лек

Лекція. Літерали об'єктів, JSON, функції зворотного виклику

Зав.каф.КН, PhD Граф М.С.

```
var shinyAndNew = new Object();
```

```
var ride = new Object();
ride.make = ‘Yamaha’;
ride.model = ‘V-Star Silverado’
ride.year = 2005;
ride.purchased = new Date(2022,3,15);
```

Припустимо, нам потрібно змінити значення дати покупки:

```
ride.purchased = new Date(2022,2,1);
```

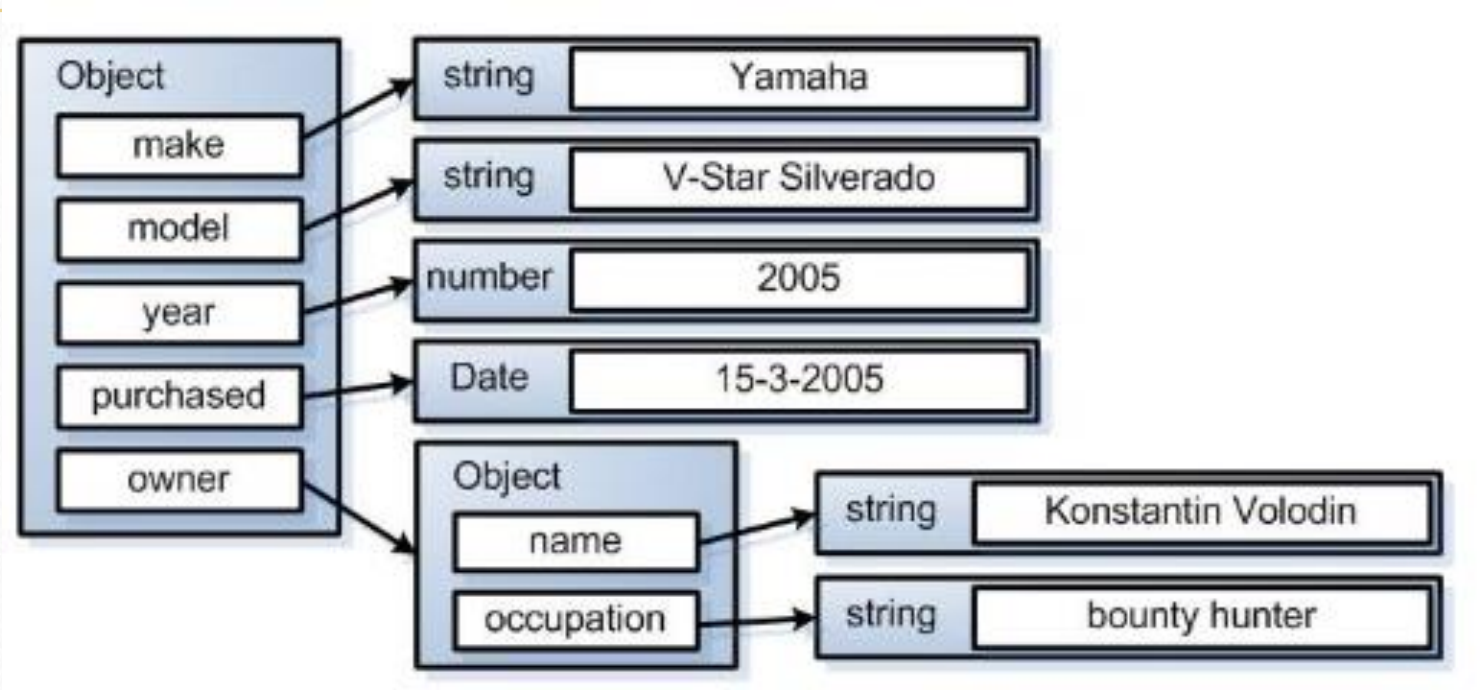
Припустимо, ми додали до нашого об'єкту ride нову властивість, що описує власника транспортного засобу. Ця властивість – ще один об'єкт JS, який у свою чергу містить такі властивості, як ім'я та професія власника:

```
var owner = new Object();
owner.name = ‘Konstantin Volodin’;
owner.occupation = ‘bounty hunter’;
ride.owner = owner;
```

Звернення до вкладених властивостей можна описати так:

```
var ownerName = ride.owner.name;
```

Тут немає жодних обмежень на глибину вкладеності. Тепер ієрархія об'єктів виглядає так:



Що робити, якщо властивість з ім'ям color.scheme?

Для таких випадків використовується більш загальний формат звернення до властивостей:

```
object[propertyNameExpression]
```

propertyNameExpression – вираз JS, що визначається як рядок, що формує ім'я властивості, до якого відбувається звернення. Наприклад, всі три наступні посилання еквівалентні:

```
ride.make
ride['make']
ride['m'+'a'+'k'+'e']
```

Так само як:

```
var p = 'make';
ride[p];
```

Застосування загального оператора посилання – єдиний спосіб звернутися до властивостей, імена яких не є допустимими ідентифікаторами JS, наприклад:

```
ride["a property name that's rather odd"]
```

Літерали об'єктів

Літерали називають уявлення значення деякого типу даних.

```
int a2 = 2;
int d = a2;
string example = "Приклад";
```

```
var ride = {
    make: ‘Yamaha’,
model: ‘V-Star Silverado’,
year: 2005,
    purchased: new Date(2005,3,15),
    owner: {
    name: ‘Konstantin Volodin’,
    occupation: ‘bounty hunter’,
}
};
```

Більшість авторів сторінок віддають перевагу такій формі запису, що називається **JSON (JavaScript Object Notation).** Структура цієї форми запису дуже проста – об'єкт позначається парою фігурних дужок, всередині яких через кому перераховані властивості. Кожна властивість позначається шляхом запису його імені та значення, розділених символом двокрапки.

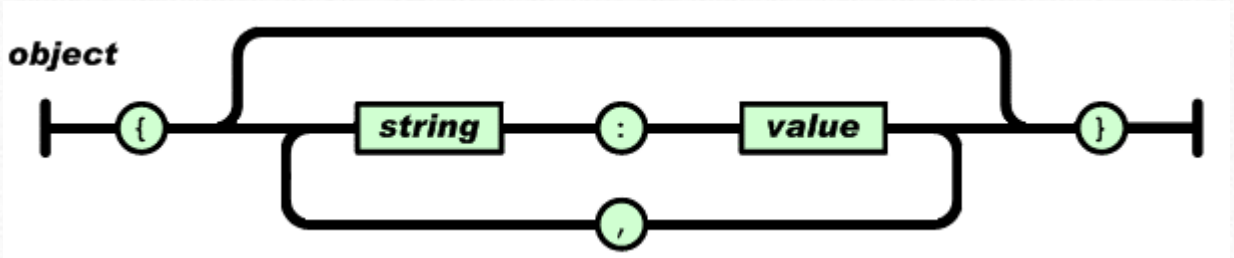
Так само у форматі JSON можна описувати масиви, помістивши список елементів, розділених комами, у квадратні дужки:

```
var someValues = [2,4,5,8,9,12,14,16,17,23,36];
```

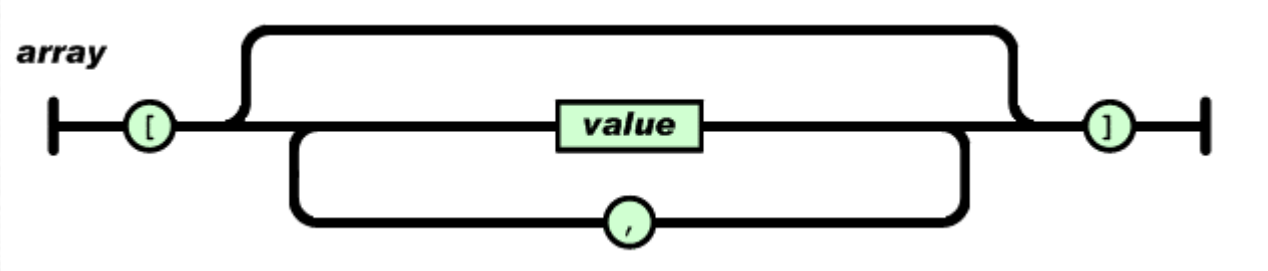
JSON (JavaScript Object Notation) - простий формат обміну даними, зручний для читання та написання як людиною, так і комп'ютером. JSON – текстовий формат, повністю незалежний від мови реалізації, але він використовує угоди, знайомі програмістам С-подібних мов, таких як C, C++, C#, Java, JavaScript, Perl, Python та багато інших. Ці властивості роблять JSON ідеальною мовою обміну даними.

- Колекція пар ключ/значення. У різних мовах ця концепція реалізована як *об'єкт* , запис, структура, словник, хеш, іменований список чи асоціативний масив.
- Упорядкований перелік значень. У більшості мов це реалізовано як *масив* , вектор, список чи послідовність.

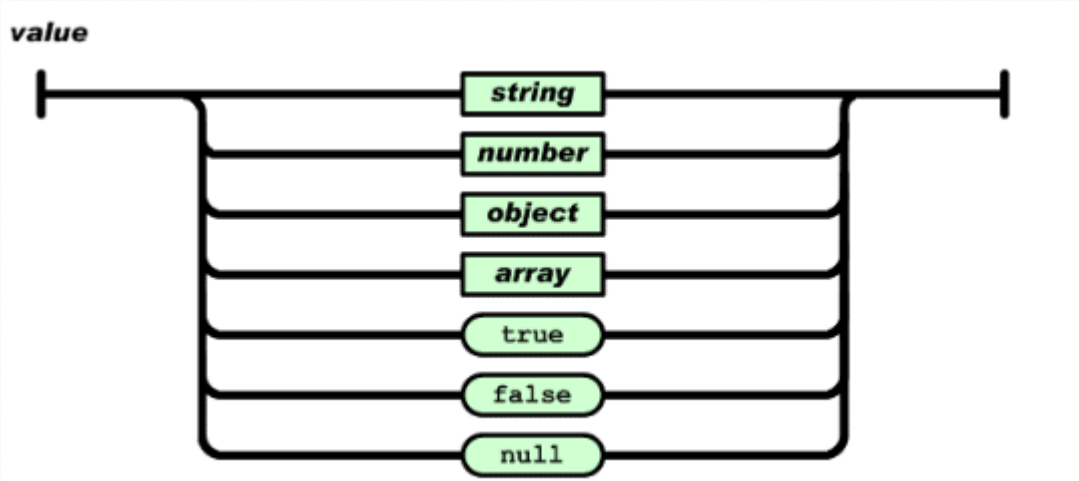
У нотації JSON це виглядає так:
Об'єкт - невпорядкований набір пар ключ/значення. Об'єкт починається з { (відкриває фігурної дужки) і закінчується } (закриває фігурною дужкою). Кожне ім'я супроводжується: (двокрапкою), пари ключ/значення поділяються, (комою).



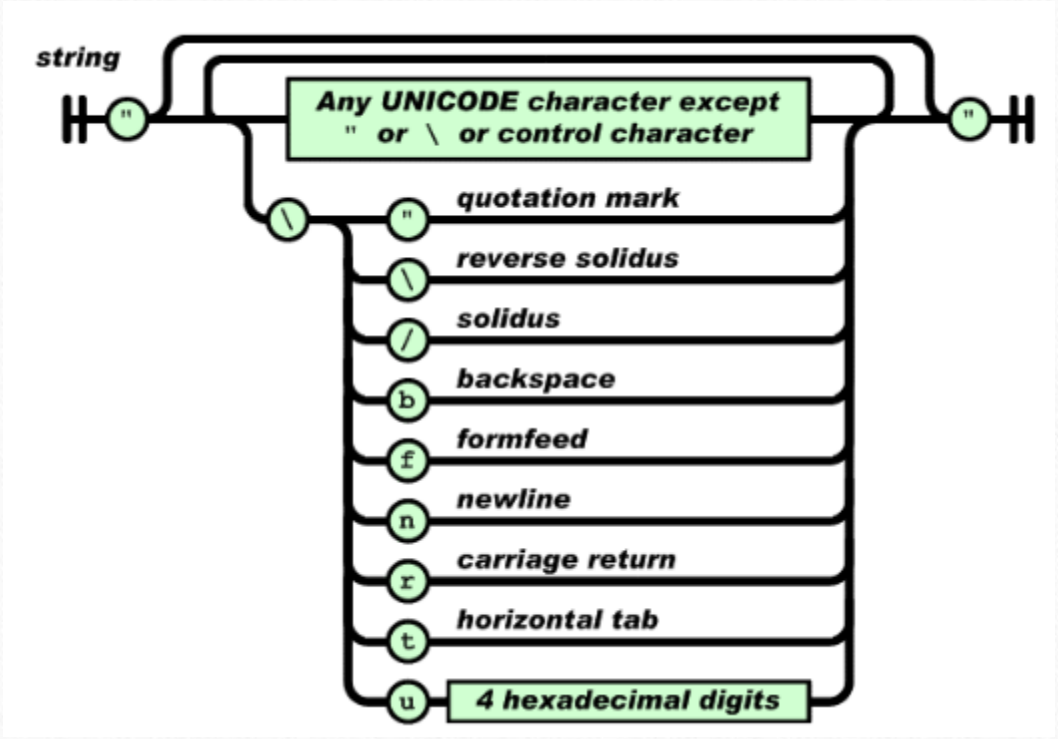
Масив – впорядкована колекція значень. Масив починається з [(відкриває квадратної дужки) і закінчується] (закриває квадратною дужкою). Значення розділені , (комою).



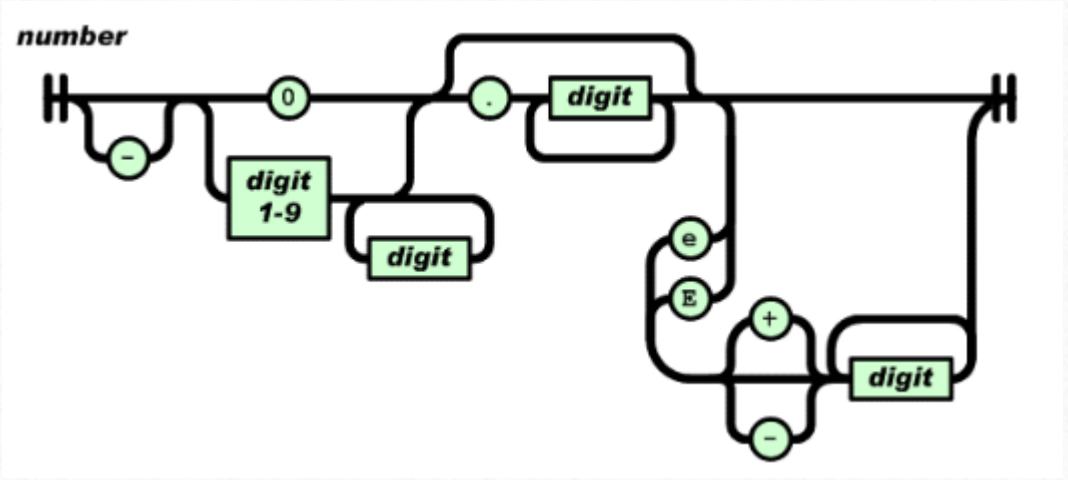
Значення може бути *рядком* у подвійних лапках, *числом* , true , false , null , *об'єктом* або *масивом* . Ці структури може бути вкладеними.



Рядок - колекція нуля або більше символів Unicode, укладена в подвійні лапки, використовуючи \ (зворотну косу рису) як символ екранування. Символ представляється як односимвольний рядок. Схожий синтаксис використовується в C та Java.



Число представляється так само, як у C або Java, крім того, що використовується тільки десяткова система числення.



Об'єкти як властивості об'єкта window

```
var aVariable = ‘No one can draw a clear line between sane and insane.’;
someObject.aProperty = ‘You move that line as you see fit for yourself.’;
```

Еквівалентні всі наступні інструкції:

```
var foo = bar;
window.foo = bar;
foo = bar;
```

Загалом огляд об'єкта Object у мові JavaScript завершено. Було розглянуто такі важливі поняття:

- Об'єкт у JS – це невпорядкований набір властивостей.
- Властивість складається з імені та значення.
- Можна оголошувати об'єкти у вигляді літералів об'єктів.
- Глобальні змінні є властивостями об'єкта window.

Функції як звичайні об'єкти

Функції в JS є іменованими сутностями. Як і з іншими типами об'єктів, на функції посилаються лише тоді, коли вони надаються змінним, властивостям або параметрам.

Розглянемо об'єкт типу Number. Інструкція
213;

```
function doSmthWonderful() {
    alert('does smth Wonderful');
}
```

Не створює функції з ім'ям **doSmthWonderful .** Ключове слово function автоматично створює екземпляр **Function** і присвоює його властивості window, ім'я якого збігається з ім'ям функції, як показано в наступному прикладі:

```
doSmthWonderful = function(){
    alert('does smth Wonderful');
}
```

Це аналогічно до оголошення змінної з літералом Number

```
aWonderfulNumber = 213;
```

Літерал функції складається з ключового слова function, за яким йде список параметрів, укладений у круглі дужки, і далі слідує тіло функції.

Коли оголошуємо глобальну іменовану функцію, створюється екземпляр Function і присвоюється властивості об'єкта window, що створюється автоматично, з урахуванням так званого імені функції. Сам собою екземпляр Function немає імені, як літерал Number чи String. Це поняття ілюструє рисунок:

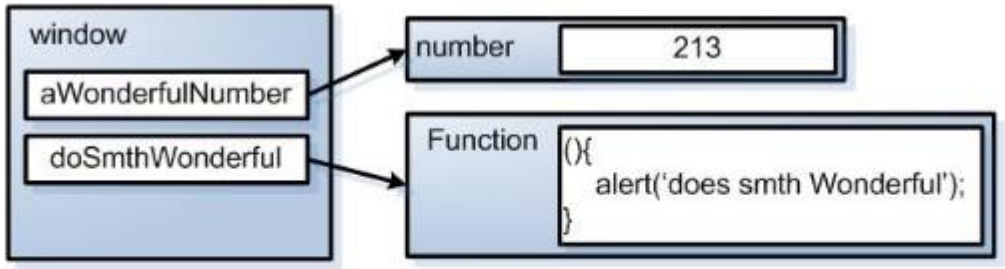


Рисунок 5. Примірник Function є неіменованим об'єктом, таким самим, як значення 213 типу Number або будь-яке інше значення JavaScript . Він називається лише посиланнями, створені йому.

всі наступні інструкції еквівалентні:

```
function hello(){alert('Hi there!'); }
hello = function(){ alert('Hi there!'); }
window.hello = function(){ alert('Hi there!'); }
```

Функції зворотного виклику

Візьмемо як приклад таймер. Ми можемо змусити таймер спрацювати, наприклад через 5 секунд, передавши відповідне значення тривалості методу window.setTimeout(). Але як цей метод повідомити нам про те, що час таймера минув, щоб ми могли виконати необхідні дії після закінчення часу очікування? Це робиться шляхом виклику функції, яку ми надамо.

Розглянемо наступний програмний код:

```
function hello() {alert('Hi there!'); }
setTimeout(hello,5000);
```

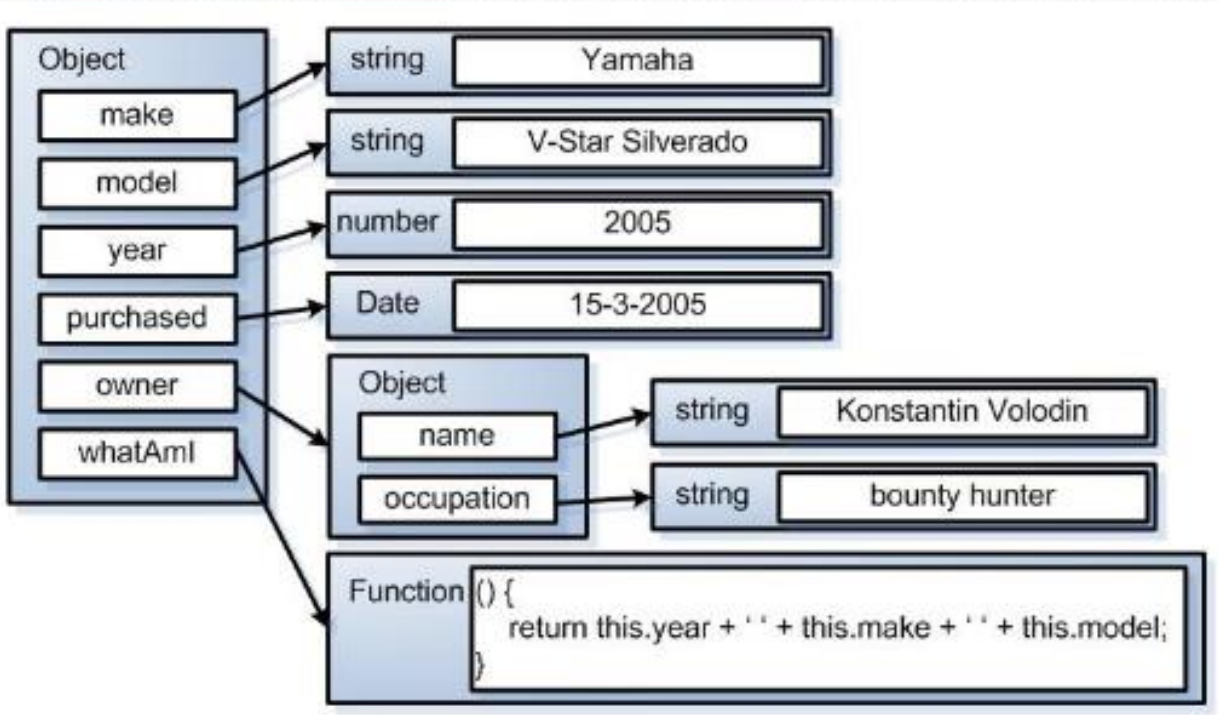
Коли час таймера закінчиться, буде викликана функція hello. Оскільки метод setTimeout() робить зворотний виклик функції у нашому програмному коді, цю функцію називають **функцією зворотного виклику**.

більш витончений спосіб запису цього фрагмента:

```
setTimeout(function() { alert('Hi there!'); },5000);
```

Приклад з мотоциклом, змінимо створення об'єкта:

```
var ride = {
    make: ‘Yamaha’,
model: ‘V-Star Silverado’,
year: 2005,
    purchased: new Date(2005,3,15),
    owner: {
    name: ‘Konstantin Volodin’,
    occupation: ‘bounty hunter’,
}
whatAmI: function() {
    return this.year + ‘ ‘ + this.make + ‘ ‘ + this.mod
}
};
```



Ця модель ясно показує, що функція не є частиною об'єкта Object , вона лише доступна через властивість об'єкта, яка називається whatAmI

Якщо функція викликається через властивість
var bike = ride.whatAmI();
то як контекст функції (посилання this) встановлюється екземпляр об'єкта, на який вказує ride. В результаті в змінну bike записується рядок 2005 Yamaha V-Star Silverado, тому що функція вибирає за допомогою цього значення властивостей об'єкта, за допомогою якого вона була викликана.

```
<html>
<head>
   <title>Function Context Example</title>
   <script>
      var o1 = {handle:'o1'};
      var o2 = {handle:'o2'};
      var o3 = {handle:'o3'};
      window.handle = 'window';
      function whoAmI() {
         return this.handle;
      }
      o1.identifyMe = WhoAmI;
      alert(whoAmI());       // window
      alert(o1.identifyMe()); // o1
      alert(whoAmI.call(o2)); // o2
      alert(whoAmI.apply(o3));    // o3
   </script>
</head>
```

```
<body>
</body>
</html><html>
<head>
   <title>Function Context Example</title>
   <script>
      var o1 = {handle:'o1'};
      var o2 = {handle:'o2'};
      var o3 = {handle:'o3'};
      window.handle = 'window';
      function whoAmI() {
         return this.handle;
      }
      o1.identifyMe = WhoAmI;
      alert(whoAmI());       // window
      alert(o1.identifyMe()); // o1
      alert(whoAmI.call(o2)); // o2
      alert(whoAmI.apply(o3));    // o3
   </script>
</head>
<body>
</body>
</html>
```

Ця послідовність повідомлень ілюструє наступне:

- Якщо функція викликається як глобальна функція, контекстом функції є екземпляр об'єкта window x.
- Якщо функція викликається як властивість об'єкта (o1 у разі), контекстом функції стає цей об'єкт y. Ми могли б сказати, що функція діє як метод цього об'єкта – аналогічно ГО мов. Але це зовсім так.
- Використання методу call() об'єкта Function призводить до того, що контекстом функції стає будь-який об'єкт, отриманий методом call() як перший параметр, - в даному випадку o2 z. У цьому прикладі функція діє як спосіб об'єкта o2 при тому, що вона ніяк не пов'язана з об'єктом o2, навіть як властивість.
- Як і у випадку з методом call(), при використанні методу apply() контекстом функції стає будь-який об'єкт, переданий як перший параметр {. Різниця між цими двома параметрами стає суттєвою, тільки коли параметри передаються функції (що ми не робили для простоти).

Інтернет програмування

лек 7

Лекція 9. Бібліотека jQuery

Зав.каф.КН, PhD Граф М.С.

jQuery — бібліотека JavaScript, що фокусується на взаємодії JavaScript та HTML. Бібліотека jQuery допомагає легко отримувати доступ до будь-якого елемента DOM, звертатися до атрибутів та вмісту елементів DOM, маніпулювати ними. Також бібліотека jQuery надає зручний API для роботи з AJAX.

Крім **jQuery** існують інші бібліотеки для JavaScript'а.

Prototype (prototypejs.org) . Являє собою прообраз усіх сучасних бібліотек JavaScript. Створена та випущена Семом Стівесоном у 2005 році. Включає функціональні можливості DOM, Ajax і обробки подій, а також різні прийоми програмування.

Yahoo UI (developer.yahoo.com/yui) . Як результат власної розробки інтегрованого середовища в компанії Yahoo, оприлюднена в лютому 2006 року. Включає функціональні можливості DOM, Ajax, обробки подій та анімації, а також цілий ряд попередньо побудованих віджетів. (Міні додатків типу календаря, сітки, меню гармошки тощо)

base2 (code.google.com/p/base2) . Створена Діном Едвардсом та випущена у березні 2007 року для підтримки функціональних можливостей DOM та обробки подій. Претендує на популярність своїми спробами реалізувати різні специфікації стандарту W3C в універсальному крос-браузерному вигляді.

One	Two	Three	Four
One	Two	Three	Four
One	Two	Three	Four
One	Two	Three	Four
One	Two	Three	Four
One	Two	Three	Four
One	Two	Three	Four

```
var tables = document.getElementsByTagName("table");
for ( var t = 0; t < tables.length; t++ ) {
    var rows = tables[t].getElementsByTagName("tr");
    for ( var i = 1; i < rows.length; i += 2 )
        if ( !/(^|s)odd(s|$)/.test( rows[i].className ) )
            rows[i].className += " odd";
}
```

jQuery :

```
$("tr:nth-child(odd)").addClass("odd");
```

Нижче наводиться мінімальний html-документ, до якого підключено бібліотеку jQuery:

```
01 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
02 <html>
03   <head>
   04   <meta http-equiv="content-type" content="text/html; charset=windows-1251">
   05   <title>Мінімальний документ, з jQuery</title>
   06   <!-- Підключаємо бібліотеку jQuery версії 1.11.1 -->
   07   <script type="text/javascript"
            08       src="jquery-1.11.1.js"></script>
   09   <script type="text/javascript" >
   10     $(
      11        function(){
      12          alert("Документ повністю завантажено");
      13        }
   14     )
   15   </script>
   16  </head>
17  <body bgcolor="#cacaff">
18    <h1> Мінімальний документ, з jQuery </h1>
19  </body >
20 </html>
```

Принцип роботи jQuery

Основною відмінністю jQuery від інших бібліотек є те, що можна застосовувати деякі дії не лише до окремого елементу, а й до колекції загалом. Тобто. Достатньо виділити необхідну колекцію елементів і застосувати до неї необхідні дії.

Ключем до розуміння роботи jQuery є функція **$()** . Ця функція так чи інакше викликається всіма методами jQuery. Визначення функції **$** можна побачити у лістингу 1. Незалежно від параметрів, переданих у функцію, знак долара поверне список об'єктів, над яким вже визначено всі доступні функції jQuery. Це дозволяє працювати з будь-якими об'єктами – вже існуючими на сторінці, створеними динамічно або отриманими через AJAX – так, ніби це одні й ті самі елементи, що вже існують на сторінці.

Суть jQuery полягає в тому, щоб відбирати елементи HTML-сторінок та виконувати над ними деякі операції.

Синтаксис відбору групи елементів простий: `$(selector) или jQuery(selector)`

Функція $() (псевдонім функції jQuery()) повертає спеціальний об'єкт JS, який містить масив елементів DOM, які відповідають зазначеному селектору.

Припустимо, що нам потрібно реалізувати поступове зникнення всіх елементів . jQuery дозволяє зробити це так:

```
$("div").fadeOut();
```

Особливістю багатьох з цих методів полягає в тому, що після завершення своїх дій вони повертають ту саму групу елементів, готову до виконання іншої операції. Припустимо, що після зникнення до елементів слід додати клас CSS removed. Записати це можна так:

```
$("div").fadeOut().addClass(“removed”);
```

У результаті такі дві інструкції дають ідентичні результати:

```
$("#somediv").html("let's add some text!");
$("#somediv")[0].innerHTML = "let's add some text!";
```

Той самий результат можна отримати за допомогою селектора, який може відібрати кілька елементів:

```
$("div").html("let's add some text!");
або
var elements = $("div");
for (i=0; i<elements.length; i++)
    elements[i].innerHTML = "let's add some text!";
```

Бібліотека jQuery підтримує різні типи селекторів.

Ось кілька прикладів:

$(“p:even”); - цей селектор відбирає всі парні елементи <p>
$(“tr:nth-child(1)”); - цей селектор відбирає перші рядки у всіх таблицях.
$(“body > div”); - цей селектор відбирає елементи , які є прямими нащадками елемента <body>.
$(“a[href$=pdf]”); - цей селектор відбирає посилання на PDF-файли.
$(“body > div:has(a)”); - цей селектор відбирає елементи , які є прямими нащадками елемента <body> та містять посилання.

Повний перелік селекторів ви знайдете за адресою:
http://api.jquery.com/category/selectors/

Лістинг 1. Визначення функції $():

```
$(html)
$(elems)
$(fn)
$(expr, context)
```

Наприклад, можна створити елемент div, який містить параграф із текстом «Привіт!» і додати його до елемента з id="body" таким чином:

```
var my_div = $("<div><p>Привіт!</p></div>");
my_div.appendTo("#body");
```

Або ще коротше:

```
$("<div><p>Привіт!</p></div>").appendTo("#body");
```

Елемент до відпрацювання скрипту:

```
<div id="body"></div>
```

Елемент після відпрацювання скрипту:

```
<div id="body"><div><p>Привет!</p></div></div>
```

Лістинг 1. Визначення функції $():

```
$(html)
$(elems)
$(fn)
$(expr, context)
```

$(elems)

Дозволяє «причепити» всю функціональність jQuery до вже існуючих елементів сторінки (а саме до елементів з об'єктної моделі документа, з DOM). Приклади:

```
$(document.body).css( "background-color", "black" );
$( myForm.elements ).hide();
```

$(expr[, context])

Знайдемо всі елементи p, що знаходяться всередині всіх елементів div на сторінці:

```
$("div > p");
```

Наш документ:

```
<p>один</p> <div><p>два</p></div> <p>три</p>
```

Результат:

```
[<p>два</p>]
```

Знайдемо всі радіокнопки у першій формі на сторінці:

```
$("input:radio", document.forms[0]);
```

Дякую за увагу!