Тема 2. Об’єктна модель JavaScript

2.1. Поняття про об’єкти

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

Створення об'єктів

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

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

Літерал об'єкту

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

```
var empty ={};          // порожній об'єкт (без властивостей)
var point ={x :0,y :0}; // об'єкт з двома властивостями x і y зі значеннями 0
var homer ={          // об'єкт з декількома властивостями
  "name": "Гомер Сімпсон"
  age: 34,
  married: true
};
```

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

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

```
var homer ={
  name: "Гомер Сімпсон"
  age: 34,
```

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

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

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

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

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

Значенням властивості colors є об'єкт {first: "yellow", second: blue }. Тут слід звернути увагу на те, що у об'єкту, який виступає значенням, у кінці відсутня крапка з комою. Це пояснюється тим що в літералі об'єкту усі властивості вказуються через кому і ніяких крапок з комою там не повинно бути використано.

Перевірка, перерахування і видалення властивостей

Для видалення властивостей об'єкту використовується оператор delete.

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

Зверніть увагу, що при видаленні властивості його значення не просто встановлюється в значення undefined, оператор delete дійсно видаляє

властивість з об'єкту. Цикл for in демонструє цю відмінність: він перераховує властивості, яким присвоєно значення undefined, але не перераховує видалені властивості.

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

```
for (var ім'я_змінної in об'єкт){
  ...   //оператори
}
```

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

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

document.write("<br> після видалення властивості : <br>")
for (var name in homer){
  document.write(name + " = " + homer[name] + "<br>");  //виводить імена і значення
    властивостей
}
```

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

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

```
var obj ={};
if ("a" in obj){
```

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

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

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

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

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

Оператор in в цьому випадку гарантує точний результат

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

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

Доступ до властивості через квадратні дужки []

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

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

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

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

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

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

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

```
obj["Моє ім'я"] = "Гомер";
```

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

Методи об'єкту

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

```
var obj ={
  name: "Гомер"

  write_hello: function(){
    document.write("Привіт");
  }
};

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

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

```
var calc ={
  num1: 5,
  num2: 5,
  compute: function( ){
    this.result = this.num1 * this.num2;
  }
};
```

```
calc.compute(); //Обчислюємо скільки буде 5*5?
document.write(calc.result); // Виводимо результат
```

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

Функція конструктор і оператор new

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

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

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

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

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

```
myCar ={
  seats: "leather"
  canDrive: true
};
```

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

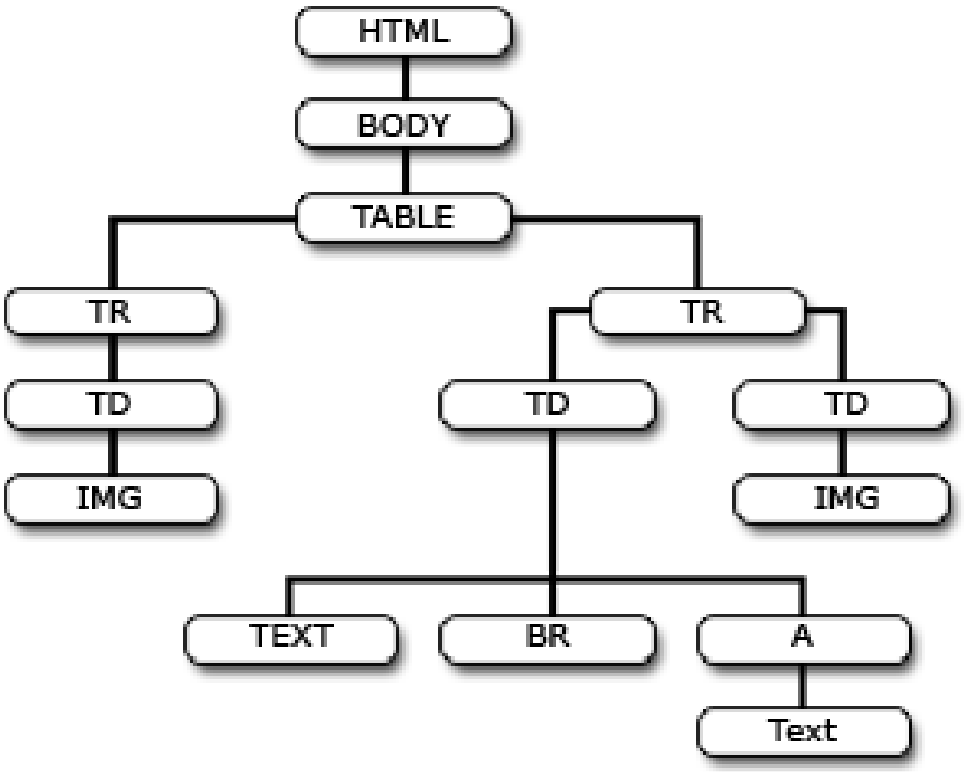
Примітка: при виклику конструктора без аргументів, дужки можна не ставити.

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

7.2. Об’єктна модель JavaScript

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

document.ім’я_об’єкту.ім’я_методу()



Пошук елементів в DOM

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

Потужніші способи пошуку пропонують javascript -біблиотеки.

Пошук по id

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

Наприклад, наступний код змінить колір тексту на блакитній в `div'i c id="dataKeeper"` :

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

Пошук по тегу

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

Наприклад, можна отримати другий елемент(нумерація в масиві йде з нуля) з тегом `li` :

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

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

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

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

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

Отримати усіх нащадків

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

Наприклад, на такому DOM:

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

Такий код:

```
1 var div = document.getElementById('d1')
2 var elems = div.getElementsByTagName('*')
3
4 for(var i=0; i<elems.length; i++) alert(elems[i].id)
```

Виведе послідовність: `ol1, li1, li2`.

Пошук по name: getElementsByName

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

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

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

Інші способи

Існують і інші способи пошуку по DOM: XPath, cssQuery і тому подібне. Як правило, вони реалізуються javascript -библиотеками для розширення стандартних можливостей браузерів.

Також є метод `getElementsByClassName` для пошуку елементів по класу, але він зовсім не працює в IE, тому в чистому вигляді їм ніхто не користується.

Часта друкарська помилка пов'язана з відсутністю букви **s в назві методу** `getElementById`, тоді як в інших методах ця буква є: `getElementsByName`.

Правило тут просте: один елемент - `Element`, багато - `Elements`. **Усі методи** `*Elements*` повертають список вузлів.

• Створення і додавання елементів

Щоб створити новий елемент - використовується метод `document.createElement(тип)`.

```
var newDiv = document.createElement('div')
```

Тут же можна і проставити властивості і зміст створеному елементу.

```
var newDiv = document.createElement('div')
newDiv.className = 'my-class'
newDiv.id = 'my-id'
newDiv.style.backgroundColor = 'red'
newDiv.innerHTML = 'Привіт, світ!'
```

Додавання в DOM

Додати новий елемент до дітей існуючого елементу можна методом `appendChild`, який в DOM є у будь-якого тега.

Код з наступного прикладу додає нові елементи до списку:

```
<ul id="list">
li>Перший елемент</li>
<ul>
```

Список:

- Перший елемент

```
// елемент-список UL
var list = document.getElementById('list')
// новий елемент
var li = document.createElement('LI')
//Властивість innerHTML встановлює або отримує усю розмітку і зміст усередині цього елементу.
li.innerHTML = 'Новий елемент списку'
// додавання в кінець
list.appendChild(li)
```

Метод `appendChild` завжди додає елемент останнім в список дітей.

Додавання в конкретне місце

Новий елемент можна додати не в кінець дітей, а перед потрібним елементом.

Для цього використовується метод `insertBefore` батьківського елементу.

Він працює так само, як і `appendChild`, але приймає другим параметром елемент, перед яким треба вставляти.

```
parentElem.insertBefore(newElem, target)
```

Наприклад, в тому ж списку додамо елементи перед першим `li`.

```
<ul id="list2">
li>Перший елемент</li>
<ul>
```

- Перший елемент

```
// батьківський елемент UL

    var list = document.getElementById('list2')
    // елемент для вставки перед ним (перший LI)
    var firstLi = list.getElementsByTagName('LI')[0]

    // новий елемент
    var newListElem = document.createElement('LI')
    newListElem.innerHTML = 'Новий елемент списку'
```

```
// вставка
list.insertBefore(newListElem, firstLi)
```

Метод insertBefore дозволяє вставляти елемент у будь-яке місце, окрім як в кінець. А з цим справляється appendChild. Так що ці методи доповнюють один одного.

Методу `insertAfter немає, але потрібну функцію легко написати на основі комбінації insertBefore і appendChild`.

Видалення вузла DOM

Щоб прибрати вузол з документу - досить викликати метод `removeChild` з його батька.

```
list.removeChild(elem)
```

Якщо батька немає "на руках", то зазвичай використовують `parentNode`. Виходить так:

```
elem.parentNode.removeChild(elem)
```

Незграбно, але працює.

- # Приклад - показ повідомлення

Зробимо що-небудь поскладніше.

В якості реального прикладу розглянемо додавання повідомлення на сторінку.

Щоб показувалося посередині екрану і було красивіше, ніж звичайний alert.

Повідомлення в HTML -варіанті (як завжди, можна подивитися, натиснувши кнопку) :

```
<style>
 .my - message {
    width :300 px;
    height :110 px;
    background - color:#F08080;
    text - align: center;
    border: 2px groove black;
 }
 .my - message - title {
   height :20 px;
   font -size :120%;
   background - color: red;
 }
 .my - message - body {
   padding: 5px;
   height: 50px;
```

```
}
</style>

<div class="my-message">
  <div class="my - message - title">Заголовок</div>
  <div class="my - message - body">Текст Повідомлення</div>
  <input class="my - message - ok" type="button" value="OK"/>
</div>
```

Як видно - повідомлення вкладене в DIV фіксованого розміру `my-message і складається із заголовка my-message-title, тіла my-message-body` і кнопки OK, яка потрібна, щоб повідомлення закрити.

Крім того, додані трохи простих стилів, щоб якось виглядало.

Показ повідомлення складається з декількох етапів.

1. Створення DOM -елементів для показу повідомлення
2. Позиціонування на екрані
3. Запуск функції видалення повідомлення по кліку на ОКИ

Створення

Для створення скільки-небудь складних структур DOM, як правило, використовують або готові шаблони і метод `cloneNode, що створює копію вузла, або властивість innerHTML`.

Наступна функція створює сполучення з вказаним тілом і заголовком.

```
function createMessage(title, body){
  // (1)
  var container = document.createElement('div')

  // (2)
  container.innerHTML = '<div class="my-message"> \
    <div class="my - message - title">'+title+'</div> \
   <div class="my - message - body">'+body+'</div> \
    <input class="my - message - ok" type="button" value="OK"/> }_
  </div>'

  // (3)
  return container.firstChild
}
```

Як видно, вона поступає досить хитро. Щоб створити елемент за текстовим шаблоном, вона спочатку створює тимчасовий елемент (1), а потім записує (2) його як innerHTML тимчасового елементу (1). Тепер готовий елемент можна отримати і повернути (3).

Позиціонування

Повідомлення позиціонуватимемо абсолютно, в центрі по ширині і на 200 пікселів нижче верхньої межі екрану.

```
function positionMessage(elem){
  elem.style.position = 'absolute'

  var scroll = document.documentElement.scrollTop || document.body.scrollTop
  elem.style.top = scroll + 200 + 'px'

  elem.style.left = Math.floor(document.body.clientWidth/2) - 150 + 'px'
}
```

Не вдаючись до тонкощів позиціонування - помітимо, що для властивості `top` 200 пікселів додаються до поточної вертикальної прокрутки, яку браузер відлічує або від `documentElement` або від `body`, - залежить від DOCTYPE і типу браузеру.

При установці `left` від центру екрану віднімається половина ширини DIV 'а з повідомленням (у нього стоїтьwidth :300).

Закриття

Нарешті, наступна функція вішає на кнопку OK функцію, що видаляє елемент з повідомленням з DOM.

```
function addCloseOnClick(messageElem){

  var input = messageElem.getElementsByTagName('INPUT')[0]

  input.onclick = function(){

    messageElem.parentNode.removeChild(messageElem)

  }

}
```

Зверніть увагу, при отриманні елементу функції не використовують DOM -свойства `previousSibling/nextSibling`.

Цьому є дві причини. Перша - надійність. Ми можемо змінити шаблон повідомлення, вставити додатковий елемент - і нічого не повинно зламатися.

Друга - ця наявність текстових елементів. Властивості `previousSibling/nextSibling` перераховуватимуть їх нарівні з іншими елементами, і доведеться їх фільтрувати.

Запуск

Створимо одну функцію, яка виконує усі необхідні для показу повідомлення операції.

```
function setupMessageButton(title, body){
  // створити
```

```
  var messageElem = createMessage(title, body)
  // позиціонувати
  positionMessage(messageElem)
  // додати обробник на закриття
  addCloseOnClick(messageElem)
  // вставити в документ
  document.body.appendChild(messageElem)
}
```

Протестувати те, що вийшло, нам допоможе наступна кнопка:

```
<input
  type="button"
  value="Показати"
  onclick="setupMessageButton('Привіт!'Ваше повідомлення')"
/>
```

2.2. Обробка подій

Важливою ознакою інтерактивних HTML-сторінок є можливість реакції на дії користувача. Наприклад, натиск на кнопці повинен викликати появу діалогового вікна, або виконання перевірки правильності введених користувачем даних. В JavaScript інтерактивність реалізована за допомогою ***перехвату*** та ***обробки подій***, викликаних в результаті дій користувача. Для цього в теги деяких елементів введені параметри ***обробки подій***. Ім'я параметру обробки події починається з префіксу on, за яким йде назва події. Наприклад, події клік кнопкою миші Click, відповідає параметр обробки події з назвою onClick. Назви та характеристики деяких подій наведені в табл. 7.1.

Таблиця 7.1

Події JavaScript

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

MouseOut	Курсор покидає зону над елементом	onMouseOut

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

Варіант 1:

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

Варіант 2:

```
<script>
    function Go() {
    alert("Hello")
    }
</script>
<td onClick="Go()"> Привіт </td>
```

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