

Класифікація БД *за технологією фізичного зберігання*:

- БД у вторинній пам'яті (традиційні);
- БД в оперативній пам'яті (in-memory databases);
- БД у третинній пам'яті (tertiary databases).

Класифікація БД *за вмістом*:

- географічні.
- історичні.
- наукові.
- мультимедійні.

Класифікація БД *за ступенем розподіленості*:

- централізовані (зосереджені);
- розподілені.

Окреме місце в теорії та практиці займають просторові (англ. spatial), тимчасові, або темпоральні (temporal) і просторово-часові (spatial-temporal) БД.

*Ієрархічні* бази даних можуть бути представлені як дерево, що складається з об'єктів різних рівнів. Верхній рівень займає один об'єкт, другий - об'єкти другого рівня і т.д.

Між об'єктами існують зв'язки, кожен об'єкт може включати в себе декілька об'єктів більш низького рівня. Такі об'єкти перебувають у відношенні предка (об'єкт більш близький до кореня) до нащадка (об'єкт більш низького рівня), при цьому можлива ситуація, коли об'єкт-предок не має нащадків або має їх декілька, тоді як у об'єкта-нащадка обов'язково тільки один предок. Об'єкти, що мають загального предка, називаються близнюками.

*Мережеві* бази даних подібні до ієрархічних, за винятком того, що в них є покажчики в обох напрямках, які з'єднують споріднену інформацію.

До основних понять мережевої моделі бази даних відносяться: рівень, елемент (вузол), зв'язок.

Вузол - це сукупність атрибутів даних, що описують деякий об'єкт. На схемі ієрархічного дерева вузли представляються вершинами графа. У мережній структурі кожен елемент може бути пов'язаний з будь-яким іншим елементом.

Незважаючи на те, що ця модель вирішує деякі проблеми, пов'язані з ієрархічною моделлю, виконання простих запитів залишається досить складним процесом.

Також, оскільки логіка процедури вибірки даних залежить від фізичної організації цих даних, то ця модель не є повністю незалежною від програми. Іншими словами, якщо необхідно змінити структуру даних, то потрібно змінити і додаток.

*Реляційна* модель орієнтована на організацію даних у вигляді двовимірних таблиць. Кожна реляційна таблиця являє собою двовимірний масив і має наступні властивості:

- кожен елемент таблиці - один елемент даних;
- всі осередки в стовпчику таблиці однорідні, тобто всі елементи в стовпчику мають однаковий тип (числовий, символьний тощо);
- кожен стовпчик має унікальне ім'я;
- однакові рядки в таблиці відсутні;
- порядок проходження рядків і стовпчиків може бути довільним.

*Об'єктна* СУБД ідеально підходить для інтерпретації складних даних, на відміну від реляційних СУБД, де додавання нового типу даних досягається ціною втрати продуктивності або за рахунок різкого збільшення термінів і вартості розробки додатків. Об'єктна база, на відміну від реляційної, не вимагає модифікації ядра при додаванні нового типу даних. Новий клас і його екземпляри просто надходять у зовнішні структури бази даних. Система управління ними залишається без змін.

*Об'єктно-орієнтована* база даних (ООБД) - база даних, в якій дані оформлені у вигляді моделей об'єктів, що включають прикладні програми, які управляються зовнішніми подіями. Результатом поєднання можливостей (особливостей) баз даних і можливостей об'єктно-орієнтованих мов програмування є об'єктно-орієнтовані системи управління базами даних (ООСУБД). ООСУБД дозволяють працювати з об'єктами баз даних також, як з об'єктами у програмуванні в об'єктно-орієнтованих мовах програмування. ООСУБД розширює мови програмування, прозоро вводячи довготривалі дані, управління паралелізмом, відновлення даних, асоційовані запити й інші можливості.

Об'єктно-орієнтовані бази даних звичайно рекомендовані для тих випадків, коли потрібна високопродуктивна обробка даних, які мають складну структуру.

Система, яка забезпечує об'єктну інфраструктуру і набір реляційних розширювачів, називається "*об'єктно-реляційною*".

Об'єктно-реляційні системи поєднують переваги сучасних об'єктно-орієнтованих мов програмування з такими властивостями реляційних систем як множинні представлення даних і високорівневі непроцедурні мови запитів.

*За технологією обробки даних* бази даних поділяються на централізовані й розподілені.

*Централізована* база даних зберігається у пам'яті однієї обчислювальної системи. Якщо ця обчислювальна система є компонентом мережі ЕОМ, можливий розподілений доступ до такої бази. Такий спосіб використання баз даних часто застосовують у локальних мережах ПК.

*Розподілена* база даних складається з декількох, можливо пересічних або навіть дублюючих одна одну частин, які зберігаються в різних ЕОМ обчислювальної мережі. Робота з такою базою здійснюється за допомогою системи управління розподіленою базою даних (СУРБД).

*За способом доступу* до даних бази даних поділяються на бази даних з локальним доступом і бази даних з віддаленим (мережевим) доступом.

Системи централізованих баз даних з мережевим доступом припускають різні архітектури подібних систем:

- файл-сервер;
- клієнт-сервер.

Файл-сервер. Архітектура систем БД з мережевим доступом передбачає виділення однієї з машин мережі в якості центральної (сервер). На такій машині зберігається спільно використовувана централізована БД. Усі інші машини мережі виконують функції робочих станцій, за допомогою яких підтримується доступ користувальницької системи до централізованої бази даних. Файли бази даних відповідно до призначених для

користувача запитів передаються на робочі станції, де в основному і проводиться обробка. При великій інтенсивності доступу до одних і тих же даних продуктивність інформаційної системи падає. Користувачі можуть створювати також на робочих станціях локальні БД, які використовуються ними монопольно.

Клієнт-сервер. У цій концепції мається на увазі, що крім зберігання централізованої бази даних центральна машина (сервер бази даних) повинна забезпечувати виконання основного обсягу обробки даних. Запит на дані, який видається клієнтом (робочою станцією), породжує пошук і вилучення даних на сервері. Витягнуті дані (але не файли) транспортуються по мережі від сервера до клієнта. Специфікою архітектури клієнт-сервер є використання мови запитів SOL.

**MySQL** - вільна система управління базами даних. MySQL є власністю компанії Oracle Corporation, що отримала її разом з поглиненою Sun Microsystems, яка здійснює розробку і підтримку додатку. Розповсюджується під GNU General Public License і під власною комерційною ліцензією, на вибір. Крім цього розробники створюють функціональність на замовлення ліцензійних користувачів, саме завдяки такому замовленню майже в найраніших версіях з'явився механізм реплікації.

Цю систему управління базами даних з відкритим кодом було створено як альтернатива комерційним системам. MySQL із самого початку була дуже схожою на mSQL, проте з часом вона все розширювалася і зараз MySQL - одна з найпоширеніших систем управління базами даних. Вона використовується, у першу чергу, для створення динамічних веб-сторінок, оскільки має чудову підтримку з боку різноманітних мов програмування.

MySQL є рішенням для малих і середніх додатків. Зазвичай MySQL використовується як сервер, до якого звертаються локальні або віддалені клієнти, проте до дистрибутиву входить бібліотека внутрішнього сервера, що дозволяє включати MySQL до автономних програм. Вихідні коди сервера компілюються на багатьох платформах. Найповніше можливості сервера виявляються в UNIX-системах, де є підтримка багатонитевості, що підвищує продуктивність системи в цілому.

Гнучкість СУБД MySQL забезпечується підтримкою великої кількості типів таблиць: користувачі можуть вибрати як таблиці типу MyISAM, що підтримують повнотекстовий пошук, так і таблиці InnoDB, що підтримують транзакції на рівні окремих записів. Більш того, СУБД MySQL поставляється із спеціальним типом таблиць EXAMPLE, що демонструє принципи створення нових типів таблиць. Завдяки відкритій архітектурі й GPL-ліцензуванню, в СУБД MySQL постійно з'являються нові типи таблиць. MySQL характеризується великою швидкістю, стійкістю і простотою використання.

Для некомерційного використання MySQL є безкоштовною. Можливості сервера MySQL:

- простота у встановленні та використанні;
- підтримується необмежена кількість користувачів, що одночасно працюють із БД;
- кількість рядків у таблицях може досягати 50 млн.;
- висока швидкість виконання команд;
- наявність простої та ефективної системи безпеки.

***15. PHP. Основи синтаксису мови PHP. Прості оператори. Робота з Web-документом.***

PHP - це мова програмування, код якої вбудовується безпосередньо в HTML-сторінку. Програму, написану на PHP, називають PHP-скриптом. При запиті користувача web-сервер переглядає документ, виконує знайдені в ньому PHP-інструкції (оператори мови PHP), а результат їхнього виконання повертає користувачеві. При цьому статична частина документа, написана мовою HTML, фактично є шаблоном, а змінювана частина формується нри виконанні PHP-інструкцій. Для віддаленого користувача подібні документи нічим не відрізняються від звичаїйних статичних HTML-докумеитів, за винятком того, що в розширенні імені файїлу для таких документів може стояти не htm або html, а php (а в старих версіях phtml або php3). PHP - це система розробки скриптів, що включає в себе CGI - інтерфейс, інтерпретатор мови та набір функцій для доступу до баз даних і різних об'єктів WWW (функції для роботи з електронною поштою, FTP, http, тощо). На сьогодні, PHP є одним із найбільш зручних і водночас достатньо потужним засобом розробки додатків WWW і інтерфейсів до баз даних в мережі Інтернет. PHP-скрипти знаходяться на сервері і їхній вміст відвідувачеві сайту переглянути неможливо. Файли скриптів мають розширення *.php. При активації скринту (заниті користувача) серверна програма виконує всі команди php цього скринту, не торкаючись статичної частини документа (HTML-код) і результат повертається програмі-браузеру. Після виконання PHP-скринта користувач бачить звичайну веб-сторінку, яка відрізняється від інших тільки розширенням.

**Установка PHP для роботи із сервером Apache**

Можлива установка PHP як модуля Apache і як окремого CGI-модуля.

Для установки необхідно:

- зупинити сервер Apache;
- розпакувати файли PHP у який-небудь каталог (наприклад, *c:\php\);*
- скопіювати файл *php4ts.dll* у *windows\system(32)\;*
- змінити настроювання Apache у файлі *apache\conf\httpd.conf:*

а) при установці як модуля:

*LoadModulephp4_module c:/php/sapi/php4apache.dll*
*AddType application/x-httpd-php .php4*
*AddType application/x-httpd-php .phtml*
*AddType application/x-httpd-php .php3*
*AddType application/x-httpd-php .php*
*AddType application/x-httpd-php-source .phps*
*# Додаємо PHP - розширення в рядок*
*Directorylndex index.html index.phtml index.phpS default.php default.phpS default.phtml*

б) при установці як CGI - додатка:

*ScriptAlias /php/ "C.Vphp/"*
*<Directory "C:/php">*
*AllowOverride None*
*Options None*
*</Directory>*
*Action application/x-httpd-php "/php/php.exe"*
*AddType application/x-httpd-php .phtml*
*AddType application/x-httpd-php .php3*
*AddType application/x-httpd-php .php*
*Action application/x-httpd-php-source "/php/php.exe"*
*AddType application/x-httpd-php-source .phps*
*# Додаємо PHP - розширення в рядок:*
*Directorylndex index.html index.phtml index.phpS default.php default.php3 default.phtml*

- · скопіювати файл *php3-dist.ini* у каталог **windows**, перейменуйте його в *php.ini*
- · роз коментувати у файлі *php.ini* рядок: *;extention=ім'я_модуля.dll* для тих модулів, що планується використовувати (для роботи з MySQL завантаження модулів не потрібно - її підтримка убудована в PHP4).

**Вступ у PHP**

PHP - це скрипт-мова (scripting language), що вбудовується в HTML, що інтерпретується й виконується на сервері. Простіше всього це показати на прикладі.

Створіть у каталозі сервера файл *ex01.php,* що містить наступний текст.

*<html>*
*<head>*
*<title>Example</title>*
*</head>*

*<body>*
*<? echo "Hi, I'm a PHP script!"; ?>*
*</body>*
*</html>*
Потім запустивши браузер, наберіть у рядку URL: *http://server_url/labal/ex01.php,* де server_url - адреса вашого web-сервера. Після виконання цього скрипту ми одержимо сторінку.
Розглянемо даний приклад докладніше: усі рядки приклада крім однієї – це звичайний статичний набір тегів HTML. Найбільший інтерес представляє рядок, укладений у тег <? ?>. Це і є PHP-скрипт. У даному випадку - це єдина команда PHP - *echo,* що здійснює вивід у потік стандартного висновку фрази 'Hi, I'm a PHP script!'.
Таким чином, після обробки сторінки на сервері браузерові клієнта передається наступний HTML-код:
*<html>*
*<head>*
*<title>Example</title>*
*</head>*
*<body>*
*Hi, I'm a PHP script!"*
*</body>*
*</html>*
фактично, після обробки сервером, замість блоків коду PHP підставляються результати їхнього виконання, тобто те, що дана ділянка коду вивела в стандартний потік виводу (наприклад, функцією *echo).*
Основна відмінність PHP від CGI-скриптів, написаних на інших мовах, типу Perl чи C - це те, що в CGI-програмах ви самі пишете виведений HTML-код, а, використовуючи PHP - ви вбудовуєте свою програму в готову HTML-сторінку, використовуючи відкриваючий і закриваючий теги <? і ?>.
Відмінність PHP від JavaScript, полягає в тому, що PHP-скрипт виконується на сервері, а клієнту передається результат роботи, тоді як у JavaScript-код цілком передається на клієнтську машину, і тільки там виконується.
PHP дуже схожий на Active Server Pages (ASP) і Java Server Pages (JSP). Усі три мови дозволяють розміщати код, який виконується на Web-сервері, всередині HTML сторінок.

**Можливості PHP**

У декількох словах: на PHP можна зробити все, що можна зробити за допомогою CGI-програм. Наприклад: обробляти дані з форм, генерувати динамічні сторінки, одержувати й посилати куки (cookies).
Крім цього в PHP включена підтримка багатьох баз даних (databases), що робить написання Web-програм із використанням БД дуже простим.
От неповний перелік підтримуваних БД:

| | | |
|---|---|---|
| Dabase, | InterBase, | Solid; |
| dBase, | MSQL, | Sybase; |
| Empress, | MySQL, | Velocis; |
| FilePro, | Oracle, | Unix rdbm; |
| Informix, | PostgreSQL. | |

В добавок до всього PHP розуміє протоколи IMAP, SNMP, NNTP, POP3 і навіть HTTP, а також має можливість працювати із сокетами (sockets) TCP/IP і спілкуватися на інших протоколах. Синтаксис PHP дуже схожий на синтаксис C чи Perl. Люди, знайомі з програмуванням, дуже швидко зможуть почати писати програми на PHP. У цій мові немає строгої типізації даних і немає необхідності в діях по виділенню/звільненню пам'яті.
Програми, написані на PHP, досить легкочитаємі. Написаний PHP – код легко прочитати й зрозуміти, на відміну від Perl-програм.

**Синтаксис PHP**

Скрипт мова PHP подібна по синтаксисі мові C за багатьма показниками. Вона підтримує змінні, масиви, звертання до функцій, різні типи змінних і безліч інших речей, що вам можуть знадобитися для написання складних програм. Кожна команда PHP починається з тега <?, і закінчується ?>. Команди можуть бути згруповані усередині однієї пари <? ?> і відокремлюватися друг від друга символом ; . Підтримуються змінні, імена змінних починаються із символу $. Так, наприклад, щоб привласнити значення 5 змінної a, потім відобразити її, можна написати наступний фрагмент:
*<?$a = 5.?>*
*<?echo $a?>*
це можна записати також у виді:
*<?; $a = 5; echo $a ?>*
чи навіть:
*<?$a* = 5;
*echo $a;?>*
це можна побачити у прикладі *ex02.php.*

**Приклад** *ex02.php*
*Перший тип об'яви: <br>*
*<?$a* **= 5?>**
*<?echo $a?> <br>*
*Другий тип об'яви: <br>*
*<?; $a = 5; echo $a ?> <br>*
*Третій тип об'яви: <br>*
*<? $a= 5;*
*echo $a; ?>*
Зайві символи пробілу, табуляції і нового рядка ігноруються. Це потрібно для того, щоб формувати блоки програми PHP, для більшої зручності читання. Регістр написання має значення для імен змінних, але не для імен функцій. Пізніше в цій документації, при огляді функцій, регістр використовується тільки для того, щоб зробити імена функцій більш читабельними. У фактичній програмі Ви можете використовувати будь-який регістр, що побажаєте. Коментарі підтримуються. Коментарі записуються точно так само як коментарі в Мові С. /* починає коментар, */ закінчує коментар. Коментарі можуть бути поміщені в будь-якім місці усередині блоку <? ... ?>.

***16. PHP. Типи даних мови PHP. Приведення типів. Визначення змінних.***

Імена змінних позначаються знаком $ (долара). Те ж саме "Привіт, я - скрипт PHP!" Можна отримати наступним чином:

```
<?php
$message = "Привіт, я - скрипт PHP!" ;
echo $message ;
?>
```

**Типи даних в PHP**
PHP підтримує вісім простих типів даних: Чотири скалярних типи:
-boolean (двійкові дані)
-integer (цілі числа)
-float (числа з плаваючою крапкою або 'double')
-string (рядки)
Два змішаних типи:
-array (масиви)
-object (об'єкти)
І два спеціальних типи:
-resource (ресурси)
-NULL ("порожні")
Існують також кілька псевдотипів:
-mixed (змішані)
-number (числа)
-callback (зворотного виклику)

Опис типу змінної не є обов'язковим і тому програмістом може бути і не визначений. Кожна змінна автоматично перетворюється в кожній з типів й різні функції використовують потрібний тип. проте, існує декілька функцій, для яких важливий тип змінної. Для ініціалізації (визначення) змінної необхідно їй просто присвоїти значення.

```
<?php
$a = 5; - змушує змінну $a стати змінною типу Integer
$b = 5.0; - змушує змінну $b стати змінною типу Double
$0 = "5"; - змушує змінну $c стати змінною типу String
?>
```

Зазвичай використовують три типи змінних. Довгі цілі (long int), подвійної точності з плаваючою комою, (double) і символьні рядки (strings). Тип змінних виявляється автоматично. Наприклад:
*<?$a = 5?>* - змушує *$a* стати змінної *muny INTEGER;*
*<?$a = 5.0?>* - змушує *$a* стати змінної типу *DOUBLE;*
*<?$a = "5"?>* - змушує *$a* стати змінної типу *STRING.*
Тип змінної взагалі не дуже важливий. Кожна змінна, незалежно від типу, перетворюється в кожній із трьох типів внутрішньо, і різні функції пробують використовувати правильний тип. Є тільки кілька функцій, для яких важливий тип змінної. Усі три типи змінних можуть також розглядатися як масиви, якщо до їхніх імен додається [значення]. На відміну від C, масиви в PHP фактично являють собою асоціативні масиви, подібні тим, що використовуються в Perl. Наступний запис, вірний:

```
<? $a[0] = 5;
$a ["hello"] = 6;
echo $a[0];
echo $a["hello"]; ?>
```

зверніть увагу, що, якщо ім'я змінної використовується, і як масив і як звичайна змінна, то ім'я звичайної змінної є синонімом елементу масиву з індексом «0». Тобто *$a = 1,* це теж саме, що *$a[0] = 1.*
PHP також підтримує не-індексовані масиви. Не-індексований масив генерує власний індекс, у міру додавання елементів до нього. Наприклад,:

```
$a[] = "Hello";
$a[] = "There";
```

першому елементу, що вставляється в не-індексований масив, завжди привласнюється індекс 0, другий 1 індекс, і m.д. Отже вищезгадані елементи можуть бути роздруковані за допомогою:

*echo $a[0];*
*echo $a[1];.*
Ви можете використовувати функцію count(), для того щоб визначити кількість елементів для будь-якого масиву.
Ще одна властивість мови, це те, що тип змінної визначає, які основні операції можуть бути виконані. Наприклад, *$a = $b + $c*може поводитися подвійно. Якщо $b це число, то числове значення $c додається до $b, і сума зберігається в $a. У цьому випадку тип $c не важливий. Операція керується типом першої змінної. Якщо $b рядок, то значення рядка $c конкатенується з $b, і результуючий рядок міститься в $a. Це також приводить до деяких непорозумінь.
Приклад *ex03.php:*
*<? $a[] = "Hello";*
*$a[] = "There";*
*echo $a[0];*
*echo "<br>";*
*echo $a[l];*
*echo "<br> кількість елементів масиву а : ";*
*echo count($a)?>*

При використанні змінних тину string можна використовувати як подвійні так і одинарні лапки. Але між ними є різниця :
**<?**
$opiIIa='Оболонь';
$8='Пиво $opIIIa';
echo $s;
//виводить Пиво $opiIIa
echo '<br>';
$8="Пиво $opiIIa";
echo $s:
// виводить Пиво Оболонь
**?>**
Змінна розглядається як масив, якщо до її імені додається значення індекса, взятого у квадратні дужки [].
Наприклад
<?php
$a[0] = 10;
$a[1] = 14;
$a[2] = 8;
//Дістали масив $a з трьох елементів
?>
Індексом масиву може бути або ціле число (Integer) або стрічка (string), елементами масиву можуть бути значення будь-якого типу.
**<?**
$a['name']='shoues';
$a['color']=Yed';
$a['size']=36;
**?>**
Масив, в якому індексами є стрічки називається асоціативним.

***17. PHP. Вирази і операції. Вирази і операції мови PHP. Оператори мови PHP.***

Операції, що визначені в мові php

**Арифметичні операції**

| **Приклад** | **Назва** | **Результат** |
|---|---|---|
| $a + $b | Додавання | Сума $a і $b |
| $a - $b | Віднімання | Різниця $a і $b |
| $a * $b | Множення | Добуток $a і $b |
| $a / $b | Ділення | Частка від ділення $a на $b |
| $a % $b | Остача | Остача від цілочисельного ділення $a на $b |

**Логічні операції**

| **Приклад** | **Назва** | **Результат** |
|---|---|---|
| *$a == $b* | дорівнює | **TRUE,** якщо $a дорівнює $b. |
| *$a === $b* | ідентично | **TRUE,** якщо $a дорівнює $b і вони одного типу, (тільки в PHP4**)** |
| *$a != $b* | не дорівнює | **TRUE,** якщо $a не дорівнює $b. |
| *$a o $b* | не дорівнює | **TRUE,** якщо $a не дорівнює $b. |
| *$a !== $b* | не ідентично | **TRUE,** якщо $a не дорівнює $b або вони різних типів, (тільки в PHP 4 і більше) |
| *$a < $b* | менше | **TRUE,** якщо $a строго менше $b. |
| *$a > $b* | більше | **TRUE,** якщо $a строго більше $b. |
| *$a <= $b* | менше або дорівнює | **TRUE,** якщо $a менше або дорівнює $b. |
| *$a >= $b* | більше або дорівнює | **TRUE,** якщо $a більше або дорівнює $b. |
| *! $a* | Заперечення | **TRUE,** якщо $a не **TRUE.** |
| *$ a & &* | Логічне «і» | **TRUE,** якщо і $a, і $b **TRUE.** |
| *$a \\ $b* | Логічне «або» | **TRUE,** якщо $a або $b **TRUE.** |

**Стрічкові операції**

Основними операціями із стрічками є:

- конкатенація - повертає об'єднання із правого та лівого аргументів.
- присвоєння ('.=') - приєднує правий аргумент до лівого аргументу.

```
<?
$a = "Hello
$b = $a . "World!": // тепер $b містить "Hello World!"
$a = "Hello ";
$a .= "World!"; // тепер $a містить "Hello World!"
?>
```

Стрічку можна розглядати і як масив символів

```
<?
$a='HeIIo';
$c=$a[1];
// с буде мати значення 'e'
?>
```

*Нумерація символів, як і в масивах, починається з нуля.*

**Операції присвоєння**

Крім базової операції присвоєння (=), існують "комбіновані операції" для всіх бінарних, арифметичних І стрічкових операцій, які дають змогу використати значення у виразі, а потім установити його значення в результат цього виразу. Наприклад:

```
$3 — 3j
$a += 5; // установлює в $a 8,
//ніби ми сказали: $a = $a + 5;
$b = "Hello
$b .= "There!"; // встановлює в $b "Hello There!",
// аналогічно $b = $b . "There!";
```

**Операції інкременту/декременту**

PHP підтримує операції pre- і post-інкременту й декременту.

| Приклад | Назва | Ефект |
|---|---|---|
| ++$a | Pre-increment | Збільшує $a на 1, потім повертає $a. |
| $a++ | Post- | Повертає $a, потім збільшує $a на 1. |
| —$a | Pre-decrement | Зменшує $a на 1, потім повертає $a. |
| $a— | Post- | Повертає $a, потім зменшує $a па 1. |

Наведемо приклад скрипту, який демонструє дію цих операцій:

```
<?php
echo "<h3>Postincrement</h3>";
$a = 5;
echo "Повинне бути 5:". $a++ . "<br> ";
echo "Повинне бути 6:". $a . "<br>";
echo "<h3>PreincreiTient</h3>";
$a = 5;
echo "Повинне бути 6:". ++$a . "<br>";
echo "Повинне бути 6:". $a . "<br>";
echo "<h3>Postdecrement</h3>";
$a = 5;
echo "Повинне бути 5:". $a- . "<br>";
echo "Повинне бути 4:". $a . "<br>";
echo "<h3>Predecreinent</h3>";
$a = 5;
echo "Повинне бути 4:". -$a . "<br>";
echo "Повинне бути 4:". $a . "<br>";
?>
```

**Математичні вирази**

PHP підтримує повний набір математичних операцій, у виразах. Враховується порядок операцій. Наступні оператори, припустимі:

*<? $a = 2 + 1 ?> Додавання*
*<? $a = 2 -1 ?> Віднімання*
*<? $a = 2 * 1 ?> Множення*
*<? $a = 2/1 ?> Ділення*
*<? $a = 2 % 1 ?> Ділення по модулю*

підтримуються і дужки і порядок операцій, так, що наступна запис вірний:

*<?$a = (2+1)"3+6/3?>*

підтримуються С-подібні оператори збільшення += і зменшення -= . Тобто

*<? $a +— $b?>*
це еквівалентно:
*<? $a = $a + $b?>*
підтримуються С-подібні порозрядні оператори =& і |=. Тобто.
*<? $a&= 4?>*
це еквівалентно:
*<? $a = $a & 4?>*.

**Список деяких математичних функцій php**
**abs -** абсолютне значення
**acos -** арккосинус
**asin -** арксинус
**atan -** арктангенс
**cos -** косинус
**cosh -** гіперболічний косинус
**deglrad -** конвертує число в градусах в еквівалент у радіанах
**exp -** е в степені...
**log10 -** логарифм із основою **10**
**log -** натуральний логарифм
**max($argl,$arg2,...) -** повертає найбільше значення з значень параметрів Sargl, $arg2 ...
**pi -** значення рі
**pow(base,exp) -** Повертає *base*, в степені *exp.*
**rad2deg -** конвертує число в радіанах в еквівалент у градусах
**rand -** генерує випадкове число
**round -** округляє число із плаваючою крапкою/AoaI
**sin -** синус
**sinh -** гіперболічний синус
**sqrt -** квадратний корінь
**tan -** тангенс
**tanh -** гіперболічний тангенс

***18. PHP. Оператори розгалуження, циклу, виклик функцій.***

Умовні оператори if, else, elseif, switch
Конструкція **if**
Вказані дії виконуються тільки тоді, коли умова є істинною

```
if (умова) {                    if($index > 0){
        Дія;                        echo 'Index > 0';
}                               }
```

Конструкція **if...else**
Якщо умова істинна, виконується дія із блоку if, в протилежному випадку — з блоку else.

```
if(умова){              if($index > 0){
  Дія;                    echo ‘Так';
}else{                          } else {
  Дія;                    echo 'Ні';
}                               }
```

Конструкція **elseif**
Якщо умова блоку if істина, виконаються дії блоку if. В іншому випадку, якщо умова блоку elseif істина, виконаються дії блоку elseif. У всіх інших випадках виконається дії з блоку else.

```
if(умова){                      if($numb= 5 &&
                                        $numb = 10)
 Дія;                             $discount = 5;
}else{                           }else{
 Дія;                             $discount =10;
}                               }
```

Конструкція **switch** Якщо значення змінної відповідає значенню одного з блоків case, виконаються дії з цього блоку. В іншому випадку - з блоку default.

```
switch(Змінна){
        case Значення 1:
                Дія 1;
                [break;]
        case Значеннння 2:
                Дія 2;
                [break;]
        [default: Дія;]
}
switch($day){
        case 1:
          echo 'Понеділок';break;
        case 2:
          echo 'Вівторок'; break;
        case 3:
          echo 'Середа'; break;
        case 4:
          echo 'Четвер'; break;
        case 5:
          echo  'П’ятниця'; break;
        case 6:
          echo 'Субота'; break;
        case 7:
          echo ‘Неділя'; break;
        default:
          echo 'Немає такого дня';
```

Оператори циклу for, while
Цикли призначені для багаторазового виконання набору інструкцій. У **циклі for** вказується початкове і кінцеве значення лічильника, а также крок, з яким лічильник буде змінюватися. Змінюватися лічильник може як в позитивну, так і негативну сторону. Дії виконаються стільки разів, скільки ітерацій пройде від початкового значення лічильника до досягнення кінцевого, з вказаним кроком.

```
for(початок;кінець;крок){
  Дія;
  ...
}
```

```
for($i = 1; $i = 5; $i++) {
  $sum += $i;
  eho $sum;
}
```

**Цикл while.** Дії будуть виконуватися до тих пір, поки умова істинна. **Цикл while** є циклом з передумовою.

```
while (умова) {
 Дія;
 ...
}
```

```
while ($state == ‘Ранок') {
  echo 'Робочий день починається';
  $state = ‘Обід';
}
```

**Цикл do ... while** Цикл **do ... while** є циклом з післяумовою. Це означає, що спочатку буде виконуватися дія, а потім перевірятися умова. Таким чином дія завжди виконається мінімум один раз.

```
do{
  Дія;
  ...
} while (умова);
```

```
do{
echo ‘Удар';
} while ($state == ‘Бокс');
```

Оператори управління циклами break, continue

**Break** перериває роботу циклу. Інтерпретатор перейде до виконання інструкцій, наступних за циклом. **Continue**перериває виконання поточної ітерації циклу. Цикл продовжить виконуватися з наступної ітерації

```
$index = 1;
while ($index < 10){
  echo "$index ";
  $index++;
  if ($index == 5)
    break;
}

$index = 0;
while ($index < 10){
  $index++;
  if ($index == 5)
    continue;
  echo "$index";
}
```

Оператор циклу foreach

**Цикл foreach** Дуже зручний при роботі з масивами. Зазначені дії виконуються для кожного елемента масиву $array, при цьому $key - номер елемента масиву $array, $ value - значення цього елемента.

```
foreach ($array as [ $key => ] $value){
 Дія;
 ...
}
```

```
$pets[] = 'Собака';
$pets[] = 'Кіт';
$pets[] = 'Риба';
foreach ($pets as $index => $value) {
  echo "Елемент № $index має значення: \"$value\"<br>";
}
```

Приклад 2

```
<html>
<head>
<title>Перегляд асоціативного масиву</title>
</head>
<body>
<?php
$ціна = array ("помідори" => 15, "огірки" => 12);
foreach ($ціна as $овочі => $грн)
 {
 echo  "$овочі коштують $грн грн.<br>";
```

```
  }
?>
</body>
</html>
```
РЕЗУЛЬТАТ ПРИКЛАДУ 2:
помідори коштують 15 грн. огірки коштують 12 грн.

ПРИКЛАД 3
```
<html>
<head>
<title>Перегляд двовимірного масиву</title>
</head>
<body>
<?php
$дані = array (
        "Іванов" => array ("ріст" => 174, "вага"  => 68)
        "Петров" => array ("ріст" => 181, "вага"  => 90),
        "Сидоров" => array ("ріст" => 166, "вага"  => 73));
foreach ($дані as $прізвище => $дані1)
  {
  echo  "<br>$прізвище:<br>";
  foreach ($дані1 as $параметр => $pp)
    {
    echo  "$параметр = $pp<br>";
    }
  }
?>
</body>
</html>
```
РЕЗУЛЬТАТ ПРИКЛАДУ3 :
Іванов: ріст = 174 вага = 68
Петров: ріст = 181 вага = 90
Сидоров: ріст = 166 вага = 73

ПРИКЛАД 4
```
<?php
$a = array ('a' => 'apple ', 'b' => 'banana ',
           'c' => array ('x ',' y ',' z'));
print_r ($a);
?>
```
РЕЗУЛЬТАТ ПРИКЛАДУ 4 :
Array ( [a] => apple [b] => banana [c] => Array ( [0] => x [1] => y [2] => z ))
print_r - Вивід усіх эллементів масиву. Зручна функція для відладки.
Підрахунок кількості елементів
Кількість елементів в масиві можна визначити за допомогою функцій count() або sizeof().
```
<html>
<head>
<title>Розмір масиву</title>
</head>
<body>
<?php
$фрукти = array("яблуко", "груша", "слива", "персик");
echo "Розмір масиву \$фрукти рівний ".count($фрукти)".<br>";
echo "Останній елемент масиву \$фрукти - ".$фрукти[count($фрукти) - 1]".<br>";
?>
</body>
</html>
```
Частота входження елементів в масив

Частоту входження елементів в масив можна визначити за допомогою функції array_count_values _. Ця функція повертає масив, в якому ключами є елементи досліджуваного масиву, і значеннями - частоти їх входження в досліджуваний масив.

```
>html>
>head>
>title>Розмір масиву>/title>
>/head>
>body>
>?php
$фрукти = array("яблуко", "груша", "слива", "персик", "груша");
print_r (array_count_values($фрукти));
?>
>/body>
>/html>
```

РЕЗУЛЬТАТ ПРИКЛАДУ 5:

Array ( [яблуко] => 1 [груша] => 2 [слива] => 1 [персик] => 1 )

Функція print_r() відображає ключі і значення масиву, вказаного в аргументі.

PHP підтримує концепцію змінних функцій. Це означає, що коли до імені змінної прикріплені круглі дужки, PHP шукає функцію з тим же ім'ям, що і результат обчислення змінної, і намагається її виконати. Цю можливість можна використовувати для реалізації обернених викликів, таблиць функцій і багатьох інших речей.

Змінні функції не будуть працювати з такими мовними конструкціями як echo, print, unset(), isset(),empty(), include, require і т.п. Вам необхідно реалізувати свою функцію-обгортку для того, щоб наведені вище конструкції могли працювати з змінними функціями.

Приклад. Робота з функціями за використанням змінних

```
<?php
function foo() {
    echo "В foo()<br />\n";
}

function bar($arg = '')
{
    echo "В bar(); аргумент був '$arg'.<br />\n";
}

// Функція-обгортка для echo
function echoit($string)
{
    echo $string;
}

$func = 'foo';
$func();        // Викликає функцію foo()

$func = 'bar';
$func('test');  // Викликає функцію bar()

$func = 'echoit';
$func('test');  // Викликає echoit()
?>
```

***19.PHP.Оператори підключення файлів***

В мові програмування PHP є два оператори підключення файлів:

- include
- require

Ці дві конструкції ідентичні у всьому, крім що include () виводить Warning !, а require () видає Fatal Error. Інакше кажучи, використовуйте require (), якщо ви хочете, щоб відсутність файлу зупиняло процес виконання роботи сторінки. include () не працює таким чином, скрипт продовжить виконання роботи сторінки.

Інсує ще оператор *include_once ma require_once.*

*include_once* включає і виконує зазначений файл під час виконання скрипта. Його поведінка ідентично висловом include, з тією лише різницею, що якщо код з файлу вже один раз був включений, він не буде включений і виконаний повторно і поверне TRUE.
include_once може використовуватися в тих випадках, коли один і той же файл може бути включений і виконаний більш ніж один раз під час виконання скрипта, в даному випадку це допоможе уникнути проблем з перевизначенням функцій, змінних і т.д.

require_once аналогічно require за винятком того, що PHP перевірить, чи включався вже даний файл, і якщо так, не буде включати його ще раз.

***20.PHP.Функції PHP для роботи СУБД на прикладі MYSQL***

**mysql_connect()** - Відкриває з'єднання з сервером MySQL.

Приклад:

```
<?php
$link = mysql_connect('localhost', 'mysql_user', 'mysql_password');
if (!$link) {
   die('Помилка з’єднання: ' . mysql_error());
}
echo 'Успішне з’єднання';
mysql_close($link);
?>
```

**mysql_close()** - Закриває з'єднання з сервером MySQL

Приклад:

```
<?php
$link = mysql_connect('localhost', 'mysql_user', 'mysql_password');
if (!$link) {
   die('Помилка з’єднання: ' . mysql_error());
}
echo 'Успішне з’єднання';
mysql_close($link);
?>
```

**mysql_result ()** - отримати необхідний елемент з набору записів;

Приклад:

```
<?php
$link = mysql_connect('localhost', 'mysql_user', 'mysql_password');

$result = mysql_query('SELECT name FROM work.employee');
}
echo mysql_result($result, 2);

mysql_close($link);
?>
```

**mysql_fetch_array ()** - занести запис в масив;

Приклад:

```
<?php
mysql_connect("localhost", "mysql_user", "mysql_password") ;

mysql_select_db("mydb");

$result = mysql_query("SELECT id, name FROM mytable");

while ($row = mysql_fetch_array($result, MYSQL_BOTH)) {
```

```
   printf ("ID: %s  Имя: %s", $row[0], $row["name"]);
}

?>
```

**mysql_fetch_row ()** - занести запис в масив;

```
<?php
$result = mysql_query("SELECT id,email FROM people WHERE id = '42'");
$row = mysql_fetch_row($result);

echo $row[0]; // 42
echo $row[1]; // email
?>
```

**mysql_fetch_assoc ()** - занести запис в асоціативний масив;

Приклад:

```
$result = mysql_query($sql);

while ($row = mysql_fetch_assoc($result)) {
   echo $row["userid"];
   echo $row["fullname"];
   echo $row["userstatus"];
}
```

**mysql_fetch_object ()** - занести запис в об'єкт.

Приклад:

```
<?php
mysql_connect("hostname", "user", "password");
mysql_select_db("mydb");
$result = mysql_query("select * from mytable");
while ($row = mysql_fetch_object($result)) {
   echo $row->user_id;
   echo $row->fullname;
}
mysql_free_result($result);
?>
```

**mysql_query()** - Посилає запит MySQL,

Приклад:

```
$query = "SELECT name, address, age FROM friends
   WHERE name='Mike’”;
$result = mysql_query($query);
```

**mysql_field_len()** - Повертає довжину зазначеного поля.

Приклад:

```
<?php
$result = mysql_query("SELECT id,email FROM people WHERE id = '42'");
```

```
$length = mysql_field_len($result, 0);
echo $length;
?>
```

**mysql_error()** - Повертає текст помилки останньої операції з MySQL

Приклад:

```
<?php
$link = mysql_connect("localhost", "mysql_user", "mysql_password");

mysql_select_db("nonexistentdb", $link);
echo mysql_errno($link) . ": " . mysql_error($link). "\n";

mysql_select_db("kossu", $link);
mysql_query("SELECT * FROM nonexistenttable", $link);
echo mysql_errno($link) . ": " . mysql_error($link) . "\n";
?>
```

***21. PHP. Функції мови PHP***

Функція — це частина програмного коду, названа унікальним іменем (з точністю до регістра букв). Основне призначення функції — вирішення визначеної задачі. Виклик функції проходить від імені в різних місцях програми, що дозволяє багаторазово виклнувати фрагмент з вказаним іменем.

Очевидний позитивний ефект такого рішення, складається в тому, що блок кода пишеться тільки раз, а потім міняється коли це необхідно.

Імя функції може складатись із латинських букв, цифр і знаків підкреслювання. Починатись з цифри воно не повинно.

Синтаксис:

*function імя(аргументи)*

*{*

*тіло функції*

*}*

Викликана функція буде містити таку ж кількість аргументів, як і в заявці.

Приклад

function foo($arg_1,$arg_2,$arg_n,) {

echo “Птиклад.\n”;

return $val;

}

В прикладі присутні наступні елементи

foo — імя функції;

$arg_1,$arg_2,$arg_n — аргументи функції;

$val — повернення значень функції.

**Вбудовані функції(часто використовувані):**

**var_dump** - виводить інформацію про зміінну
**print** - виводить повідомлення в однин або більше рядок
**print_r** - виводить легку для читання інформацію про змінну
**die** - Припиняє дію роботи
**define** - Визначає іменовану константу
**Array** - використовується спеціально для створення масивів. При цьому вона дозволяє створювати порожні масиви.
**asort** - сортує масив, так, щоб його значення йшли в алфавітному (якщо це рядки) або в зростаючому (для чисел) порядку.

**in_array** - повертає true, якщо елемент зі значенням $ val присутній в масиві $ arr.
**isset** - Визначає, чи була встановлена змінна значенням, відмінним від NULL
**is_array** - Визначає, чи є змінна масивом
**is_null** - Перевіряє, чи є значення змінної рівним NULL
**strlen** - Повертає довжину рядка

***22.PHP. Класи і модулі в PHP.***

Класс являється основним поняттям обєктно–орієнтованого програмування.

Класс — це опис методів і властивостей. Класс не створюється і не використовується в програмі.

Обєкт це змінна, яка має властивості і методи, опис в класі, від якого він створюється. Можна створити декілюка обєктів від одного класу.

Конструктор — це метод обєкта, який викликається автоматично прикожному створені обєкта.

Як правило конструктори використовуються для присвоєння першопочаткових значень властивостям обєкта. І хоча конструктор появився в PHP давно, в PHP 5 була змінена схема найменування конструктора — метод**_ _construct()** являється тепер конструктор класу.

Аналогічно при знищені обєкта викликається спеціальний метод **_ _destract()–** деструктор класу. Необхідність в виклику деструкторів виникає лише при роботі з обєктами, які використовують великий обєм ресурсів, так як всі змінні і обєкти автоматично знищуються по завершені сценарія.

Властивість — це та сама змінна, тільки в середині обєкта.

Метод — це та сама функція, тільки в середині обєкта.

Обєкт — це структурована змінна, яка містить всю інформацію про деякий фізичний предмет.

Класс — це опис таких обєктів і дій, які можна виконувати.

Клас визначається за допомогою наступного синтаксису.

*class Імя_класу{*

*var $імя_властивості;*

*/*список властивостей*/*

*function імя_метода(){*

*/*визначення метода*/*

*}*

Класи являються як би шаблонами для реальних змінних. Змінна потрібного типу(об’єкт) створюється із класу з допомогою оператора new.

Створивши об’єкт, ми можемо застосувати до нього всі методи і отримати всі властивості визначені в описі класу.

Для цього використовується такий синтаксис.

*$імя_обєкта–>назва_властивості*

або

*$імя_обєкта–>назва_метода(список аргументів).*

Відзначу що перед назвою властивості чи метода знак $ не ставиться.

Приклад створення класу:

```
<?PHP

//визначення класа

class Людина {

//визначення властивості

public $імя;

public $вага;

public $вік;

public $стать;

public $колір;

//визначення методів

public function їжа()

{

echo $this–>імя.”їжа…”;

}
```

Тут ми можемо створити людину

```
<?PHP

$Іван = new Людина;

//ім’я

$Іван–>ім’я =”Іван”;

//який вік

$Іван–>вік =”26”;

$Іван–>стать=”мужчина”;

$Іван–>колір =”синій”;

$Іван–>вага =95;

$Іван–>їсть();
```

Модуль

Модуль це певний алгоритм, який працює в якомусь оточенні (cms / cmf / "движок") і виконує якийсь шматок покладених на нього обов'язків. фактично "движок" для модуля перетворюється в одну велику бібліотеку. можна написати 100 файлів з однаковими шматками коду і невеликою відмінністю, а можна все однакове винести в загальний код (cms / cmf / "движок"), а унікальний алгоритм залишиться вмістом модуля.

***23.PHP. Кукі(cookies). Програмування cookies. Приклади.***

в PHP cookie — це текстові файли, що зберігаються на комп'ютері клієнта, вони зберігаються в цілях відстеження. PHP прозоро підтримує HTTP файли cookie.

Для ідентифікації повертаються користувачів передбачено три етапи:

Файл на сервері відправляє набір файлів cookie у браузері. Наприклад, ім'я, вік або ідентифікаційний номер і т. д.

Браузер зберігає цю інформацію на локальному комп'ютері для майбутнього використання.

Коли в наступний раз браузер відправляє будь-який запит до веб-серверу, він відправляє ці файли на сервер і сервер використовує цю інформацію для ідентифікації користувача.

У цій статті ми розповімо, як встановлювати файли cookie, як працювати з ними і як їх видаляти.

Приклад створення "cookie" в мові програмування PHP:

В PHP для цього використовують спеціальну функцію, це setcookie ($name, $value, $time).

яка приймає три аргументи:

$name — назва куки, може люба бути

$value — значення

$time — час існування куки, звісно що не потрібно виставляти час який вже пройшов, оскільки кука буде не дійсна.

Для того щоб виставити тривалість "cookie" на одну добу, потрібно додати в функції time() вісімдесять шість тисяч чотириста. Тоді появиться вказівка, що тривалість "cookie" одна доба.

$ _COOKIE - це глобальний масив, який містить всі cookies, які сервер встановив на стороні користувача.

Приклади встановлення "cookie":

*setcookie ( 'name', 'value', 'time'()+86400);*

*setcookie("login", "pupkin", time() + 3600 * 24 * 7);*

*setcookie("password", "qwerty", time() + 3600 * 24 * 7);*

Після того, як "cookie" встановлені, при кожному відвідуваннні користувачем сторінки на сайті, PHP інтерпретатор формує глобальний масив $ _COOKIE, в котрому ключами є назви "cookie", за якими зберігаються їхні значення.

***24.PHP. Сесії***

Сесії і призначені для зберігання відомостей про користувачів при переходах між кількома сторінками. При використанні сесій дані зберігаються у тимчасових файлах на сервері. Використання сесій дуже зручно і виправдано в таких додатках як Інтернет-магазини, форуми, дошки оголошень, коли, по-перше, необхідно зберігати інформацію про користувачів протягом певного періоду часу, а, по-друге, своєчасно надавати користувачеві нову інформацію.

$ _SESSION – це глобальний масив, який містить всі змінні сесії поточного користувача.

Приклад Сесії:

```
<?
  session_start ();
  $ _SESSION ['Username'] = "maksim";
  echo 'Привіт,'. $ _SESSION ['username'].<br>;
?>
  <a href="page2.php"> На наступну сторінку </ a>
```

Після цього, користувач maksim натискає на посилання і потрапляє на сторінку page2.php, код якої наведено в лістингу:

```
<?
  session_start ();
  echo $ _SESSION ['username']. ', Ти прийшов на іншу сторінку цього сайту! ';
  echo ("<br>");
?>
  <a href="page3.php"> На наступну сторінку </ a>
```

При натисканні на посилання, користувач потрапляє на сторінку page3.php, при цьому відбувається розреєстрації сеансовом змінної і знищення сесії. Відповідний код реалізації приведений в лістингу:

```
<?
  session_start ();
  unset ($ _SESSION ['username']); / / разрегістріровалі змінну
  echo 'Привіт,'. $ _SESSION ['username'];
  / * Тепер ім'я користувача вже не виводиться * /
  session_destroy (); / / знищуємо сесію
```

***25. jQuery. Основи jQuery. Функція $() бібліотеки. Завантаження документа DOM. Події ready(). Використання альтернативної нотації. Обгортка jQuery. Допоміжні функції. Розширення jQuery. Сполучення jQuery з іншими бібліотеками.***

jQuery - це бібліотека JavaScript, заснована на принципі «пиши менше, роби більше». Це не мова програмування, а інструмент, який спрощує написання спільних завдань JavaScript. Крім того, jQuery володіє кроссбраузерною сумісністю, а значить, ви можете бути впевнені в тому, що будь-який сучасний браузер коректно відобразить вивід програми.

Селектори повідомляють jQuery, з яким елементом потрібно працювати. Більшість селектор jQuery схожі на CSS, але все ж вони мають деякі особливості. Щоб отримати доступ до селектору, використовуйте функцію jQuery; $(): $("selector")

Нижче перераховані деякі популярні селектори:
$ ( «*»): Wildcard, вибирає всі елементи сторінки.
$ (This): Current, вибирає поточний елемент.
$ ( «P»): Tag, вибирає кожен екземпляр тега <p>.
$ ( «. Example»): Class, вибирає кожен елемент класу example.
$ ( «# Example»): ID, вибирає один екземпляр унікального ID example.
$ ( «[Type = 'text']»): Attribute, вибирає будь-який елемент з text атрибута type.
$ ( «P: first-of-type»): Pseudo Element, вибирає перший тег <p>.

У загальних рисах, jQuery використовується для з'єднання з HTML-елементами в браузері за допомогою DOM. Document Object Model (DOM) - це метод, за допомогою якого JavaScript (і JQuery) взаємодіє з HTML в браузері. Кожен HTML-елемент в DOM вважається нодою - це об'єкт, з яким може взаємодіяти JavaScript. Ці об'єкти розташовані у вигляді дерева, де кожен вкладений елемент є гілкою, а <html> - коренем. JavaScript може додавати, видаляти і змінювати такі елементи.

Подія - це будь-яка взаємодія користувача з браузером (зазвичай за допомогою миші або клавіатури). Подія ready спрацьовує коли документ готовий для відображення. Наприклад:

```
$(document).ready(function() {
$("#demo").html("Hello, World!");
});
```

Код наведений вище запустить подію, яка відобразить текст в елементі з id=demo.

У jQuery 3 поведінка методу data () була трохи змінена, щоб він підходив під специфікації Dataset API. Тепер він переводить імена властивостей у альтернативну (верблюжу) нотацію:

При використанні старої версії Ви матимете такий результат: {custom-property: "Hello World"}

У jQuery 3 Ви отримаєте: {customProperty: "Hello World"}

В jQuery величезне значення відіграє об'єкт, іменований «обгорткою», або «обгортаючим набором». Він являє собою набір елементів, які були обрані для подальшої обробки. Як тільки ви створите код jQuery, ви автоматично викличте готовий до роботи об'єкт jQuery. Обрані вами елементи будуть «обгорнуті» в об'єкт jQuery, над яким можна здійснювати різні дії. Функції, застосовані до набору, будуть циклічно застосовуватися до кожного з його елементів. Об'єкт jQuery виконує всі інструкції, які передаються до нього через ланцюжок операторів, і зупиняє циклічне виконання тоді, коли справа доходить до останнього об'єкта в наборі.

Допоміжні функції jQuery - це функції належать простору імен jquery ( $), але не впливає на обгорнутий набір елементів (на відміну від методів jquery). Такі функції визначені не глобальному просторі імен ( window), а в просторі імен jquery ( $). Допоміжні функції, слідуючи логіки, не повинні впливати на DOM-об'єкти (залишимо це методам), тобто допоміжні функції впливають на об'єкти js або виконують будь-які

специфічні операції. Дані функції, як правило, корисні при написанні самих методів. Їх є достатньо багато, і ми також можемо створювати їх самостійно. Найпопулярнішими з них є:
$.trim(value) - видаляє початкові і кінцеві пробільні символи з рядка value і повертає результат
$.each(container,callback) - дозволяє обійти елементи масиву і об'єкти jquery, та викликати для кожного елемента функцію (callback).
**$.**inArray(value,array) - дозволяє знайти в масиві потрібне вам значення, якщо воно є, а також його місце розташування.

Щоб втримати ядро бібліотеки jQuery в розумних межах, туди включені тільки найбільш універсальні методи і функції. Але окрім ядра є велика кількість розширень, які допомагають вирішити нестандартні ситуації. Найпопулярнішими розширеннями для jQuery можна вважати ajax який допомагає відправляти запити на сервер без перезагрузки сторінки, jQuery UI – який має набір функцій анімаційних ефектів і віджетів оформлення, а також багато інтеграційних розширень створених для взаємодії з іншими популярними сервісами.

***26. jQuery. Вибір елементів. Визначення селектор. Визначення контексту. Звуження області пошуку за допомогою контексту. Базові селектори CSS. Селектори вибору нащадків, контейнерів і атрибутів. Групування селекторів.***

Одним з важливих функціональностей jQuery є вибірка елементів. Щоб щось робити з елементами, маніпулювати ними, застосовувати до них методи jQuery, нам треба спочатку їх отримати. Бібліотека надає нам зручний спосіб вибору елементів, заснований на селекторах. Нам досить передати в функцію jQuery селектор і ми можемо отримати потрібний нам елемент, який відповідає даному селектору. Наприклад, якщо необхідно отримати всі еелементи img, то ми можемо використовувати такий вираз: $("img"). В даному випадку "img" буде виступати в якості селектора.

Для обмеження пошуку селектора використовують контекст. З його допомогою можна обрати не всі елементи, які відповідають селектору, а тільки ті які, наприклад знаходяться всередині іншого елементу:

```
<div id="menu">
    <p><a href="1.html" class="open">Ссылка 1</a></p>
    <p><a href="2.html" class="open">Ссылка 2</a></p>
</div>
<p><a href="3.html" class="open">Ссылка 3</a></p>
```

На сторінці у нас три посилання, у всіх у них визначено один і той же клас, але дві з них знаходяться в елементі div і саме їх ми хочемо отримати. Тоді ми використовуємо вираз $(".open", "div#menu") - тут другий параметр-селектор буде контекстом вибірки.

Подібно до написання css стилів функціонують і селектори jQuery. Вибір усіх елементів сторінки ($ ( "*")), пошук елементів по назві тега ($ ( "p") ), вибір за значенням атрибута id ($ ( "# name")), вибір елементів певного класу ($ ( ". Class")), також є можливість формувати вибірку по кільком характеристиках ($ ( "Selector1, selector2, selectorN")).

Також є можливість обирати всіх нащадків селектора, здійснювати пошук по атрибутах, обирати тільки елементи, з якими виконується умова:

```
$('div span');          // вибір всіх span елементів в елементах div
$('div > span');        // вибір всіх span елементів в елементах div, де span є прямим нащадком div'a
$('div, span');         // вибір всіх div и span елементів (групування)
$('span + img');        // вибір всіх img елементів перед якими йдуть span елементи
$('span ~ img');        // вибір всіх img елементів після першого елемента span
$("div[id]");               // вибір всіх div з атрибутом id
$("div[title!='my']");      // вибор всіх div з атрибутом title не рівного my
```

Варіантів селекторів може бути безліч, головне зрозуміти основні правила їх формування.

***27. jQuery. Вибір елементів по позиції. CSS і нестандартні селектори jQuery. Одержання обгорненого набору. Перетворення набору обраних елементів з урахуванням взаємин. Одержання зрізів обгорненого набору елементів. Фільтрація елементів.***

jQuery - це чудовий JavaScript Framework, який підкуповує своєю простотою в розумінні і зручністю в користуванні**.**

Різноманітність селекторів jQuery величезне, можна обирати елементи по класах, іменах, атрибутах, тегах, вибирати дочірні, та тільки ті, для яких виконується певно умова. Також можна обирати техи по їх позиції в документі DOM:

```
$('#banner').prev();     // вибір попереднього елемента від знайденого
$('#banner').next();     // вибір наступного елемента від знайденого
$('div:first');      // вибираємо перший div в DOM
$('div:not(.red)');          // вибираємо div'и у яких немає классу red
$('div:eq(N)');              // вибираємо div під номером N в DOM
$('div:contains(text)'); // вибираємо div'и які містять текст
$('div:has(p)');         // вибираємо div'и які містять p
$("form :checkbox:checked"); // вибір всіх вибраних чекбоксів
```

При написанні селектора ми можемо отримати як один елемент, так і набір елементів. Для маніпуляцій з набором елементів в jQuery передбачено ряд функцій:

фільтрація набору

.eq () - повертає елемент, що йде під заданим номером в наборі обраних елементів.

.filter () - фільтрує набір обраних елементів за допомогою заданого селектора або функції.

.first () - повертає перший елемент в наборі.

.has () - фільтрує набір обраних елементів, залишаючи ті, які мають нащадків, відповідних селектору.

.is () - перевіряє, чи міститься в наборі, хоча б один елемент задовольняє заданому селектору.

.last () - повертає останній елемент в наборі.

.not () - для отримання елементів, що не відповідають заданим умовам.

.slice () - для отримання елементів з індексами з певної області (наприклад від 0 до 5).

обхід набору

.each () - викликає задану функцію для кожного елемента набору.

.map () - викликає задану функцію для кожного елемента набору, і в підсумку створює новий набір, складений з значень, повернутих цією функцією.

інші методи

.add () - додає задані елементи в набір.

.andSelf () - додає елементи з попереднього набору, до поточного (мова йде про одному ланцюжку методів).

.contents () - знаходить всі дочірні елементи у вибраних елементів. В результат, крім елементів, включається і текст.

.end () - повертає попередній набір елементів в поточній ланцюжку методів.

***28. jQuery. Робота з DOM-об'єктами. Створення елементів DOM. Переміщення вниз по дереву, вгору по дереву та по дереву в межах одного ієрархічного рівня.***

Бібліотека jQuery пропонує свій спосіб реєстрації події завантаження сторінки. Відповідний код представлений в прикладі нижче:

```
$(document).ready(function(e) {
    //
});
```

У цьому сценарії ми передаємо змінну document функції $ () як аргумент і викликаємо метод ready () , передаючи йому функцію, яку хочемо виконати після закінчення завантаження і готовності DOM до роботи. Можете помістити цей елемент script в будь-яке місце документа, будучи впевненим у тому, що jQuery не допустить дострокового виконання функції.

Функція function (), що передається методу ready (), буде викликана лише після завантаження документа, але не раніше, ніж завершиться побудова DOM.

Одна з найважливіших областей застосування функціональності jQuery - це вибір елементів DOM. Як показано, як здійснити вибір всіх елементів з класом t:

```
var tList = $(“.t”);
```

Бібліотека jQuery не підміняє собою DOM-модель, а лише набагато полегшує роботу з нею. Об'єкти HTMLElement можна використовувати, як і раніше, але jQuery спрощує перехід від об'єктів jQuery до об'єктів DOM і назад. На мою думку, та простота, з якою можна переходити від традиційної DOM-моделі до об'єктів jQuery і назад і яка дозволяє підтримувати зворотну сумісність зі сценаріями і бібліотеками, не пов'язаними з jQuery, є наслідком елегантності побудови самої бібліотеки jQuery.

Об'єкти jQuery можна створювати, передаючи об'єкт або масив об'єктів HTMLElement функції $ () як аргумент. Такий спосіб зручний при роботі з JavaScript-кодом, що не орієнтованим на jQuery, або в ситуаціях, коли jQuery відкриває доступ до базових DOM-об'єктів, наприклад при обробці подій.

Об'єкт jQuery може розглядатися і як масив об'єктів HTMLElement. Це означає, що поряд з розвиненими засобами, пропонованими бібліотекою jQuery, як і раніше можна використовувати об'єкти DOM. Можете використовувати властивість length або метод size () для визначення числа елементів, які входять в набір обраних елементів, що міститься в об'єкті jQuery, і отримувати доступ до окремих DOM-об'єктів, використовуючи індексний нотацію масивів (дужки [i]).

Для вилучення об'єктів HTMLElement з об'єкта jQuery, що розглядається як масив, можна використовувати метод toArray (). Особисто я намагаюся працювати тільки з об'єктами jQuery, але іноді, наприклад в разі успадкованого коду, в якому можливості jQuery не використовуються, зручніше працювати безпосередньо з DOM-об'єктами.

Приклад перерахування вмісту об'єкта jQuery з метою доступу до вмісту в ньому елементів HTMLElement наведено нижче:

```
var elems = $('img:odd');
for (var i = 0; i < elems.length; i++)
{
  console.log("Элемент: " + elems[i].tagName + " Ссылка: " + elems[i].src);
}
```

Метод each () дозволяє визначити функцію, яка буде виконана для кожного з DOM-об'єктів, що містяться в об'єкті jQuery. Відповідний приклад наведено нижче:

```
$('img:odd').each(function(index, element) {
  console.log("Элемент: " + element.tagName + " Ссылка: " + element.src);
});
```

Бібліотека jQuery передає зазначеної функції два аргументи. Перший з них - це індекс елемента в колекції, а другий - власне об'єкт елемента. В даному прикладі ми виводимо на консоль ім'я дескриптора і значення властивості src, отримуючи при цьому той же результат, що і в попередньому прикладі.

Серед селекторів передбачені і такі, які дозволяють переміщатись по ієрархії об`єктів DOM:

```
$('#banner').prev();     // вибір попереднього елемента від знайденого
$('#banner').next();     // вибір наступного елемента від знайденого
$('div:first');      // вибираємо перший div в DOM
$('div:eq(N)');              // вибираємо div під номером N в DOM
$('div span');           // вибір всіх span елементів в елементах div
$('div > span');         // вибір всіх span елементів в елементах div, де span є прямим нащадком div'a
```

***29. jQuery. Створення нових елементів HTML з використанням функції $(). Створення нових елементів шляхом клонування існуючих. Створення елементів засобами DOM API. Вставка дочірніх елементів і елементів нащадків. Вставка вмісту в початок та кінець елементів.***

На сьогоднішній день, jQuery - це найпопулярніша JavaScirpt бібліотека, вирішальна купу проблем. Ми можемо без особливих зусиль маніпулювати контентом - замінювати, вставляти, видаляти, створювати анімацію і т.д.

Метод Append призначений для створення і вставки нового елемента в уже існуючий контекст. Технічно, новий елемент вставляється прямо перед закривається тегом батька.

```
$('body').append('<div>');
```

Метод clone () дозволяє створювати нові елементи на основі існуючих. Виклик цього методу призводить до клонування кожного з елементів, що містяться в об'єкті jQuery, разом з усіма їх елементами-нащадками. Відповідний приклад наведено нижче:

```
var newElems = $('div.dcell').clone();
```

Для створення об'єктів HTMLElement можна використовувати безпосередньо програмний інтерфейс DOM, що, власне кажучи, jQuery і робить замість нас, коли ви залучаєте для цієї мети інші методики. Ось як виглядає створення елементу, та додавання йому дочірнього елементу використовуючи засоби DOM API :

```
$(function() {

var divElem = document.createElement("div");
divElem.classList.add("dcell");

var imgElem = document.createElement("img");
imgElem.src = "http://professorweb.ru/downloads/jquery/lily.png";

    divElem.appendChild(imgElem);

});
```

Також є передбачені функції, які додають вмістиме в початок, або в кінець елемента:

```
$('body').prepend('<p>new text!</p>'); - додає тег p з його вмістимим в початок елемента body
$(".list").append("<li class='item'>Тест</li>"); - додає тег li з його вмістимим в кінець елементів класу list
```

***30. jQuery. Вставка одних і тих же елементів в різні місця документа. Вставка батьківських елементів і елементів предків. Вставка сестринських елементів***

Один і той же набір нових елементів може бути вставлений в документ лише один раз. Після цього будь-яка спроба використання цього набору в якості аргументу при виклику методів, призначених для вставки в DOM нових вузлів, призведе не до дублювання цього набору, а до його переміщення. Для вирішення цієї проблеми слід створити копії елементів, які підлягають вставці, за допомогою методу clone (). Переглянутий варіант сценарію наведено нижче:

```
$('#row1').append(newElems);
$('#row2').prepend(newElems.clone());
```

В бібліотеці є команди для створення нових елементів, і їх вставки у вже істуючі як дочірні, батьківські або сестринські елементи:

prepend (HTML/ jQuery/ HTMLElement []) – Вставляє зазначені елементи в якості перших дочірніх елементів в усі вибрані елементи

appendTo (jQuery) – Вставляє елементи, що містяться в об'єкті jQuery, в якості останніх дочірніх елементів в елементи, задані аргументом

Як ви можете здогадатися, jQuery також пропонує комплекс методів для вставки елементів в документ як сіблінгов (сестринських елементів) існуючих елементів.

after(HTML) – Вставляє зазначені елементи в якості наступних сестринських елементів для кожного елемента в об'єкті jQuery

before(HTML) – Вставляє зазначені елементи в якості попередніх сестринських елементів для кожного елемента в об'єкті jQuery

insertAfter(HTML) – Вставляє елементи в об'єкт jQuery як наступних сестринських елементів для кожного елемента, зазначеного в аргументі

insertBefore(HTML) – Вставляє елементи в об'єкт jQuery як попередніх сестринських елементів для кожного елемента, зазначеного в аргументі

jQuery надає цілий набір методів для вставки елементів в якості батьківських або елементів-предків для інших елементів. Це явище відоме як враппінг (бо один елемент "обгорнутий" іншим)

wrap(HTML) – Обертає зазначені елементи навколо кожного елемента в об'єкті jQuery

wrapAll(HTML) – Обертає зазначені елементи навколо набору елементів в об'єкті jQuery(як одну групу)

wrapInner(HTML) – Обертає зазначені елементи навколо змісту елементів в об'єкті jQuery

wrap(function)\wrapInner(function) – Обертає елементи динамічно з використанням функції

Якщо ви робите враппінг, ви можете призначити кілька елементів в якості аргументів, але ви повинні переконатися, що є тільки один внутрішній елемент. В іншому випадку jQuery не зможе зрозуміти, що робити. Це означає, що кожен елемент в аргументі методу повинен мати не більше одного з батьків і не більше одного дочірнього елемента.

***31. jQuery. Заміна HTML-розмітки або тексту.Переміщення, копіювання, очищення та зміна елементів. Метод unwrap().***

Для заміни HTML-розмітки або тексту в jquery використовується метод заміни replace(), який шукає рядок для заданого значення, або регулярного виразу, і повертає новий рядок, в якому вказані значення замінюються.

Видалення елементів зі сторінки здійснюється за допомогою методів empty(), remove(), detach() і unwap().

Заміна елементів на сторінці здійснюється за допомогою методів replaceWith() та replaceAll().

Для переміщення елементів на сторонці немає спеціального методу, але можна виділити елементи, які хочемо перемістити і визвати один з методів append(), appendTo()  або prepend().

Метод .unwrap() протилежний методу .wrap(). Видаляє батьківські елементи у вибраних елементів, при цьому сам вміст залишається на місці.

За допомогою методів jQuery завдання обходу і маніпуляції елементами DOM вирішуються досить простими та інтуїтивно зрозумілими способами.

Для обходу і маніпуляції елементами DOM в бібліотеці jQuery існують наступні методи: text(),  attr(), append(), wrap(), remove(), clone() та багато інших.

-Метод text() може читати (отримувати) або писати (замінювати) текст всередині даного елемента HTML.

-Функція attr() може читати і модифікувати атрибути HTML-елементів

-Метод append() вставляє новий HTML-код в кінець вибраного HTML-елмента. Новий HTML об’єднується з HTML кодом,  який був у елемента до цього.

-Метод wrap() може огорнути вибраний елемент HTML в інший елемент HTML

-Метод remove() видаляю вибраний елемент HTML з дерева DOM

-Метод clone() клонує (копіює) обраний елемент, що дозволяє продублювати потрібний елемент в іншому місці дерева DOM.

***32. jQuery. Збереження власних даних в елементах.***

Збереження даних в елементах можливо зо допомогою метода .data().

Він дозволяє прив'язувати власні змінні до любих елементів сторінки. Метод має декілька варіантів використання:

.data(key, value) – встановлює змінну key із значенням value всім вибраним елементам сторінки.

.data({key1: value1, key2:value2, …}) – встановлює групу змінних key1,key2,… із значеннями value1,value2,…, всім вибраним елементам сторінки

.data(key) – повертає значення змінної key у першого вибраного елемента

.data() – повертає об'єкт із всіма змінними, які прикріплені до першого з вибраних елементів.

Для роботи з класами в jQuery є певні методи, які дозволяють присвоїти(створити) клас для елемента, отримати значення класу або його ім'я, додати клас, видалити його, замінити на інший і т.д

Метод .attr('class','значення класу') дозволяє присвоїти(створити) клас необхідним елементам.

Метод addClass('ім'я класу') дозволяє добавити один або декілька класів до вибраних елементів.

Метод removeClass('ім'я класу') дозволяє видалити класи у вибраних елементів.

Метод toggleClass('ім'я класу') дозволяє замінювати клас або декілька класів (розділених між собою відступом). Під заміною класа мається на увазі виконання однієї з наступних дій: видалення класа, якщо він є у елемента, додавання класу, якщо його немає.

Отримання значення кінцевої властивості CSS, яке застосовується до елемента, в jQuery виконується за допомогою метода .css(), також цей метод використовується щоб встановити стиль вибраному елементу .ccs('назва властивості', значення) . Видалити певний стиль у елемента можна за допомогою такої конструкції - $('img').css('height', '') (присвоїти пустий рядок).

***33.jQuery. Робота з елементами форми.***

Для роботи з елементами форми (отримання і зміни значень наприклад елемента input) використовується метод val().

Val() – повертає значення першого з елементів, які знаходяться в об’єкті jQuery.

Val(значення) – змінює значення всіх елементів, які знаходяться в об’єкті jQuery.

Val(функція) – змінює значення всіх елментів, які знаходяться в об’єкті jQuery за допомогою функції.

В бібліотеці jQuery є перелік основних функцій-подій:

on() - встановлює обробники подій на обрані елементи сторінки

off() - Видаляє з обраних елементів сторінки обробники подій.

click() - Встановлює обробник “кліка” мишею по елементу, або, запускає цю подію

dblclick() - Встановлює обробник подвійного “кліка” по елементу

toggle() - По черзі виконує одну з декількох заданих дій

focus() - Встановлює обробник отримання фокусу, або, запускає цю подію

blur() - Встановлює обробник втрати фокусу, або, запускає цю подію

Для взаємодії з елементами сторінки і самим браузером в jQuery існує велика кількість готових об’єктів. Всі об’єкти разом складають об’єктну модель браузера (BOM – Browser Object Model). В самому верху цієї моделі знаходиться глобальний об’єкт window,  він представляє собою одне з вікон або вкладку браузера. Доступ до різних об’єктів вікна браузера відбувається за допомогою наступних основних об’єктів: navigator, history, location, screen, document і т.д. Серед всіх цих об’єктів найбільшу зацікавленість і значимість для розробника має об’єкт document, який являється коренем об’єктної моделі документа (DOM – Document Object Model).

Об’єкт document являє собою HTML документ, завантажений у вікно(вкладку) браузера. За допомогою властивостей і методів даного об’єкта ми можемо отримати доступ до вмістимого HTML-документа, а також змінювати його вмістиме, структуру і оформлення.

***34. jQuery. Модель подій jQuery.***

Однією з найважливіших функцій jQuery є реакція на різні дії користувача, будь те натискання на кнопку, чи гіперпосилання, просте наведення мишки на який-небудь елемент і т.д. Усе, що користувач може робити на сторінці, називається подією, як і все, що автоматично відбувається в браузері. Наприкад, завантаження сторінки – це подія.

В бібліотеці jQuery є дуже велика кількість оброблювачів подій, наведемо ті, які використовуються найчастіше:

- .change() – встановлює обробник змінення заданого елементу форми
- .click() – встановлює обробник ‘кліка’ мишкою по елементу
- .hover() – встановлює обробник двох подій: mouseenter, mouseleave
- .ready() – встановлює обробник готовності дерева DOM
- .resize() – встановлює обробник зміни розмірів вікна браузера
- .scroll() – встановлює обробник прокрутки елементів документа
- .toggle() – по черзі виконує одну із декількох  заданих дій
- І багато інших

Підключення цих подій здійснюється шляхом  додавання потрібних обробників до елементів DOM. Наприклад, $(“#link”).hover(function(){}) – додали обробник події mouseenter і mouseleave до елемента DOM з id = “link”.

Для видалення оброблювачів подій використовуються наступні методи:

die(), off(), event.preventDefault()

Для того, щоб відмінити дії браузера по замовчуванню існує два способи:

- Основний спосіб – це скористатись об’єктом подій. Для відміни дії браузера існує стандартний метод event.preventDefault()
- Якщо ж обробник прописаний через .on() (а не через addEventListener), то можна просто повернути значення false з обробника.

***35.jQuery. Установка разового обробника подій.***

В бібліотеці jQuery існує велика кількість стандартних подій, які можна реалізовувати за допомогою різних методів. За допомогою цих же методів можна встановити разовий обробник подій, тобто щоб певна подія виконувалась тільки один раз.

Це можна реалізувати за допомогою двох методів, а саме: методу, який встановлює обробник події(наприклад .on()) та методу event.preventDefault(), який не дасть виконати події за замовчування.

Таким чином комбінуючи методи встановлення обробника подій та event.preventDefault() ми можемо зробити одноразовий обробник подій.

Одним з методів встановлення оброблювачів подій є метод .bind(), який встановлює оброблювач події на вибрані елементи.

Метод .live() також встановлює обробник подій на вибрані елементи, але має одну важливу особливість, що відрізняє його від .bind(): якщо на сторінку будуть вставлені нові елементи, які відповідають поточному селектору, то вони також будуть реагувати на задані події.

Запуск обробника подій відбувається автоматично за відповідних умов, залежить від того, обробник якої події ми встановили, наприклад, якщо ми встановили обробник події .hover() на якийсь елемент нашої сторінки і написали, що саме повинно статися, коли ця подія виконається, то в цьому випадку функція спрацює, коли ми наведемо мишкою на відповідний елемент сторінки. Всередині самої функції ми можемо прописати методи, які можуть напряму змінювати елементи нашої сторінки.

Такий обробник належить до типу подій, які викликаються вручну,  окрім цього є обробники, які викликаються автоматично на таких подіях як, наприклад, завантаження сторінки(window, .onload()), завантаження дерева DOM(document) і т.п.

***36. jQuery. Об'єкт Event.***

Об'єкт event – це об'єкт , який вміщує в собі дані про події, які трапилися. Екземпляр цього об'єкта передається у всі обробники подій, які були встановлені методам бібліотеки jQuery. Об'єкт event відрізняється від стандартних об'єктів подій, які отримують обробники, встановлені звичайними методами . Окрім цього в event добавлені додаткові поля та методи.

Наступні поля гарантовано присутні в кожного об'єкті подій: altKey, attrChange, attrName, bubbles, cancelable, charCode, clientX, clientY, ctrlKey, currentTarget, data, fromElement, keyCode, offsetX, offsetY, toElement, view та багато інших.

event.target – DOM-елемент, який є джерелом події

event.type – тип події

event.preventDefault() – відміняє виконання події

event.pageX, event.pageY – координати курсора мишки відносно лівого верхнього кута документа

і т.д

Метод triggerHandler(event Type [, extraParameters]) – метод, який запускає всі обробники подій, прикріплені до елементів на стронці.

triggerHandler(event Type, [, extraParameters])

де, event Type – рядок з назвою події, наприклад .hover(), .focus()

extraParameters – масив з додатковами параметрами , які будуть передані в обробник.

Метод triggerHandler() працює майже так само, як і метод .trigger(), але має деякі відмінності. Метод triggerHandler() не викликає стандартних подій (наприклад відправка форми). В той час як .trigger() діє на всі підходящі елементи в об'єкті jQuery, triggerHandler() спрацьовує тільки для першого знайденого елемента.

Метод .triggerHandler() повертає те, що і метод обробник, який спрацьовую в результаті "вистріла" події. Якщо ні один з обробників не буду визваний, то буде повернуто undefined.

***37. jQuery. Використання прямих методів для роботи з подіями браузера. Використання прямих методів для роботи з подіями миші.***

У бібліотеці jQuery визначено ряд так званих прямих методів, що дозволяють пов'язувати часто використовувані події з функціями-обробниками. Нижче ці функції описані з аргументом function, який задає функцію-обробник для даної події. Це найбільш поширений спосіб роботи з подіями, який вимагає менших зусиль по набору тексту і більш чітко показує, з яким саме подією пов'язується обробник.

**Приклад використання одного з прямих методів представлений нижче:**

```
$(function() {
   $('img').mouseenter(function() {
      $(this).css("border", "thick solid red");
   });
});
```

Це еквівалентно використанню методу bind () для прив'язки події mouseenter. Однак прямі методи можна використовувати і в якості аналогів методу trigger (). Для цього їх слід викликати без вказівки аргументу.

**Відповідний приклад наведено нижче:**

```
$(function() {
         $('img').bind("mouseenter", function() {
      $(this).css("border", "thick solid red");
   });
      $("<button>Запустити</button>").appendTo("#buttonDiv").click(function (e) {
      $('img').mouseenter();
      e.preventDefault();
   });
});
```

Тут додається кнопка (елемент button), після клацання на якій вибираються елементи img і викликаються їх обробники події mouseenter. Нижче перераховуються різні категорії прямих методів і подій, яким вони відповідають:

**<u>Події браузера</u>**

- error () Відповідає події error, яке спрацьовує при виникненні проблем із завантаженням зовнішніх ресурсів, наприклад зображень
- resize () Відповідає події resize, яке спрацьовує при зміні розміру вікна браузера
- scroll () Відповідає події scroll, яке спрацьовує при використанні смуг прокрутки

**<u>Події мишки</u>**

- click () Відповідає події click, яке спрацьовує, коли користувач виконує клацання мишею

- dbclick () Відповідає події dbclick, яке спрацьовує, коли користувач виконує подвійне клацання мишею
- focusin () Відповідає події focusin, яке спрацьовує при використанні смуг прокрутки
- focusout () Відповідає події focusout, яке спрацьовує, коли елемент втрачає фокус
- hover (), hover (функція, функція) Запускається, коли покажчик миші переміщається в область елемента або залишає її. Якщо вказана тільки одна функція, вона використовується в обох випадках
- mousedown () Відповідає події mousedown, яке спрацьовує при натисканні мишею над елементом
- mouseenter () Відповідає події mouseenter, яке спрацьовує при наведенні покажчика миші на область екрану, яку займає елементом
- mouseleave () Відповідає події mouseleave, яке спрацьовує, коли курсор миші покидає область екрану, яку займає елементом
- mousemove () Відповідає події mousemove, яке спрацьовує, коли курсор миші переміщається в межах області екрану, займаної елементом
- mouseout () Відповідає події mouseout, яке спрацьовує, коли курсор миші покидає область екрану, яку займає елементом
- mouseover () Відповідає події mouseover, яке спрацьовує, коли курсор миші знаходиться в області екрану, займаної елементом
- mouseup () Відповідає події mouseup, яке спрацьовує при відпуску раніше утримуючи кнопки миші над елементом

***38. jQuery. Використання прямих методів для роботи з подіями форми. Використання прямих методів для роботи з подіями клавіатури.***

У бібліотеці jQuery визначено ряд так званих прямих методів, що дозволяють пов'язувати часто використовувані події з функціями-обробниками. Нижче ці функції описані з аргументом function, який задає функцію-обробник для даної події. Це найбільш поширений спосіб роботи з подіями, який вимагає менших зусиль по набору тексту і більш чітко показує, з яким саме подією пов'язується обробник.

**Приклад використання одного з прямих методів представлений нижче:**

```
$(function() {
   $('img').mouseenter(function() {
      $(this).css("border", "thick solid red");
   });
});
```

Це еквівалентно використанню методу bind () для прив'язки події mouseenter. Однак прямі методи можна використовувати і в якості аналогів методу trigger (). Для цього їх слід викликати без вказівки аргументу.

**Відповідний приклад наведено нижче:**

```
$(function() {
         $('img').bind("mouseenter", function() {
      $(this).css("border", "thick solid red");
   });
      $("<button>Запустити</button>").appendTo("#buttonDiv").click(function (e) {
      $('img').mouseenter();
      e.preventDefault();
   });
});
```

Тут додається кнопка (елемент button), після клацання на якій вибираються елементи img і викликаються їх обробники події mouseenter. Нижче перераховуються різні категорії прямих методів і подій, яким вони відповідають:

**<u>Події форми</u>**

blur (функція) Відповідає події blur, яке спрацьовує, коли елемент втрачає фокус

change (функція) Відповідає події change, яке спрацьовує при зміні значення

елемента

focus (функція) Відповідає події focus, яке спрацьовує, коли елемент отримує

фокус

select (функція) Відповідає події select, яке спрацьовує при виборі користувачем значення елемента

submit (функція) Відповідає події submit, яке спрацьовує при відправці форми користувачем

**Події клавіатури**

keydown (функція) Відповідає події keydown, яке спрацьовує, коли користувач натискає кнопку на клавіатурі

keypress (функція) Відповідає події keypress, яке відбувається, коли користувач натискає і відпускає клавішу на клавіатурі

keyup (функція) Відповідає події keyup, яке спрацьовує, коли користувач відпускає клавішу на клавіатурі

***39. jQuery. Використання базових ефектів. Перемикання видимості елементів. Перемикання стану відображення елементів. Одностороннє перемикання видимості елементів. Анімація видимості елементів. Використання функцій зворотного виклику в ефектах. Анімаційні ефекти при зміні візуального стану елементів.***

**Використання базових ефектів**

Призначенням більшості базових ефектів є просте відображення чи приховування елементів. Методи, які використовуються для цього:

- hide() - приховує всі елементи, що містяться в об'єкті jQuery.
- show() - відображає всі елементи, що містяться в об'єкті jQuery
- toggle() - перемикає (відображає, якщо вони приховані, і приховує, якщо вони відображаються) видимість елементів, що містяться в об'єкті jQuery

Приклад використання методів show() і hide() без аргументів для створення простих ефектів:

```
$ ("<button>Приховати</button><button>Показати</button>")
        .appendTo(■#buttonDiv")
        .click(function(e) {
        if ($(e.target).textO =* "Приховати") {
                $(1#rowl div.dcell1).hide();
                } *lee {
                $('#rowl div.dcell').show();
                }
                e.preventDefault();

        });
```

**Перемикання видимості елементів.**

Для переводу елемента з видимого стану в невидимий і навпаки використовується метод toggle().

Приклад:

$(document).ready(function() {

$ (" <button>nepe^Переключити</button>" ) . appendTo ( "#buttonDiv")

.click(function(e) {

**$('div.dcell:first-child').toggle();**

e .preventDefault() ;

};

В цьому прикладі ми додаєм в документ кнопку після натискання якої викликається метод toggle(), змінюючи видимість тих елементів які є дочірними елементами.

**Одностороннє перемикання видимості елементів.**

Передача методу toggle() логічного(булевого)значення дозволяє ставити обмеження на можливі зміни стану видимості елементів. Якщо в якості аргументу передається true, тоді будуть відображатись лише приховані елементи. Передача false працює навпаки, видимі елементи приховуються а приховані не стають видимими. Приклад:

```
$(document).ready(function() {

$ ( "<button>nepe^KD4HTb</button>" ) . appendTo ( "#buttonDiv" )

.click(function(e) {

$('div.dcell:first-child').toggle(false);

e .preventDefault();

});

});
```

**Анімація видимості елементів.**

Процес відображення і приховування елементів можна анімувати, передаючи методу  show (), hide () або toggle () параметр, що задає тривалість анімації. В цьому випадку процес приховування або відображення елементів буде здійснюватися плавно протягом зазначеного періоду.

В цьому прикладі для задання тривалості анімаційного переходу ми використовуєм строковий параметр fast, за допомогою якого вказуєм що на перемикання видимості елементів img в документі відводиться 600 мс.

```
ript type="text/javascript">

$(document).ready(function() {

$("<button>Перемикнути</button>").appendTo("#buttonDiv")

.click(function(e) {

$('img') .toggle("fast", "linear");

e .preventDefault();

}>;

}>;
```

**Використання функцій зворотного виклику в ефектах. Анімаційні ефекти при зміні візуального стану елементів.**

Методам show(),  hide() і toggle() можна передавати в якості аргумента функцію яка буде викликана по закінченню анімації. Цю можливість можна використовувати для оновлення стану інших елементів.

Функція hideVisibleElements() використовує метод hide(), для анімації приховування видимого ряда елементів

```
function hideVisibleElement() {

$(visibleRow).hide("fast", showHiddenElement);

}
```

При цьому ми вказуєм імя функції яка повинна бути викликана во завершенню ефекта, в даному випадку – showHiddenElement.

***40. jQuery. Створення власних анімаційних ефектів. Поступове відображення та приховання елементів. Ефект масштабування. Ефект падіння. Ефект розсіювання.***

**Створення власних анімаційних ефектів.**.

.animate (properties, [duration], [easing], [callback]) - виконує анімацію за рахунок плавної зміни значень CSS-властивостей з можливістю вказання тривалості і стиля анімації

properties - список CSS-властивостей, які беруть участь в анімації і їх кінцевих значень. (Див. Опис нижче)

duration - тривалість виконання анімації. Може бути задана в мілісекундах або строковим значенням 'fast' або 'slow' (200 і 600 мілісекунд).

easing - зміна швидкості анімації (чи буде вона сповільнюється до кінця виконання або навпаки прискориться). (Див. Опис нижче)

callback - функція, яка буде викликана після завершення анімації.

```
$('div').animate(
  {
    opacity: "hide",
    height: "hide"
  },
5000);
```

Приховає div-елементи, за рахунок зменшення прозорості та зменшення висоти (згортанням) елемента.

**Поступове відображення та приховання елементів.**

.show () .hide ()

За допомогою цих функцій можна плавно показувати і приховувати вибрані елементи на сторінці, за рахунок зміни розміру і прозорості. Відзначимо, що після приховану елемента, його css-властивість display стає рівним none, а перед появою, воно отримує своє колишнє значення назад.

.show (duration, [callback]) .hide (duration, [callback]):

duration - тривалість виконання анімації (появи або приховування). Може бути задана в мілісекундах або строковим значенням 'fast' або 'slow' (200 і 600 мілісекунд). Якщо цей параметр не заданий, анімація буде відбуватися миттєво, елемент просто з'явиться / зникне

callback - функція, задана в якості обробника завершення анімації (появи або приховування). Їй не передається ніяких параметрів, однак, всередині функції, змінна this буде містити DOM-об'єкт аніміруемого елемента. Якщо таких елементів декілька, то обробник буде викликаний окремо, для кожного елемента.

**Ефект масштабування.**

**scale** - Ефект "Масштабування". За допомогою опції percent: відсотки Ви можете задати на скільки відсотків від поточного розміру необхідно зменшити або збільшити елемент.

```
$(document).ready(function() {
    $("#twitter").show( "scale",
    {percent: 100, }, 2000 );
```

});

**Ефект падіння.**

Droppable() - елементи до яких був застосований цей метод отримують вміння приймати елементи які можна переміщувати, створені за допомогою метода Draggable.

$(1#droppable1).droppable({

drop: function() {

$(1#draggable1).text("Переміщено")

}

**Ефект розсіювання.**

Ефект Puff можна використовувати з методами show / hide/ toggle. Це створює ефект затягування, масштабуючи елемент і приховуючи його одночасно.

```
$(document).ready(function() {
        $("#hide").click(function(){
          $(".target").hide( "puff", { }, 2000 );
        });
        $("#show").click(function(){
          $(".target").show( "puff", {percent:100}, 2000 );
        });
```

***41. jQuery. Анімаційні ефекти та черги. Одночасне відтворення анімаційних ефектів. Почергове виконання функцій. Додавання функцій у чергу анімаційних ефектів. Зупинка ефектів і очищення черги. Заборона відтворення анімаційних ефектів. Визначення типу браузера. Ознаки, що визначають тип браузера. Застосування інших бібліотек разом з jQuery.***

**Анімаційні ефекти та черги.**

Коли використовуються ефекти, jQuery створює для кожного вібраного елемента чергу функцій, виконуючих анімацію. Кожна така функція починає виконуватись тільки після того як закінчиться виконання попередньої. Існує ряд методів які дозволяють отримувати інформацію про стан черги чи управляти нею.

query() - Повертаються чергові ефекти, які повинні бути виконані для елементів, що містяться в об'єкті jQuery

query(функція) - Включає функцію в кінець очереди

stop() - Зупиняє поточну анімацію

delay(тривалість) - Вставляє затримку між ефектами, які перебуваютьв черзі

**Одночасне відтворення анімаційних ефектів.**
За замовчуванням JQuery поміщає анімацію в чергу ефектів, щоб вони відбувалися один за іншим. Якщо ви хочете, щоб анімація сталася, відразу встановіть прапорець queue: false в свою карту опцій анімації.

```
$(function () {
   $("#first").animate({
      width: '200px'
   }, { duration: 200, queue: false });
   $("#second").animate({
      width: '600px'
   }, { duration: 200, queue: false });
});
```

**Почергове виконання функцій.**

Функція Toggle - За допомогою неї ми можемо змінювати необхідні нам дії при роботі з одним конкретним об'єктом. Припустимо, що у нас є якийсь текст і кнопка. При натисканні на неї один раз текст забарвлюється в червоний колір, натискаємо ще раз - повертається колишній колір.

```
$(".button").toggle(function() {
$(".text").css({"color": "red"});
        }, function() {
        $(".text").css({"color": "blue"});
        });
```

**Додавання функцій у чергу анімаційних ефектів.**

queue() - Повертає або змінює (в залежності від переданих параметрів) чергу функцій.

Справа в тому, що до елементів сторінки, jQuery прив'язує одну або кілька черг функцій. Кожна функції в такій черзі, автоматично викликається, коли завершиться попередня. Функції, які виконують анімаційні ефекти, поміщаються в таку чергу автоматично. Метод queue () дозволяє отримувати і змінювати черзі функцій.

```
$("div").queue(function () {      // додамо нову функ . в чергу

        // якась функція //

    $(this).dequeue();            //продовжити  чергу

  });
```

**Зупинка ефектів і очищення черги.**

.stop () - Зупиняє виконання поточної анімацію. Метод має один варіант використання:

.stop ([clearQueue], [jumpToEnd]):

Зупиняє виконання поточної анімації. Після зупинки анімації, наступна в черзі буде запущена автоматично, звичайно, якщо черга не порожня. Якщо необхідно зупинити всю чергу, то необхідно вказати true в параметрі clearQueue.

**Заборона відтворення анімаційних ефектів**

Скасування виконання всіх анімацій

Встановивши це властивість в true, ви вимкніть всі анімації, які можна виконувати за допомогою jQuery. Для того, що б анімації заробили знову, необхідно встановити це властивість назад в false.

```
jQuery.fx.off = true;
```

$ ('# Foo'). SlideUp (). FadeIn (); // виклик цих, і всіх наступних анімацій ні до чого не приведе

**Визначення типу браузера. Ознаки, що визначають тип браузера.**

**$ .Browser: objectv:** - Містить версію і тип використовуваного браузера. Інформація береться з змінної navigator.userAgent, яку надає сам браузер. Ця інформація може не відповідати дійсності, оскільки може бути змінена користувачем в настройках браузера. Тому, замість цього, ми рекомендуємо використовувати $ .support для уточнення конкретних особливостей браузера.

$ .Browser має наступні поля типів браузера:

- webkit (починаючи з jQuery 1.4)
- safari
- opera
- msie
- mozilla

Наприклад, для того, щоб дізнатися, чи маємо ми справу з Internet Explorer, потрібно використовувати наступний код:

```
if ($. browser.msie)

{

}
```

А ось так можна дізнатися версію браузера: **$. Browser.version.**

**Застосування інших бібліотек разом з jQuery.**

**AngularJS**

Це популярний фреймворк корпоративного рівня, який використовується багатьма розробниками для створення і обслуговування складних веб-додатків.

**React**

React - JS-бібліотека для створення користувацьких інтерфейсів. Це проект з відкритим вихідним кодом, здебільшого розроблений в Facebook, за участю ряду великих компаній.

**Backbone**

Цей фреймворк відомий своєю простотою і вміщується в один JS-файл. Backbone є повноцінним MVC-фреймворком з маршрутизацією. Моделі і колекції можуть взаємодіяти з RESTful API. В уявленнях використовується декларативна обробка подій, а маршрутизатор керує станом за допомогою URL. Достатньо лише створити односторінкове додаток без надлишкового функціоналу і складності.

**Babylon.js**

Хочете зробити кросбраузерності гру, цілком відповідну сучасним веб-стандартам? Тоді придивіться до Babylon.js, тривимірному движку на базі WebGL і JavaScript. Він дозволяє створювати неймовірні високоякісні ігри з реалістичною фізикою, звуком, системою частинок і іншими красотами.

***42. jQuery. Взаємодія із сервером за технологією Ajax. Знайомство з Ajax. Створення екземпляра XMLHttpRequest. Ініціалізація запиту.***

**Взаємодія із сервером за технологією Ajax. Знайомство з Ajax.**

Ajax - підхід до побудови призначених для користувача інтерфейсів веб-додатків, що полягає в «фоновому» обміні даними браузера з веб-сервером. В результаті, при оновленні даних веб-сторінка не перезавантажується повністю, і веб-додатки стають швидше і зручніше.

**AJAX модель**

1. Браузер створює виклик JavaScript, Який потім активує XMLHttpRequest.
2. У фоновому режимі веб-браузер створює HTTP-запит до сервера.
3. Сервер отримує, витягує і відправляє дані назад в веб-браузер.
4. Веб-браузер отримує запитані дані, який будуть безпосередньо відображатись на сторінці. Перезавантаження НЕ потрібно.

**Створення екземпляра XMLHttpRequest.**

План роботи з об'єктом XMLHttpRequest можна представити таким чином:

1. Створення екземпляра об'єкта XMLHttpRequest
2. Відкриття з'єднання
3. Установка обробника події (потрібно робити після відкриття і до відправки в IE)
4. Відправлення запиту.

**Приклад:**

```
var xhttp;

if (window.XMLHttpRequest){

  xhttp=new XMLHttpRequest();
```

**Методи XMLHttpRequest:**

- open (тіп_запроса, url, спосіб) Створює запит. тип_запроса встановлює тип запиту (GET або POST).url встановлює шлях до запитуваного файлу.спосіб встановлює спосіб виконання запиту. При значенні true запит буде виконаний асинхронно, при false синхронно.
- send ('дані') Дозволяє передати дані на сервер.
- setRequestHeader (заголовок, значення) Дозволяє додати HTTP заголовок до запиту.

**Ініціалізація запиту.**

Алгоритм запиту до сервера виглядає так:

1. Перевірка існування XMLHttpRequest.
2. Ініціалізація з'єднання з сервером.
3. Посилка запрсов сервера.
4. Обробка отриманих даних.

**Запит - jQuery.ajax (url, [settings])**

Здійснює запит до сервера без перезавантаження сторінки. Це низькорівневий метод, який володіє великою кількістю налаштувань. Він лежить в основі роботи всіх інших методів ajax. Має два варіанти використання:

url - адреса запиту.

settings - в цьому параметрі можна задати налаштування для даного запиту. Здається за допомогою об'єкта в форматі {ім'я: значення, ім'я: значення ...}. Жодна з налаштувань не є обов'язковою. Встановити налаштування за замовчуванням можна за допомогою методу $ .ajaxSetup ().

***43. jQuery. Спостереження за ходом виконання запиту. Одержання відповіді. Використання прямих методів Ajax. Виконання GET – запитів Ajax. Виконання POST - запитів Ajax.***

Зазвичай засоби Ajax асоціюються з відправкою даних форми, проте їх реальна сфера застосування набагато ширша. У бібліотеці jQuery визначено ряд так званих прямих методів, які в дійсності є оболонкою для викликів функцій ядра Ajax і дозволяють швидко і просто вирішувати типові завдання Ajax. Існує кілька простих методів цільового використання, доступних для вирішення звичайних завдань Ajax. По суті, ці прямі методи являють собою оболонки, які служать виключно для виклику функції $ .ajax () з наперед встановленими значеннями ряду параметрів. Використання цих методів тягне за собою незначну втрату продуктивності, оскільки при цьому ви фактично викликаєте метод, всередині якого відбувається установка значень параметрів і здійснюється виклик власне функції $ .ajax (). Однак зручності, що забезпечуються використанням прямих методів, насправді лише прискорюють розробку багатьох сценаріїв.

Перш за все, Ajax використовується для того, щоб виконати HTTP-запит GET з метою завантаження HTML-фрагмента, який можна додати в документ.

```
<script type="text/javascript">

$(document).ready(function() {

$.get("flowers.html", function(data) {

var elems = $(data).filter(ldlvl).addClass(l,dcell”);

elems.slice(0, 3).appendTo('#rowl');

elems.slice(3).appendTo("#row2");

});

 });

</script>
```

Тут використовується метод get (), якому передаються два аргументи. Перший з них-це URL-адрес, який вказує на документ, який ми хочемо завантажити. В даному випадку використовується адреса flowers.html, який буде інтерпретуватися як URL, заданий щодо URL-адреси, що використовується для завантаження основного документа. Другий аргумент - функція, яка буде викликатися в разі успішного виконання запиту. Як вже зазначалося вище, в Ajax інтенсивно використовуються функції зворотного виклику, оскільки запити виконуються в асинхронному режимі.

Незважаючи на те що зазвичай метод post () застосовується для відправки даних форми, дозволяється відправляти з його допомогою практично будь-які необхідні дані. Для цього потрібно лише створити об'єкт, що містить дані, викликати метод serialize () для їх відповідного форматування і передати їх методу post (). Ця методика може стати в нагоді, якщо ви отримуєте дані від користувача, минаючи форму, або хочете включити в POST-запит лише частина даних, введених в елементах форми. Використання методу post () таким способом

```
<script type="text/javascript">

$(document).ready(function() {

$('button').click(function(e) {

var requestData = {

apples: 2,

oranges: 10
```

```
};

$.post("http://node.jacquisflowershop.com/

order", requestData,

function(responseData) {

alert(JSON.stringify(responseData));

})

e.preventDefault();

})

});

</script>
```

У цьому сценарії явно створюється об'єкт і визначаються його властивості. Цей об'єкт передається методу post (), а для відображення отриманого від сервера відповіді використовується метод alert (). (Насправді сервера абсолютно байдуже, дані якого типу він отримує від браузера. Він просто спробує їх скласти і отримати підсумкову суму.)

***44. jQuery. Завантаження вмісту в елемент. Завантаження динамічних даних. Виконання запитів GET та POST за допомогою jQuery.***

Мабуть, найчастіше технологія Ajax використовується для отримання від сервера фрагментів вмісту і додавання цих фрагментів в деякі важливі для нас місця дерева DOM. Вміст може бути фрагментом HTML-розмітки або простим текстом, який потім стане вмістом необхідного елемента. Інструкція jQuery $('#someContainer').load('someResource'); легко справляється із завантаженням вмісту з сервера за допомогою одного з самих простих методів jQuery Ajax: load ().

Синтаксис методу load

load (url, parameters, callback)

Ініціює запит Ajax по заданому URL-адресою, можливо, з додатковими параметрами. Якщо вказана функція зворотного виклику callback, вона буде викликана після закінчення запиту. Вміст всіх елементів в обернутися наборі буде заміщено текстом відповіді.

Параметри

url (рядок) URL-адресу ресурсу на стороні сервера, якому відправляється запит.

parameters (рядок | об'єкт | масив) Визначає дані, які передаються у вигляді параметрів запиту. Цей аргумент може бути рядком, який буде використовуватися, як рядок запиту; об'єктом, властивості якого будуть перетворені в параметри запиту; або масивом об'єктів, імена і значення властивостей яких визначають пари ім'я / значення параметрів запиту. Якщо в цьому параметрі передається об'єкт або масив, запит виконується методом POST; якщо відсутня або в ньому передається рядок, - методом GET.

Callback (функція) Необов'язковий аргумент. функція зворотного виклику, яка викликається після того як дані, отримані у відповіді, будуть завантажені в елементи обернутого набору. Як параметр цієї функції передається текст відповіді, код статусу (зазвичай: "success") і об'єкт XHR.

Ця функція буде викликана для кожного елемента в обернутися наборі, при цьому сам елемент буде переданий їй через контекст функції (this).

Незважаючи на простоту використання цього методу, він володіє деякими важливими особливостями. Наприклад, якщо для визначення параметрів в аргументі parameters йому передається об'єкт або масив, запит виконується методом POST HTTP, в іншому випадку ініціюється запит GET. Якщо буде потрібно виконати запит GET з параметрами, ми повинні будемо включити їх в URL у вигляді рядка запиту. Але при цьому на наші плечі лягає вся відповідальність за правильність оформлення рядка запиту і кодування імен та значень параметрів. Для цього можна використовувати зручний метод JavaScript encodeURIComponent () або допоміжну функцію jQuery $ .param ().

Нерідко в бізнес-додатках, особливо на сайтах інтернет-магазинів, потрібно отримувати дані з сервера в реальному масштабі часу, щоб надати користувачам найсвіжішу інформацію. Зрештою, ніхто не хоче вводити покупців в оману, пропонуючи їм купити щось, чого насправді немає.

Одна з приємних особливостей застосування технології Ajax (поряд з невимушеністю, яку надає використання бібліотеки jQuery) складається в повній незалежності від серверних технологій. Очевидно, що від вибору серверної технології залежить структура адрес URL, але крім цього нам не потрібно турбуватися про те, що відбувається на стороні сервера. Ми просто відправляємо HTTP-запити, іноді з додатковими параметрами, і поки сервер відповідає на запити, повертаючи очікувану інформацію, нас найменше цікавить, яка технологія використовується на тій стороні, - Java, Ruby, PHP або старий добрий CGI .Найприкметніша операція, яка виконується в обробнику події готовності документа, - це застосування методу load () для швидкого завантаження з сервера фрагмента HTML-розмітки і розміщення його в дереві DOM як вкладеного вмісту

існуючого елемента. Цей метод надзвичайно зручний і прекрасно підходить для веб-додатків, що приводяться в рух серверними механізмами, такими як JSP і PHP.

Метод load () виконує запит GET або POST в залежності від того, в якому вигляді йому передаються (якщо взагалі передаються) дані з параметрами запиту, але іноді буває необхідний трохи більший контроль над тим, яким методом HTTP виконується запит. Навіщо це нам? Можливо, для того, щоб відповідати вимогам сервера. За традицією автори веб-сторінок безвідповідально ставилися до методів GET і POST, використовуючи то один, то інший і не замислюючись над тим, для яких цілей вони призначені. Призначення кожного з методів полягає в наступному

Запити GET дотримуються ідемпотентності (тотожність). Стан сервера і дані для програми не повинні змінюватися під впливом запитів GET. Один і той же запит GET, виконуючись знову і знову, повинен повертати в точності одні й ті ж результати (передбачається, що в цей час не відбувається нічого іншого, що призвело б до зміни стану сервера). Синтаксис функції GET **$.get(url,parameters,callback,type)**

Запити POST можуть бути неідемпотентними (нетотожні). Дані, що передаються сервера в таких запитах, можуть використовуватися для зміни стану програми, наприклад, для додавання записів в базу даних або видалення інформації з сервера. Синтаксис функції POST **$.post(url,parameters,callback,type)**

Таким чином, запити GET повинні використовуватися для отримання даних (що і випливає з назви методу). Для цього може знадобитися передавати деякі дані на сервер, наприклад, щоб ідентифікувати номер моделі взуття для отримання інформації про колір. Але коли дані передаються на сервер, щоб викликати зміни, слід використовувати метод POST.

***45. jQuery. Одержання даних методом GET та POST у форматі JSON. Використання методів для роботи з конкретними типами даних. Отримання даних в форматах XML, текстовому та JSON.***

Якщо застосування формату XML не є нагальною потребою або з якихось причин не підходить для передачі даних, замість нього часто використовується формат JSON. Одна з причин - з форматом JSON дуже просто працювати в клієнтських сценаріях, а jQuery ще більше спрощує справу. У випадках, коли заздалегідь відомо, що відповідь буде мати формат JSON, можна використовувати допоміжну функцію $ .getJSON (), яка автоматично проаналізує отриманий рядок JSON, а отримані дані передасть функції зворотного виклику. Ця допоміжна функція має наступний синтаксис:

**$.getJSON(url,parameters,callback)**

Ініціює запит GET до сервера, використовуючи заданий URL-адресу та будь-які параметри, передані у вигляді рядка запиту. Відповідь сервера інтерпретується як рядок в форматі JSON, а отримані з неї дані передаються функції зворотного виклику.

Параметри

| | |
|---|---|
| url | (рядок) URL-адресу ресурсу на стороні сервера, якому відправляється запит GET. |
| Parameters | (рядок \| об'єкт \| масив) Визначає дані, які передаються у вигляді параметрів запиту. Цей аргумент може бути рядком, яка буде використовуватися як рядок запиту; об'єктом, властивості якого будуть перетворені в параметри запиту; або масивом об'єктів, імена і значення властивостей яких визначають пари ім'я / значення параметрівзапиту. |
| callback | (функція) Функція, що викликається після успішного завершення запиту. Першим параметром цієї функції передаються дані, отримані в результаті інтерпретації відповіді в форматі JSON. Другим параметром передається код статусу. У третьому параметрі функції буде передана посилання на об'єкт XHR. |

Ця функція являє собою зручну заміну виклику функції $ .get () зі значенням json в аргументі type і відмінно підходить в ситуаціях, коли потрібно отримати дані від сервера без зайвих складнощів, властивих формату XML. Функції $ .get () і $ .getJSON () бібліотеки jQuery надають нам потужні інструменти для роботи із запитами GET.

В одному випадку ми хочемо виконати запит GET, в іншому хочемо (або повинні) виконати запит POST. Можна навести масу причин на користь методу POST перед методом GET. По-перше, відповідно до специфікації протоколу HTTP, метод POST слід використовувати для виконання всіх неідемпотентних запитів. Тобто якщо запит може викликати зміни стану на стороні сервера, слід вибрати метод POST (по крайней мере, тим, хто прагне дотримуватися положень специфікації). З іншого боку, відповідно до загальноприйнятої практики і угодами, допускається використовувати метод POST в разі, коли обсяг переданих даних занадто великий і його не можна передати сервера в рядку запиту URL, - це обмеження залежить від використовуваного броузера. Іноді серверний ресурс, з яким ми співпрацюємо, підтримує виключно запити POST або навіть може виконувати різні операції в залежності від того, який запит використаний, - POST або GET. Для випадків, коли бажано або необхідно виконувати запити методом POST, jQuery надає допоміжну функцію $ .post (), яка діє точно так же, як функція $ .get (), за винятком використовуваного методу протоколу HTTP. Її синтаксис:

**$.post(url,parameters,callback,type)**

Ініціює запит POST до сервера, використовуючи заданий URL-адресу та всі параметри, що передаються в тілі запиту.

Параметри

| | |
|---|---|
| url | (рядок) URL-адресу ресурсу на стороні сервера, якому відправляється запит POST. |

parameters (рядок | об'єкт | масив) Визначає дані, які передаються у вигляді параметрів запиту. Цей аргумент може бути рядком, яка буде використовуватися як рядок запиту; об'єктом, властивості якого будуть перетворені в параметри запиту; або масивом об'єктів, імена і значення властивостей яких визначають пари ім'я / значення параметрів запиту.

Callback (функція) Необов'язковий аргумент. Функція, що викликається після успішного ззавершення запиту. Першим параметром цієї функції передається текст відповіді, інтерпретація якого залежить від значення параметра type, другим - код статусу. У третьому параметрі функції буде передано об'єкт XHR.

Type (рядок) Необов'язковий параметр, що визначає, як повинен інтерпретуватися текст відповіді. Може мати одне з наступних значень: html, text, xml, json, script або jsonp.

У всьому іншому, крім типу запиту POST, функція $ .post () використовується точно так же, як функція $ .get (). Бібліотека jQuery сама подбає про передачу параметрів в тіло запиту (на противагу формуванню рядки запиту) і відповідно встановить метод HTTP.

БібліотекаjQuery надає три допоміжних методи, які роблять роботу з деякими типами даних більш зручною. Метод load () призначений для отримання тільки HTML-даних, що дозволяє поєднати запит HTML-фрагмента, обробку відповіді від сервера для створення набору елементів і вставку цих елементів в документ на одну дію. Приклад використання методу load ()

```
<script type="text/javascript">
$(document).ready(function() {
        $('<button>Ajax</button>').appendTo('#buttonDiv')
        .click(function(e) {
                $(1#rowl').load("flowers.html");
                e.preventDefault();
        });
 });
</script>
```

У цьому сценарії ми викликаємо метод load () для елемента, в який хочемо вставити нові елементи, і передаємо йому URL-адресу в якості аргументу. Якщо запит завершується успішно, а отриманий від сервера відповідь містить действи- тельнир HTML-фрагмент, елементи вставляються в вказане місце в документі.

Метод getScript () завантажує файл JavaScript, а потім виконує інструкції які в ньому містяться
$.getScript("myscript.j s");

Для завантаження даних JSON з сервера призначений метод getJSON (). Можливо, це найменш корисний з усіх трьох допоміжних методів, оскільки він не робить з даними нічого понад те, що робить базовий метод get (). **$.getJSON(”mydata.json"/ function(data))**

Щоб отримати дані у форматі XML
```
$.ajax({
        url: "name.xml",
        cache: false,
        dataType: "xml",
        success: function(xml){
        } });
```

***46. jQuery. Повне керування запитами Ajax. Виконання запитів Ajax з усіма налаштуваннями. Робота з подіями Ajax. Обробка подій Ajax. Обробка успішних запитів. Обробка помилок.***

Для випадків, коли потрібно мати повний контроль над виконанням запитів Ajax, jQuery надає універсальну допоміжну функцію $ .ajax () для виконання Ajax-запитів. Інші методи і функції бібліотеки jQuery, засновані на технології Ajax, в кінцевому рахунку для ініціації запиту використовують саме цю функцію. Її синтаксис:

$ .ajax (options). Ініціює Ajax-запит, використовуючи передані їй параметри для управління передачею запиту і звернення до функції зворотного виклику. Параметри options (об'єкт) Об'єкт, властивості якого визначають параметри виконання операції. Повертає екземпляр XHR. Аргумент options може визначати широкий діапазон значень для налаштування дій, виконуваних функцією. Ось деякі з них

| | |
|---|---|
| url | (рядок) URL-адрес запиту. |
| Type | (рядок) Вживаний метод HTTP, зазвичай POST або GET. якщо відсутній, за замовчуванням використовується метод GET. |
| data | (рядок \| об'єкт \| масив) Визначає значення, які будуть використовуватися як параметри запиту та передаватися разом із запитом. Якщо запит виконується методом GET, ці дані передаються у вигляді рядка запиту. Якщо запит виконується методом POST, дані передаються в тілі запиту. В будь-якому випадку кодування значень виконується допоміжною функцією $ .ajax. У цьому параметрі допускається передавати рядок, який буде використовуватися як рядок запиту або як тіло відповіді; об'єкт, властивості якого будуть перетворені в параметри запиту; або масив об'єктів, імена і значення властивостей яких визначають пари ім'я / значення параметрів запиту. |
| cache | (логічне значення) Якщо має значення false, відповідь не буде кешуватися оглядачем. За замовчуванням приймає значення true коли параметр dataType має значення script або jsonp. |
| context | (елемент) Визначає елемент, який буде використовуватися в якості контексту для всіх функцій зворотного виклику, асоційованих із запитом. |

Тут вказано декілька найпопулярніших параметрів, насправді їх набагато більше.

Бібліотека jQuery надає спосіб визначити значення властивостей Ajax за замовчуванням, які будуть використовуватися якщо їх не перевизначити явно. Це допоможе спростити сторінки, які виконують безліч однотипних запитів Ajax. Функція для установки значень за замовчуванням називається $ .ajaxSetup () і має наступний синтаксис:

$ .ajaxSetup (options)

Встановлює переданий набір параметрів як значення за замовчуванням для всіх наступних викликів функції $ .ajax ().

Параметри:

options (об'єкт) Об'єкт, властивості якого визначають значення властивостей Ajax за замовчуванням. Це ті ж самі властивості, які використовуються функцією $ .ajax (). Ця функція не повинна використовуватися для визначення оброблювачів success, error і completion.

Це може бути корисним в будь-якому місці сценарію, зазвичай при завантаженні сторінки (але за бажанням автора це може бути будь-яке інше місце), для установки значень за замовчуванням, які застосовуються для всіх наступних викликів функції $ .ajax ().

Кілька параметрів дозволяють вказувати функції для обробки подій, які можуть запускатися на протязі життєвого циклу Ajax-запиту. Саме таким способом можна вказувати функції зворотного виклику, які відіграють важливу роль в Ajax-запитах. Список параметрів пов'язаних з подіями, разом з їх короткими описами:

| | |
|---|---|
| beforeSend | Задає функцію, яка буде викликатися перед запуском Ajax-запиту |
| complete | Задає функцію, яка буде викликатися при успішному або невдалому завершення Ajax-запиту |
| error | Задає функцію, яка буде викликатися при невдалому завершенні запиту |
| success | Задає функцію, яка буде викликатися при успішному виконанні запиту |

У прикладі, що пояснює використання параметра success, при виконанні функції були опущені два аргументи - повідомлення, яке описує результат запиту, і об'єкт jqXHR. Приклад використання функції, яка приймає ці аргументи,

```
<script type="text/javascript">
	$(document).ready(function() {
		$.ajax({
			url: "mydata.json",
			success: function(data, status, jqxhr) {
				console.log("Статус: " + status);
				console.log("Статус jqXHR: " + jqxhr.status +
				" " + jqxhr.statusText); console.log(jqxhr.getAllResponseHeaders());
				var template = $(1#flowerTmpl');
				template.tmpl(data.slice(0, 3)).appendTo("#rowl");
				template.tmpl(data.slice(3)).appendTo("#row2");
			}
		}>; }>;
</script>
```

Аргумент status - це рядок, що описує результат запиту. Функція зворотного виклику, яку ми ставимо, використовуючи параметр success, виконується лише для успішних запитів, і тому значенням даного аргументу зазвичай є success. Винятком є випадок, коли ви використовуєте параметр ifModified

Параметр error використовується для вказівки функції, яка повинна викликатися при невдалому завершенні запиту. Відповідний приклад

```
<style type="text/css">
	.error {color: red; border: mediiun solid red; padding: 4px; margin: auto; width: 200px; text-align:
	center}
</style>
<script type="text/javascript">
	$(document).ready(function() {
		$.ajax("NoSuchFile.json",{
			success: function(data, status, jqxhr) {
				var template = $('#flowerTmpl');
```

```
                        template.tmpl(data.slice(0, 3)).appendTo("#rowl");
                        template.tmpl(data.slice(3)).appendTo("#row2");
                }
                error: function(jqxhr, status, errorMsg) {
                        $('<div class=error/>')
                                .text("CTaTyc: " + status + " Ошибка:
                                        " + errorMsg)
                                .insertAfter('hl1);
                }
        }>;
        });
</script>
```

Тут запитується відсутній на сервері файл NoSuchFile. json, і тому запит свідомо не зможе бути виконаний, в результаті чого буде викликана функція, задана за допомогою параметра error. Аргументами цієї функції є об'єкт jqXHR, а також повідомлення про стан помилки і повідомлення про помилку, отримане у відповіді сервера.

Аргумент status може мати одне із значень

| | |
|---|---|
| abort | Запит був скасований ще до його відправлення (використовується об'єкт j qXHR) |
| error | Загальна помилка, про яку зазвичай повідомляє сервер |
| parsererror | Неможливість синтаксичного розбору даних, повернутих сервером timeout Закінчення допустимого часу очікування відповіді сервера |

***47. jQuery. Завдання декількох обробників подій. Налаштування контексту для подій. Використання глобальних подій для Ajax. Управління глобальними подіями.***

Подія - це "реакція" браузера на дію користувача. Наприклад, коли користувач на сторінці натискає на кнопку, браузер відповідно до нього генерує подію. Кілька оброблювачів можуть стати в нагоді в тих випадках, коли одна і та ж функція повинна використовуватися для обробки подій, джерелом яких служать різні набори елементів. Значення властивості data можна використовувати для того, щоб визначити, який тип відповідної реакції потрібно в тому чи іншому випадку.

Налаштування контексту.

jQuery.proxy (function, context);

jQuery.proxy (context, name);

jQuery - дуже популярна бібліотека javascript, тому спочатку ми розглянемо застосування jQuery.proxy для прив'язки контексту до функції. jQuery.proxy повертає нову функцію, яка при виклику викликає оригінальну функцію function в контексті context. З використанням jQuery.proxy вищеописану завдання можна вирішити так:

setTimeout ($. proxy (object.f, object) 1000);

Для обробки даних глобальних подій в jQuery надає наступні методи:

**ajaxComplete**: реєструє обробник завершення запиту AJAX, **ajaxError**: реєструє обробник, який буде викликатися, якщо запит завершився з помилкою, **ajaxSend**: реєструє функцію, яка буде спрацьовувати перед відправкою запиту, **ajaxStart**: реєструє обробник, який запускається при запиті ajax, **ajaxStop**: реєструє обробник, який запускається при завершенні запитів ajax, **ajaxSuccess**: реєструє функцію, яка запускається при успішному завершенні запиту. Всі методи приймають в якості параметра функцію обробника. У свою чергу дана функція у всіх методах, крім ajaxStart і ajaxStop може приймати три параметра:

об'єкт **Event**, що містить дані про подію, об'єкт **jqXHR**, пов'язаний з даними ajax-запитом та додаткові параметри.

Для управління глобальними подіями призначена опція global в методі ajax. За умовчанням вона має значення true, тобто глобальні події будуть генеруватися, і ми їх зможемо обробити. Якщо ж цей параметр дорівнює false, то події не будуть генеруватися.

***48. Введення в jQuery UI. Оформлення і ефекти. Використання бібліотеки jQuery UI.***

Можливість застосування різних ефектів до елементів була і в jQuery. За допомогою ефектів jQuery Ви, наприклад, могли змусити абзац поступово пропадати або з'являтися, застосовувати до елементів анімацію і багато іншого.

Ефекти jQuery UI значно розширюють ці можливості. Тепер можна змусити елементи пропадати або з'являтися п'ятнадцятьма різними способами.В анімації Ви можете використовувати властивості зміни кольору фону, тексту і кордонів елементів.jQuery UI також розширює базову функціональність jQuery для маніпулювання класами оформлення.

**Blind** Ефект "Осліплення", **Bounce** Ефект "Відскок", **Clip** Ефект "Відсікання", **Drop** Ефект "Падіння", **Explode** Ефект "Вибух", **Fade** Ефект "Вицвітання", **Fold** Ефект "Складка", **Highlight** Ефект "Освітлення", **Puff** Ефект "Розсіювання", **Pulsate** Ефект "пульсування", **Scale** Ефект "Масштабування". За допомогою опції percent: відсотки Ви можете задати на скільки відсотків від поточного розміру необхідно зменшити або збільшити елемент, **Shake** Ефект "Тряска", **Slide** Ефект "ковзання", **Size** Ефект "Калібрування". За допомогою опції to: і включених в неї опцій width: ширина_в_пікселях і height: висота_в_пікселях Ви можете задати розміри, до яких необхідно "відкалібрувати" поточний елемент, **Transfer** Ефект "Перехід". За допомогою опції to: id_ілі_class_елемента Ви можете вказати елемент, в який перейде поточний елемент. За допомогою опції className: ім'я_класу Ви можете оформити ефект переходу за допомогою CSS.

Всі плагіни jQuery UI спроектовані так, щоб UI-віджети можна було підлаштовувати під особливості оформлення та роботи більшості сайтів. Оформлення кожного віджета описано за допомогою CSS і містить два рівня правил оформлення: базові стилі CSS-фреймворка і специфічні стилі, що описують правила обраної теми оформлення. Вносити зміни в оформлення jQuery UI плагінів можна такими способами:

Вносити зміни в css вручну. Щоб отримати більший контроль над тим, як оформлені елементи, які беруть участь в роботі плагіна, можна на початку конфігурувати тему за допомогою ThemeRoller, а потім внести необхідні зміни в отриманий ui.theme.css файл або в один з css-файлів конкретного плагіна (останнє, як вже було описано, не рекомендується). Так наприклад, ви можете задати радіус кутів у кнопок, відмінний від радіуса у інших компонентів або змінити шлях до спрайту картинок плагіна.

Зробити css-файл "з нуля". Якщо необхідно кардинально змінити зовнішній вигляд елементів плагіна, то цілком можливо, кращим варіантом буде створити його самостійно, "з нуля". Однак, це вимагає доброго розуміння html і css структури UI-плагінів, а так же хороших знань в області css.

Для того, щоб скористатися можливостями плагінів jQuery UI їх необхідно спочатку підключити до сторінки, на якій вони будуть використовуватися. Існують два варіанти підключення jQuery UI:

Локальне підключення. Даний спосіб вимагає скачування спеціального файлу з офіційного сайту;

Віддалене підключення. Даний спосіб не вимагає скачування файлу, а замість цього використовує його віддалено (дана послуга надається компанією Google).

Приклад використання:

```
<script src="//ajax.googleapis.com/ajax/libs/jquery/1.10.2/jquery.min.js">
</script>
<script src="//ajax.aspnetcdn.com/ajax/jquery.ui/1.10.3/jquery-ui.min.js"></script>
```

***49. Короткий огляд бібліотеки jQuery. Ефекти jQuery UI. jQuery UI з мишею.***

JQuery - це кроссплатформенная JavaScript-бібліотека, призначена для спрощення клієнтського сценарію HTML. JQuery - найпопулярніша в даний час бібліотека JavaScript.Вона робить такі речі, як обхід і маніпулювання документами HTML, обробка подій, анімація і Ajax набагато простіше з простим у використанні API, який працює у великій кількості браузерів.
Можливість застосування різних ефектів до елементів була і в jQuery. За допомогою ефектів jQuery Ви, наприклад, могли змусити абзац поступово пропадати або з'являтися, застосовувати до елементів анімацію і багато іншого.

Ефекти jQuery UI значно розширюють ці можливості. Тепер можна змусити елементи пропадати або з'являтися п'ятнадцятьма різними способами.В анімації Ви можете використовувати властивості зміни кольору фону, тексту і кордонів елементів.jQuery UI також розширює базову функціональність jQuery для маніпулювання класами оформлення.

**Blind** Ефект "Осліплення", **Bounce** Ефект "Відскок", **Clip** Ефект "Відсікання", **Drop** Ефект "Падіння", **Explode** Ефект "Вибух", **Fade** Ефект "Вицвітання", **Fold** Ефект "Складка", **Highlight** Ефект "Освітлення", **Puff** Ефект "Розсіювання", **Pulsate** Ефект "пульсування", **Scale** Ефект "Масштабування". За допомогою опції percent: відсотки Ви можете задати на скільки відсотків від поточного розміру необхідно зменшити або збільшити елемент, **Shake** Ефект "Тряска", **Slide** Ефект "ковзання", **Size** Ефект "Калібрування". За допомогою опції to: і включених в неї опцій width: ширіна_в_пікселях і height: висота_в_пікселях Ви можете задати розміри, до яких необхідно "відкалібрувати" поточний елемент, **Transfer** Ефект "Перехід". За допомогою опції to: id_ілі_class_елемента Ви можете вказати елемент, в який перейде поточний елемент. За допомогою опції className: ім'я_класу Ви можете оформити ефект переходу за допомогою CSS.

Елемент, до якого застосовано взаємодія Draggable можна переміщати за допомогою покажчика миші в межах вікна браузера. Властивості:

- **axis** Обмежує можливості переміщення певними напрямками. Значення за замовчуванням - false, воно означає відсутність обмежень, але можна також вказати значення "x" (переміщення тільки уздовж осі X) або "y" (переміщення тільки уздовж осі Y)
- **containment** Обмежує розташування переміщуваного елемента певної областю екрану. Типи підтримуваних значень описані в таблиці нижче, при розгляді відповідного прикладу. Значення за замовчуванням - false, воно означає відсутність обмежень
- **delay** Визначає час, протягом якого має здійснюватися перетягування елемента, перш ніж він переміститься. Значення за замовчуванням - 0, воно означає відсутність затримки
- **distance** Визначає відстань, на яку користувач повинен перетягнути елемент з його початкової позиції, перш ніж він дійсно переміститься. Значення за замовчуванням - 1 піксель
- **grid** Здійснює примусову прив'язку переміщуваного елемента до осередків сітки. Значення за замовчуванням - false, воно означає відсутність прив'язки

***50. Віджети jQuery UI. Відпускання перетягування елементів. Події відпускання елементів. Упорядкування об'єктів. Підключення впорядкованих списків. Зміна розмірів елементів. Додавання здатності до зміни розмірів. Виділення елементів. Додавання здатності до виділення.***

jQuery UI надає набір готових віджетів призначених для створення призначеного для користувача інтерфейсу веб-додатків. Поведінка віджетів може налаштовуватися за допомогою переданих їм опцій. Зовнішній вигляд віджетів може бути налаштований. Загальний вигляд створення віджетів jQuery UI виглядає приблизно наступним чином:

Угруповання елементів на сторінці особливим чином (індивідуально для кожного віджета); Застосування до згрупованим елементів спеціального методу, який перетворює їх в віджет. Елемент, до якого застосовано взаємодія Draggable можна переміщати за допомогою покажчика миші в межах вікна браузера. Взаємодія Draggable підтримує простий набір подій, що повідомляють про перетягуванні елементу. Ці події описані нижче:

- **create** Відбувається в момент застосування взаємодії Draggable до елементу
- **start** Відбувається в момент початку перетягування
- **drag** Відбувається при кожному переміщенні миші в процесі перетягування
- **stop** Відбувається в момент відпускання кнопки миші в процесі перетягування

Взаємодія Sortable дозволяє змінювати порядок розташування елементів у наборі шляхом їх перетягування з одного місця в інше. Щоб застосувати взаємодію Sortable, слід вибрати елемент, що містить окремі об'єкти, які ви хочете відсортувати, і викликати метод sortable ().

**.resize().** Даний метод є спрощеним скороченням операції .on ('resize', handler) або .trigger ('resize').Події resize відбувається, коли змінюються розміри об'єкта window (вікна браузера). При включеній опції fiiispace дозволяє зміна розміру віджета відповідно до змін розміру батьківського елемента.

Параметри highlight і unhighlight дозволяють вказати функції, які повинні використовуватися для візуального виділення елементів, що містять неправильні значення.

*51. Віджети jQuery UI: Кнопки і групи кнопок. Індикатори ходу виконання операції. Повзунки*

**Віджет Button** відносно простий, але це не заважає йому надавати трансформує вплив на HTML-документи. Даний віджет забезпечує застосування теми jQuery UI до елементів button і a. Це означає, що розмір, форма, характеристики шрифту і колір елемента перетворюються таким чином, щоб їх зовнішній вигляд відповідав обраній вами темі оформлення jQuery UI. Застосування віджета Button зводиться до використання jQuery для вибору елементів, які ви хочете перетворити, і викликом методу button (). Все інше jQuery UI бере на себе. Метод buttonset () можна використовувати по відношенню до будь-яких інших елементів, до яких застосовується метод button (). Кінцевим результатом цього є застосування стилю групи перемикачів, але не поведінки, тому кожна кнопка буде працювати незалежно від інших.

**Progress Bar** - індикатора процесу. За допомогою індикатора процесу можна інформувати користувача про досягнутий прогрес в ході виконання будь-якої задачі. Цей індикатор використовується лише в разі детермінованих задач, тобто задач, масштаб яких відомий, і можна точно визначити процентну частку виконаного обсягу роботи. Іншу категорію складають недетерміновані завдання. Хід виконання таких завдань насилу піддається кількісній оцінці, так що в цьому випадку необхідно лише повідомити користувачеві, що до завершення завдання доведеться трохи почекати. Щоб створити індикатор процесу, слід вибрати елемент div і викликати метод progressbar (). Для створення індикатора процесу слід використовувати порожній елемент div. Наявність будь-якого вмісту впливає на розташування віджета на сторінці.

**Віджет Slider** дозволяє створювати повзунки з елементів HTML-документа. Для створення повзунків використовується метод slider (). Повзунки застосовуються в тих випадках, коли користувачеві необхідно надати можливість вибирати значення, що лежать в деякому заданому діапазоні. Візуальний стиль повзунок автоматично підлаштовується під тему оформлення, загальну для всіх віджетів. Користувач може переміщати повзунок вздовж шкали за допомогою миші або клавіатури. За замовчуванням повзунки розташовуються в горизонтальному напрямку, але, використовуючи властивість orientation, можна створювати і вертикальні повзунки.

***52. Віджети з функцією автодоповнення. Віджети вибору дати. Віджети з вкладками. Віджети jQuery UI. Багатосторінкові віджети аккордіон. Діалоги***

**Віджет Autocomplete** полегшує ручне введення інформації, пропонуючи на вибір користувачеві варіанти автоматичного заповнення текстових полів під час введення символів. Продумане використання цього віджету забезпечує значну економію часу за рахунок прискорення введення даних і зниження ймовірності помилок введення. Щоб створити елемент управління автозаповненням, слід викликати метод autocomplete () для відповідного елемента input. Йому необхідно передавати об'єкт відображення, що містить значення опції source. Ця опція дозволяє вказати джерело, з якого повинні вилучатись дані для автозаповнення. Роль такого джерела може грати простий масив значень. Віджет Autocomplete не перевіряє правильність введених даних, і користувач може вводити в текстовому полі будь-які значення, а не тільки ті, які визначені опцією source.

**Datepicker** - зручний інтерактивний календар, що полегшує введення дат. Віджет Datepicker спрощує вибір дати і її уявлення в уніфікованому вигляді, тим самим знижуючи ймовірність появи помилок. Існують два основні способи створення віджета Datepicker. Найчастіше його приєднують до елементу input за допомогою методу datepicker (). Негайної зміни зовнішнього вигляду елемента при цьому не відбувається, але як тільки елемент отримає фокус введення, поруч з ним відобразиться календар, за допомогою якого можна буде вибрати необхідну дату. Календарі описаного типу називаються спливаючими календарями. Другий спосіб використання віджету Datepicker передбачає його вбудовування в документ. Для цього слід вибрати елемент div або span з допомогою jQuery і викликати метод datepicker (). Вбудований календар відображається весь час, поки видно HTML-елемент, на основі якого він створений. За допомогою опції defaultDate можна вказати дату, яка буде автоматично підсвічуватися при відкритті календаря. Якщо значення опції defaultDate не встановлено, замість нього використовується поточна дата, яка встановлена у системі.

Для створення **віджета Tabs** використовується метод tabs (). Для застосування цього методу потрібно окрема структура HTML- елементів. Елемент, до якого ви збираєтеся застосувати метод tabs (), повинен містити елементи двох типів. Елементи першого типу-це елементи вмісту, тобто елементи, вміст яких має відображатися в окремих вкладках. До другого типу відносяться структурні елементи містять інформацію, яка використовується для створення структури вкладок. Як контейнери вмісту використовуються елементи div. Дуже важливо, щоб кожен елемент вмісту був забезпечений ідентифікатором (атрибутом id) і віджет міг знаходити елемент, який потрібно відобразити. Для створення необхідної структури використовуються елементи li, кожен з кото рихдолжен містити елемент a. Кількість елементів li визначає кількість вкладок.

jQuery UI надає набір готових віджетів призначених для створення призначеного для користувача інтерфейсу веб-додатків. Переглянуті в цьому питанні віджети є частиною jQuery UI.

**Віджет Accordion** дозволяє представити набір панелей, що мають інформаційний вміст, таким чином, щоб в кожен момент часу відображалася лише одна з них. При виборі користувачем іншої панелі відображувана до цього панель закривається, а нова відкривається. Панелі віджета Accordion відмінно підходять для відображення вмісту в ситуаціях, коли він представляється у вигляді декількох незалежних частин, і ви не хочете перевантажувати користувача, відображаючи їх одночасно. Для створення віджета Accordion використовується метод accordion ().

**Віджет Dialog** створює плаваюче вікно з заголовком і областю вмісту, що зовні нагадує діалогові вікна звичайних додатків. Віджети Dialog можна використовувати як для виведення повідомлень, так і для залучення уваги користувачів до важливих подій. Однак, як і в разі будь-яких інших елементів, що затуляють собою частину вмісту документа, в використанні діалогових вікон слід дотримуватися помірності і вдаватися до їхньої допомоги лише в ситуаціях, коли вивід інформації в макеті самого документа викликає труднощі. Для створення віджета Dialog, що дозволяє перетворювати блокові елементи (звичайно-div) в діалогові вікна, слід вибрати елемент за допомогою jQuery і викликати для нього метод dialog (). Віджет Dialog відноситься до числа тих віджетів, для нормальної роботи яких потрібна певна структура HTML-елементів. Для віджету Dialog потрібно елемент div з атрибутом title. Значення цього атрибута є заголовком діалогового вікна. Вміст елемента div використовується в якості інформації, яка відображається в діалоговому вікні.