Тема 4. Об’єктна модель Java Script

JavaScript відноситься до об’єктно-орієнтованих мов програмування.

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

Операції з об’єктами JavaScript

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

Створення об'єктів

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

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

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

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

  age: 34,
  married: true
```

```
};
```

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

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

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

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

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

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

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

Значенням властивості colors є об'єкт {first: "yellow", second: blue }. Тут слід звернути увагу на те, що у об'єкту, який виступає значенням, у кінці відсутня

крапка з комою. Це пояснюється тим що в літералі об'єкту усі властивості вказуються через кому і ніяких крапок з комою там не повинно бути використано.

Перевірка, перерахування і видалення властивостей

Для видалення властивостей об'єкту використовується оператор delete.

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

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

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

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

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

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

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

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

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

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

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

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

Оператор in в цьому випадку гарантує точний результат

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

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

Доступ до властивості через квадратні дужки []

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

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

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

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

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

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

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

```
obj["Моє ім'я"] = "Гомер";
```

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

Методи об'єкту

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

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

```
obj.write hello(); //виклик методу
```

Метод повинен мати доступ до даних об'єкту для повноцінної роботи. Для доступу до об'єкту з методу використовується ключове слово this. Воно посилається на об'єкт, в контексті якого викликаний метод і дозволяє звертатися до інших його методів і властивостей : `var calc ={ num1: 5, num2: 5, compute: function( ){ this.result = this.num1 * this.num2;`

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

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

Функція конструктор і оператор new

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

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

```
//визначуваний конструктор
function Car(seats){ this.seats = seats; this.canDrive
  = true;
}
```

//викликаємо конструктор для створення нового об'єкту

```
var myCar = new Car("leather");
```

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

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

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

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

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

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

Обробка подій

Важливою ознакою інтерактивних HTML-сторінок є можливість реакції на дії користувача. Наприклад, натиск на кнопці повинен викликати появу діалогового вікна, або виконання перевірки правильності введених користувачем даних. В JavaScript інтерактивність реалізована за допомогою *перехоплення* та *обробки подій,* викликаних в результаті дій користувача. Для цього в теги деяких елементів введені параметри *обробки подій.* Ім'я параметру обробки події починається з префіксу on, за яким йде назва події. Наприклад, події клік кнопкою миші Click, відповідає параметр обробки події з назвою onClick. Назви та характеристики деяких подій наведені в табл. 4.1.

Таблиця 4.1 **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.

Стандартні об’єкти і функції JavaScript

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

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

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

Відзначимо, що перший елемент масиву має номер 0.

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

Наприклад:

ar1 = new Array(1, 2, 3)

ar2 = new Array(3)

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

a = ar1[2]

ar1[0] = 7

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

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

ar1[100] = 7,

то розмір масиву буде автоматично установлений рівним 101.

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

a = ar1. length

Зручність використання масивів забезпечується рядом методів, представлених в табл. 4.2

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

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

Таблиця 4.2 **Методи об’єкту Array**

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

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

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

push	Додає один або декілька елементів в кінець масиву і повертає останній добавлений елемент. var array = ["one", "two"] // додати елементи "three", "four" var pushed = array.push("three", "four")
reverse	Переставляє елементи масиву в зворотному порядку: перший елемент стає останнім, а останній першим. arr = [1,2,3] a = arr.reverse()
shift	Знищує перший елемент масиву і повертає його значення. var arr = ["мій","маленький", "масив"] var my = arr.shift() // => "мій" alert(arr[0])
slice	Створює перетин масиву в вигляді нового масиву var arr = [1, 2, 3, 4, 5] arr.slice(2) // => [3, 4, 5] arr.slice(1, 4) // => [2, 3, 4]

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

sort	Сортує елементи масиву arr = [1,-1, 0]; a = arr.sort();
unsift	Додає один або більше елементів в початок масиву та повертає нову довжину масиву var arr = ["a", "b"]; unshifted = arr.unshift(-2, -1);

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

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

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

Прочитати або змінити параметри створеного об’єкту Date можливо за допомогою ряду методів.

Найбільш вживані методи показані в табл. 4.3

Таблиця 4.3- Методи об’єкту Date

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

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

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

a = Math.sin(1)

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

a = Math. pow(5,6)

Методи об’єкту Math, що використовуються найбільш часто представлені в табл. 4.4.

Таблиця 4.45 Методи об’єкту Math

Метод	Призначення
abs	Повертає абсолютне значення змінної. Math.abs(-2);
sin, cos, tan	Повертають значення тригонометричних функцій. Аргументи задаються в радіанах.
acos, asin, atan	Повертають значення обернених тригонометричних функцій
exp, log	Повертають значення експоненціальної функції та функції
ceil	Повертає найменше ціле число, більше або рівне значенню аргументу

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

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

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

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

ім 'я об'єкту = new String(Рядок)

Параметром конструктору може бути будь-який рядок. Наприклад:

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

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

h=a.length

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

a =a.toLowerCase();

Таблиця 2.6 Методи об'єкту String

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

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

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

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

Аргументи: string - рядкове представлення числа;

radix - основа системи числення

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

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

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

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

Функція parseFloat аналогічна:

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

Аргументи: strVal - рядок, що представляє числове значення

Використання об’єктів window, document

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

Ім ’я змінної = window.open([lM ’я_ файлу],[Ім ’я_вікна],[Параметри])

Ім’я змінної - це ім’я для звернення до нового вікна в програмі JavaScript.

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

Ім ’явікна - це ім’я, що буде вказане в якості цілі в гіпертекстовому посиланні на це вікно із іншого HTML-документу.

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

Таблиця 2.7 - Основні параметри ві кна

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

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

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

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

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

```
myw.close()
```

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

```
close()
```

Цікавим методом об’єкту window є метод *setTimeout,* за допомогою якого можливо запрограмувати виконання деяких команд після закінчення встановленого терміну часу. Синтаксис методу такий: `setTimeout("Kog_JavaScript",iHTepBan_4acy)`

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

запрограмувати так: `setTimeout("myfunction()",3000)`

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

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

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

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

Інші методи об'єкта window наведено в таблиці 2.8.

Таблиця 2.8 - Методи об'єкта Window

alert()	Викликає вікно сповіщення, яке містить текст повідомлення і клавішу ОК. `window.alert("Вітаю, це сповіщення");`
blur()	Робить вікно неактивним. `win.blur()`
clearInterval()	Припиняє повторне виконання коду заданого setInterval(). `clearlnterval(id);`
clearTimeout()	Відміняє заплановане методом setTimeout() виконання коду. `clearTimeout(id)`
confirm()	Викликає вікно підтвердження що містить текст повідомлення і клавіші ОК і Відміна. Повертає true або false `var x = confirm('Натисніть ОК або Отмена.');`
focus()	Робить вікно активним. `window.focus()`
moveBy()	Зміщує вікно відносно його поточної позиції. `win = window.open();` `win.moveBy(200,100);`

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

print()	Роздруковує вміст поточного вікна. `print();`

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

Пошук елементів на сторінці

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

Потужніші способи пошуку пропонують javascript -библиотеки.

Пошук по id

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

Наприклад, наступний код змінить колір тексту на блакитний в div 'і з id="dataKeeper" : `document.getElementById('dataKeeper').style.color = 'blue'` *Пошук по тегу*

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

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

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

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

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

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

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

Отримати усіх нащадків

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

Наприклад, на такому DOM:

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

Такий код:

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

виведе послідовність: ol1, li1, li2.

Пошук по name.

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

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

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