Лекція13
Browser Object Model і об'єкт Window.
Робота з Document Object Model

13.1 Об’єктна модель браузера (BOM). Об’єкт **window**

Не існує офіційних стандартів для моделі Object Browser Model (BOM).

Оскільки сучасні браузери реалізують (майже) ті ж методи і властивості для JavaScript інтерактивно, їх часто називають методами і властивостями BOM.

Об’єкт **window** підтримується всіма браузерами. Цей об’єкт представляє собою вікно браузера.

Всі глобальні об'єкти JavaScript, функції і змінні автоматично стають членами об'єкта вікна.

Глобальні функції є властивостями об'єктів вікон. Загальні функції — це методи об'єктів вікна.

Навіть об'єкт документа (в HTML DOM) є властивістю віконного об'єкта:

```
window.document.getElementById("header");
```

теж саме за іншого визначення:

```
document.getElementById("header");
```

Для визначення розміру вікна браузера можна використовувати дві властивості.

Обидві властивості повертають розмір пікселя:

- **window.innerHeight** – внутрішня висота вікна браузера (в пікселях)
- **window.innerWidth** – внутрішня ширина вікна браузера (в пікселях)

Вікно браузера (область перегляду) не містить панелі інструментів або смуги прокручування.

Властивості об’єкта **window** для Internet Explorer 8, 7, 6, 5:

- **document.documentElement.clientHeight**

- **document.documentElement.clientWidth**

або

- **document.body.clientHeight**
- **document.body.clientWidth**

Практичне застосування властивостей JavaScript (підтримується всіма браузерами):

```
var w = window.innerWidth
|| document.documentElement.clientWidth
|| document.body.clientWidth;

var h = window.innerHeight
|| document.documentElement.clientHeight
|| document.body.clientHeight;
```

Приклад обчислює висоту та ширину вікна переглядача (не враховуючи панелі інструментів і смугу прокручування).

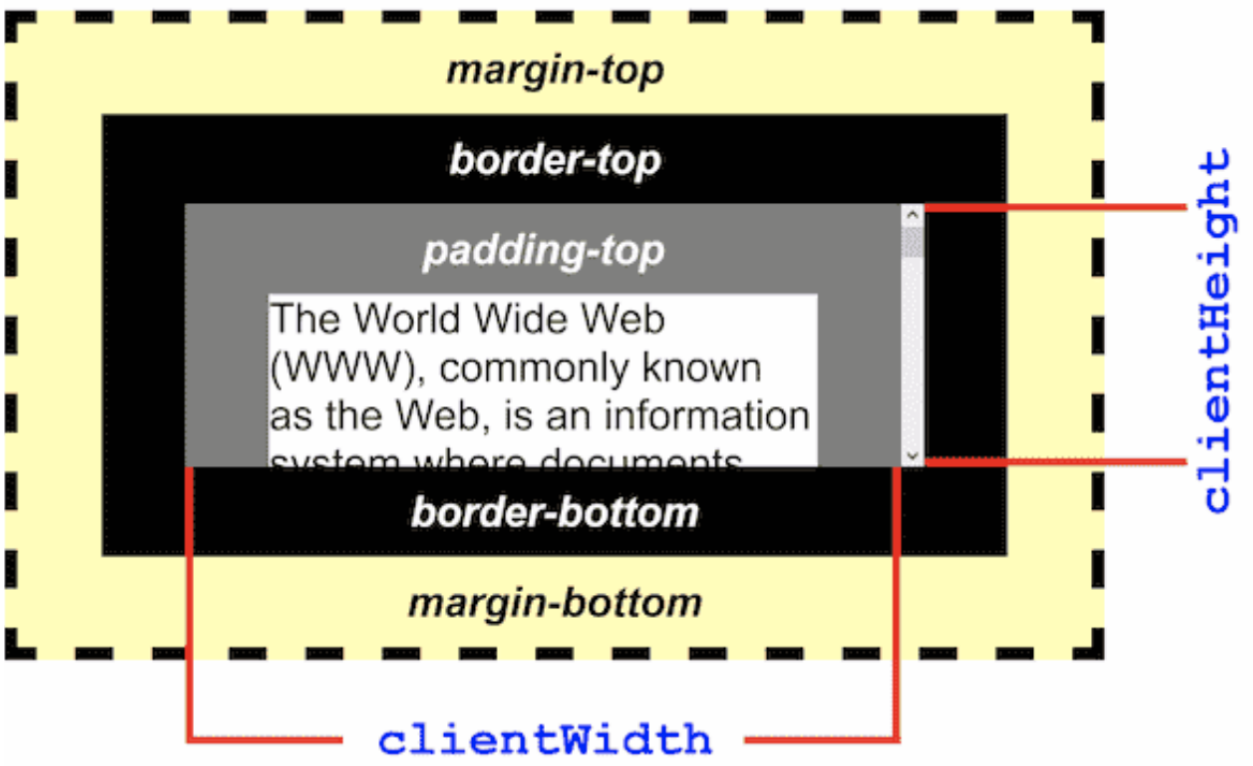
Властивість Element.clientHeight може використовуватися тільки для читання, дорівнює нулю для елементів без CSS або вбудованих в елементи що є контейнерами. В іншому випадку, ця властивість визначає внутрішню висоту елемента в пікселях. Він включає в себе внутрішні відступи навколо елемента (padding), але не включає ширину рамки (border), зовнішні поля (margin) і горизонтальні смуги прокрутки (якщо такі є).

Значення clientHeight може бути обчислено як: CSS height + CSS padding - висота горизонтальної смужки прокрутки (якщо така є).

Якщо clientHeight використовується на кореневому елементі (елементі <html>), або на елементі<body>, якщо документ знаходиться в режимі Quirks (Quirks mode або режим сумісності – режим роботи Content Management System (CMS) веб-браузеров, в якому вони зворотньо сумісні з веб-сторінками, створеними для старих браузерів, в цьому режимі браузер

ігнорує частину правил CSS, імітуючи роботу старого CMS), повертається висота перегляду (за винятком будь-якої смужки прокрутки).

Властивість Element.clientHeight округлює значення до цілого числа. Якщо потрібне дробове значення, слід скористатися методом Element.getBoundingClientRect().



Інші методи об’єкта **window**:

- **window.open()** - відкриває нове вікно
- **window.close()** - закриває поточне вікно
- **window.moveTo()** - переміщує поточне вікно
- **window.resizeTo()** - змінює розмір поточного вікна

13.2 BOM об’єкт Screen

Об’єкт **window.screen** містить інформацію про екран користувача.

Об’єкт **window.screen** можна використовувати без префакса **window**.

Властивості:

- **screen.width**
- **screen.height**
- **screen.availWidth**
- **screen.availHeight**
- **screen.colorDepth**

- **screen.pixelDepth**

Властивість **screen.width** повертає ширину екрана користувача в пікселях:

```
document.getElementById("demo").innerHTML =
"Ширина екрана: " + screen.width;
```

Властивість **screen.height** повертає висоту екрана користувача в пікселях:

```
document.getElementById("demo").innerHTML =
"Висота екрана: " + screen.height;
```

Властивість **screen.availWidth** повертає ширину екрана користувача в пікселях мінус елементи інтерфейсу (на кшталт панелі задач Windows).

В наступному прикладі виводиться досяжна ширина екрана:

```
document.getElementById("demo").innerHTML =
"Досяжна ширина екрана: " + screen.availWidth;
```

Властивість **screen.availHeight** повертає висоту екрана користувача в пікселях мінус елементи інтерфейсу (на кшталт панелі задач Windows).

В наступному прикладі виводиться досяжна висота екрана:

```
document.getElementById("demo").innerHTML =
" Досяжна висота екрана: " + screen.availHeight;
```

Властивість **screen.colorDepth** повертає кількість біт, які використовуються для відображення кольора.

Всі сучасні комп’ютери використовують 24 або 32 апаратних біта для кольорового відображення:

- 24 біта = 16,777,216 різних кольорів - "True Colors"
- 32 біта = 4,294,967,296 різних кольорів - "Deep Colors"

В більш старих комп’ютерах використовувалося16 біт або 65,536 різних кольорів - "High Colors".

В зовсім старих комп’ютерах та мобільних телефонах використовувалося 8 біт або 256 різних кольорів - "VGA colors".

В наступному прикладі виводиться глибина кольору екрана в бітах:

```
document.getElementById("demo").innerHTML =
"Глибина кольору екрана: " + screen.colorDepth;
```

Значення #rrggbb (rgb), яке використовується в HTML, є елементом палітри "True Colors" (16,777,216 кольорів).

Властивість **screen.pixelDepth** повертає глибину пікселя екрана в бітах:

```
document.getElementById("demo").innerHTML =
"Глибина пікселя екрана: " + screen.pixelDepth;
```

В сучасних комп’ютерах глибина кольору та глибина пікселя – це одне й теж саме.

13.3 BOM – Об’єкт Location

Об’єкт **window.location** може використовуватися для отримання адреси (URL) поточної сторінки та перенаправлення браузера на нову сторінку.

Об’єкт **window.location** може записуватися без префікса **window**.

Деякі приклади використання об’єкта **window.location**:

- **window.location.href** повертає посилання (URL) поточної сторінки
- **window.location.hostname** повертає домене ім’я веб-хоста

- **window.location.pathname** повертає шлях та ім’я файла поточної сторінки
- **window.location.protocol** повертає веб-протокол, який було використано для з’єднання (http: або https:)
- **window.location.assign** загружає новий документ

Властивість **window.location.href** повертає URL поточної сторінки.

```
document.getElementById("demo").innerHTML =
"Адреса сторінки: " + window.location.href;
```

Властивість **window.location.hostname** повертає ім’я інтернет хоста поточної сторінки.

```
document.getElementById("demo").innerHTML =
"Ім’я хоста сторінки: " + window.location.hostname;
```

Властивість **window.location.pathname** повертає шлях до поточної сторінки.

```
document.getElementById("demo").innerHTML =
"Шлях сторінки: " + window.location.pathname;
```

Властивість **window.location.protocol** повертає веб-протокол сторінки.

```
document.getElementById("demo").innerHTML =
"Протокол сторінки: " + window.location.protocol;
```

Властивість **window.location.port** повертає номер порта інтернет хоста поточної сторінки.

```
document.getElementById("demo").innerHTML =

"Номер порта: " + window.location.port;
```

Більшість браузерів не показують номера порта за умовчанням (80 для http та 443 для https).

Метод **window.location.assign()** загружає новий документ:

```
<html>

<head>

<script>

function newDoc() {

   window.location.assign("https://msiter.ru")

}

</script>

</head>

<body>


<input type="button" value="Загрузити новий документ"
onclick="newDoc()">


</body>

</html>
```

13.4 BOM – Об’єкт History

Об’єкт **window.history** містить історию сторінок, які відвідав браузер.

Об’єкт **window.history** може записуватися без префікса **window**.

Для забезпечення приватності користувачів, існують деякі обмеження методів отримання доступу JavaScript до цього об’єкту.

Деякі методи:

- **history.back()** – виконує ті ж самі дії, що й за натискання кнопки браузера "Назад"
- **history.forward()** - виконує ті ж самі дії, що й за натискання кнопки браузера "Вперед"

Метод **history.back()** загружає попередній URL в списку сторінок, які були відвідані. В принципі цей метод діє так само як і кнопка браузера "Назад".

В наступному прикладі на сторінці створюється кнопка повернення назад:

```
<html>
<head>
<script>
function goBack() {
    window.history.back()
 }
</script>
</head>
<body>

<input type="button" value="Назад" onclick="goBack()">

</body>
</html>
```

Метод **history.forward()** загружає наступний URL у списку сторінок, які були відвідані.

В принципі цей метод діє так само як і кнопка браузера "Вперед".

В наступному прикладі на сторінці створюється кнопка перехода вперед:

```
<html>
<head>
<script>
function goForward() {
    window.history.forward()
 }
</script>
</head>
<body>

<input type="button" value="Вперед" onclick="goForward()">

</body>
</html>
```

13.5 BOM - Об’єкт Navigator

Об’єкт **window.navigator** містить інформацию про браузер відвідувача сторінки.

Об’єкт **window.navigator** може записуватися без пріфикса **window**.

Деякі приклади застосування об’єкта:

- **navigator.appName**
- **navigator.appCodeName**
- **navigator.platform**

Файли кукіс

Властивість **cookieEnabled** повертає значення true, якщо файл «cookies» (спеціальні файли-мітки) дозволені, інакше повертає значення false:

```
<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML =
"cookiesEnabled встановлено в " + navigator.cookieEnabled;
</script>
```

Властивість **appName** повертає ім’я браузера, як застосунка:

```
<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML =
"navigator.appName - " + navigator.appName;
</script>
```

Слід зауважити, що ім’ям застосунку для браузерів IE11, Chrome, Firefox та Safari є "Netscape".

Властивість **appCodeName** повертає кодове ім’я браузера:

```
<p id="demo"></p>

<script>
```

```
document.getElementById("demo").innerHTML =
"navigator.appCodeName - " + navigator.appCodeName;
</script>
```

Кодовим ім'ям для браузерів Chrome, Firefox, IE, Safari та Opera є "Mozilla".

Властивість **product** повертає ім'я CMS браузера:

```
<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML =
"navigator.product - " + navigator.product;
</script>
```

Слід зауважити, що не можна повністю покладатися на дані цієї властивості. Більшість браузерів повертає ім'я "Gecko"

Властивість **appVersion** повертає інформацію про версії браузера:

```
<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML =
navigator.appVersion;
</script>
```

Властивість **userAgent** повертає заголовок користувацького агента, який було надіслано браузером серверу:

```
<p id="demo"></p>


<script>

document.getElementById("demo").innerHTML =
navigator.userAgent;

</script>
```

Властивість **platform** повертає платформу браузера (операційну систему):

```
<p id="demo"></p>


<script>

document.getElementById("demo").innerHTML =
navigator.platform;

</script>
```

Властивість **language** повертає мову браузера:

```
<p id="demo"></p>


<script>

document.getElementById("demo").innerHTML =
navigator.language;

</script>
```

Властивість **onLine** повертає true, якщо браузер подключено до Інтернет:

```
<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML =
navigator.onLine;
</script>
```

Метод **javaEnabled()** повертає true, якщо обробка Java увімкнута:

```
<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML =
navigator.javaEnabled();
</script>
```

Слід зауважити, що інформація з об'єкта **navigator** може бути некоректною, тому її не можна використовувати для визначення версии браузера, оскільки:

- різні браузери можуть використовувати одне й теж саме ім'я
- дані об'єкта navigator можуть бути змінені власником браузера
- деякі браузери надають невірні ідентифікатори, для того щоб обходити перевірку сайтів
- браузери не можуть повідомляти про нові операційні системи, які були впроваджені після впровадження цих браузерів

13.6 BOM – Спливаючі вікна повідомлень

Існує три види спливаючих вікон JavaScript: з попередженням, з підтвердженням та з пропозицією.

Вікно с попередженням часто використовується для доведення певної інформації до користувача.

Коли з'являється вікно з попередженням, то, для того, щоб продовжити роботу, користувачу слід натиснути кнопку "OK".

Синтаксис:

```
window.alert("Текст попередження");
```

Метод **window.alert()** може записуватися без префікса **window**.

Наприклад:

```
alert("Це вікно з попередженням!");
```

Вікно с підтвердженням часто використовується, якщо потрібно, щоб користувач прийняв певні умови або відмовився.

Коли з'являється вікно з підтвердженням, то, для того, щоб продовжити роботу, користувачу слід натиснути кнопку "OK", або кнопку "Скасувати". В першому випадку вікно повертає значение true, в другому – false.

Синтаксис:

```
window.confirm("Текст підтвердження");
```

Метод **window.confirm()** може записуватися без префікса **window**.

```
if (confirm("Наттисніть кнопку!")) {
    txt = "Ви натиснули OK!";
} else {
    txt = " Ви натиснули Скасувати!";
```

```
}
```

Вікно з пропозицією ввода часто використовується, якщо потрібно, щоб користувач перед входом на сторінку ввів якесь значення.

За появи вікна с пропозицією ввода, для продовження роботи, користувачеві слід ввести певне значення та натиснути кнопку "OK" або "Скасувати".

За натискання кнопки "OK" вікно повертає значення, яке було введено користувачем. За натискання кнопки "Скасувати" – **null**.

Синтаксис:

```
window.prompt("Текст пропозиції","Текст за умовчанням");
```

Метод **window.prompt()** може записуватися без префікса **window**.

```
var person = prompt("Введіть Ваше ім’я", "Котигорошку");

if  (person == null || person == "") {
    txt = "Користувач скасував введення.";
} else {
    txt = "Привіт " + person + "! Як справи?";
}
```

Для виведення у спливаючому вікні повідомлення в кілька рядків слід використовувати комбінацію символів "\n".

```
alert("Привіт\nЯк справи?");
```

13.7 BOM - Події за таймером

JavaScript може виконуватися через завданий період часу. Таке виконання є виконанням за таймером.

Об'єкт **window** дозволяє виконувати код через попередньо визначений період часу або через певні часові інтервали. Такі часові інтервали називаються подіями за таймером.

Існує два ключових метода, які дозволяють таке виконання коду:

- **setTimeout(*функція, мілісекунди*)** — виконує функцію після завданого в мілісекундах терміну.
- **setInterval(*функція, мілісекунди*)** — є методом аналогічним до попереднього, але виклик функції постійно повторюється.

Оба методи — **setTimeout()** та **setInterval()** — є методами об'єкта **window**.

Синтаксис метода **setTimeout()**:

```
window.setTimeout(функція, мілісекунди);
```

Метод **window.setTimeout()** може записуватися без префікса **window**.

Перший аргумент — функція, яка викликається.

Другий аргумент завдає час в мілісекундах, який має пройти до виклику функції.

У наступному прикладі після натискання кнопки через 3 секунди з'являється спливаюче вікно з повідомленням "Привіт":

```
<button onclick="setTimeout(myFunction, 3000)">Try
it</button>


<script>
function myFunction() {
    alert('Привіт');
}
</script>
```

Метод **clearTimeout()** зупиняє виконання функції, яка була викликана методом **setTimeout()**.

Синтаксис метода **clearTimeout()**:

```
window.clearTimeout(timeoutVariable);
```

Метод **window.clearTimeout()** може записуватися без префікса **window**.

Метод **clearTimeout()** використовує змінну, яка повертається методом **setTimeout()**:

```
myVar = setTimeout(функція, мілісекунди);
clearTimeout(myVar);
```

Якщо функцію, яка спрацьовує за таймером, ще не було викликано, то її виконання можна зупинити викликом метода **clearTimeout()**.

Наступний приклад є аналогічним до попереднього, але в ньому додано кнопку "Зупинити":

```
<button onclick="myVar = setTimeout(myFunction,
3000)">Запуск</button>

<button onclick="clearTimeout(myVar)">Зупинити</button>
```

Метод **setInterval()** регулярно повторює виклик функції через завданий інтервал чачу.

```
window.setInterval(функція, мілісекунди);
```

Метод **window.setInterval()** може записуватися без префікса **window**.

Перший аргумент — функція, яка викликається.

Другий аргумент завдає час в мілісекундах, який має пройти між повтореннями виклику функції.

В наступному прикладі викликається функція myTimer, яка виводить поточний час, кожну секунду:

```
var myVar = setInterval(myTimer, 1000);


function myTimer() {
    var d = new Date();
    document.getElementById("demo").innerHTML =
d.toLocaleTimeString();
}
```

Метод **clearInterval()** зупиняє виконання функції, яка викликається методом **setInterval()**.

Синтаксис:

```
window.clearInterval(timerVariable);
```

Метод **window.clearInterval()** може записуватися без префікса **window**.

Метод **clearInterval()** використовує змінну, яка повертається методом **setInterval()**:

```
myVar = setInterval(функція, мілісекунди);
clearInterval(myVar);
```

Наступний приклад є аналогичним до попереднього, але в ньому додано кнопку "Зупинити час":

```
<p id="demo"></p>
```

```
<button onclick="clearInterval(myVar)">Зупинити час
</button>



<script>

var myVar = setInterval(myTimer, 1000);

function myTimer() {

    var d = new Date();

    document.getElementById("demo").innerHTML =
d.toLocaleTimeString();

}

</script>
```

13.8 BOM - JavaScript Cookies

Cookies дозволяють зберігати користувацьку інформацію у веб-сторінках.

Cookies – це дані, які зберігаються у невеликих текстових файлах на комп'ютері користувача.

Після того, як веб-сервер надіслав веб-сторінку користувачеві, з'єднання закривається, і сервер забуває всю інформацію про користувача.

Cookies призначені для того, щоб вирішити проблему того "як запам'ятати інформацію про користувача сайта":

- Коли користувач відвідує веб-сторінку, його дані (наприклад, ім'я) можуть зберігатися в cookies.
- За наступного відвідування цієї сторінки цим користувачем, cookies будуть " пам'ятати " його дані.

Cookies зберігаються у вигляді пари *ім'я-значення*:

```
username = John Doe
```

Коли браузер надсилає запит на веб-сторінку серверу, до запиту додаються cookies, пов'язані з цією сторінкою. Таким чином сервер отримує необхідні дані, що дозволяє «запам'ятати» інформацію про користувача.

Слід зауважити, що жоден з прикладів наведених у цьому розділі не працюватиме, якщо у браузері вимкнено підтримку файлів cookie.

JavaScript може створювати, читати і видаляти cookie за допомогою document.cookie.

Можна створювати cookie за допомогою JavaScript наступним чином:

```
document.cookie = "username=John Doe";
```

За умовчанням, cookies видаляються, коли браузер закривається. Можна додати дату (в UTC), коли зупиняється дія cookies:

```
document.cookie = "username=John Doe; expires=Thu, 18 Dec
2013 12:00:00 UTC";
```

За допомогою параметра *path* можна вказати браузеру, за яким шляхом належать cookies. За замовчання, cookies належать до поточної сторінки:

```
document.cookie = "username=John Doe; expires=Thu, 18 Dec
2013 12:00:00 UTC; path=/";
```

Прочитати cookies за допомогою JavaScript можна наступним чином:

```
var x = document.cookie;
```

document.cookie повертає всі cookies в одному рядку: "*cookie1=значення; cookie2=значення; cookie3=значення;*".

Змінюються cookies за допомогою JavaScript так само, як і створюються (при цьому старі cookies переписуються):

```
document.cookie = "username=John Smith; expires=Thu, 18 Dec
2013 12:00:00 UTC; path=/";
```

Під час видалення cookies не потрібно визначати для них значення.

Для видалення cookies достатньо в параметрі *expires* задати дату, яка вже пройшла:

```
document.cookie = "username=; expires=Thu, 01 Jan 1970
00:00:00 UTC; path=/;";
```

Потрібно впевнитися, що параметр *path* відповідає завданому в cookies шляху, щоб не видалити не той cookies. Деякі браузери не дозволяють видаляти cookies, якщо не задано параметра *path*.

Значення властивості **document.cookie** здається звичайним текстовим рядком, проте якщо записати в **document.cookie** цілий рядок **cookie**, а потім зчитати його, можна побачити лише пару "*ім'я-значення*".

За встановлення нового **cookie**, попередні **cookie** не затираются. Нові додаються до **document.cookie**, і якщо знову прочитати **document.cookie**, то отримаємо щось подібне до " **cookie** = значення; **cookie** = значення;".

Таким чином, якщо потрібно знайти значення певного **cookie**, слід написати JavaScript функцію, яка буде шукати значення потрібного **cookie** в рядку **cookies**.

Приклади використання cookies

Ниже наведено приклад JavaScript кода, який створює cookies для збереження імені відвідувача.

Коли користувач вперше заходить на веб-сторінку, йому пропонують ввести своє ім'я, яке потім зберігається в cookies.

За наступного відвідування користувачем цієї ж сторінки, буде виведено повідомлення з привітанням.

Для работи цього приклада потрібно створити три функції JavaScript:

1. Функцію для встановлення значення cookies

2. Функцію для читання значення cookies
3. Функцію для перевірки значення cookies

Функція для встановлення значення cookies

Спочатку напишемо функцію, яка буде зберігати ім'я відвідувача у змінній *cookie*:

```
function setCookie(cname, cvalue, exdays) {
    var d = new Date();
    d.setTime(d.getTime() + (exdays*24*60*60*1000));
    var expires = "expires="+ d.toUTCString();
    document.cookie = cname + "=" + cvalue + ";" + expires + ";path=/";
}
```

Пояснення до прикладу:

- Аргументами функції є: ім'я cookie (cname), значення cookie (cvalue) та кількість днів до закінчення дії cookie (exdays).
- Функція встановлює cookies додаючи до одного рядка ім'я cookie, значення та дату його закінчення.

Функція для читання значення cookies

Наступною створюємо функцію, яка повертає значення завданого cookies:

```
function getCookie(cname) {
    var name = cname + "=";
    var decodedCookie = decodeURIComponent(document.cookie);
    var ca = decodedCookie.split(';');
    for(var i = 0; i < ca.length; i++) {
        var c = ca[i];
```

```
        while (c.charAt(0) == ' ') {
            c = c.substring(1);
        }
        if (c.indexOf(name) == 0) {
            return c.substring(name.length, c.length);
        }
    }
    return "";
}
```

Розглянемо як працює ця функція:

- В якості аргумента приймаємо ім’я cookie (cname).
- Створюємо змінну (name) з текстом, який слід шукати (cname + "=").
- Декодуємо рядок з cookies для обробки спеціальних символів, наприклад, '$'
- Разбиваємо значення document.cookie за символом ";" та розміщуємо результат у масиві ca (ca = decodedCookie.split(';')).
- В циклі обходимо масив ca (i = 0; i < ca.length; i++) та зчитуємо кожне значення до змінної c (c = ca[i]).
- Якщо cookie знайдено (c.indexOf(name) == 0), повертаємо його значення (c.substring(name.length, c.length).
- Якщо cookie не знайдено, повертаємо порожній рядок "".

Функція для перевірки значення cookies

Останньою створюємо функцію, яка перевіряє, чи встановлено cookie.

Якщо cookie встановлено, то виводиться привітання, інакше виводиться спливаюче вікно із пропозицією користувачеві ввести своє ім’я, яке зберігається протягом 365 днів за допомогою функції **setCookie()**:

```
function checkCookie() {
    var username = getCookie("username");
    if (username != "") {
```

```
        alert("З поверненням, " + username);
    } else {
        username = prompt("Введіть, будь-ласка, ваше ім’я:",
"");
        if (username != "" && username != null) {
            setCookie("username", username, 365);
        }
    }
}
```

Якщо з’єднати всі фрагменти отримаємо наступний код:

```
function setCookie(cname, cvalue, exdays) {
    var d = new Date();
    d.setTime(d.getTime() + (exdays * 24 * 60 * 60 * 1000));
    var expires = "expires="+d.toUTCString();
    document.cookie = cname + "=" + cvalue + ";" + expires +
";path=/";
}

function getCookie(cname) {
    var name = cname + "=";
    var ca = document.cookie.split(';');
    for(var i = 0; i < ca.length; i++) {
        var c = ca[i];
```

```
        while (c.charAt(0) == ' ') {
            c = c.substring(1);
         }
        if (c.indexOf(name)  == 0) {
            return c.substring(name.length, c.length);
         }
    }
    return "";
}

function checkCookie() {
    var user = getCookie("username");
    if (user != "") {
        alert("з поверненням" + user);
    } else {
        user = prompt("Введіть, будь-ласка, ваше ім’я:",
"");
        if (user != "" && user != null) {
            setCookie("username", user, 365);
        }
    }
}
```

В наведеному вище прикладі функція **checkCookie()** запускається одразу після завантаження сторінки.

13.9 JS AJAX

AJAX надає розробнику наступні можливості:

- Прочитати дані з веб-сервера після того, як сторінка була завантажена
- Оновлювати веб-стрінку без її перезавантаження
- Відправляти дані на веб-сервер у фоновому режимі

Приклад AJAX

HTML сторінка:

```
<!DOCTYPE html>
<html>
<body>

<div id="demo">
  <h2>Попросимо AJAX змінити цей текст</h2>
  <button type="button" onclick="loadDoc()">Змінити контент</button>
</div>

</body>
</html>
```

HTML сторінка містить секцію та кнопку <button>.

Секція використовується для виведення інформації з сервера.

Кнопка <button> викликає (якщо на неї натиснули) JavaScript функцію.

Функція запитує дані з сервера та виводить їх.

Функція loadDoc():

```
function loadDoc() {
```

```
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
     document.getElementById("demo").innerHTML =
this.responseText;
    }
  };
  xhttp.open("GET", "ajax_info.txt", true);
  xhttp.send();
}
```

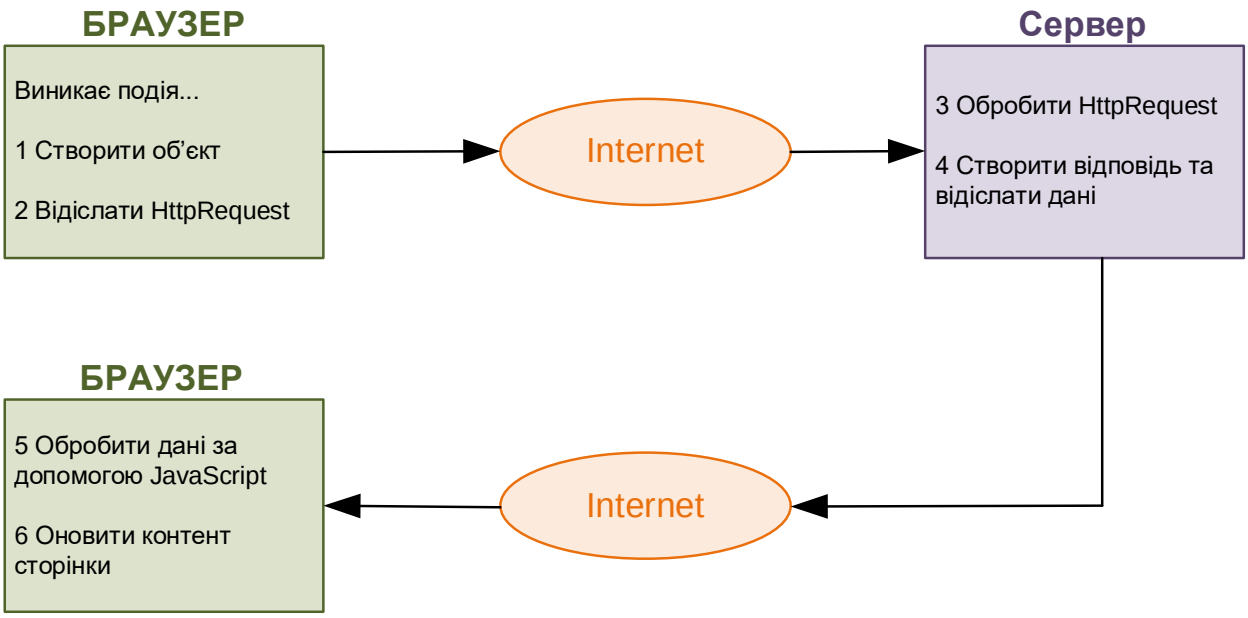
AJAX – від англ. Asynchronous JavaScript And XML.

AJAX - це комбінація кількох технологій:

- Об’єкта XMLHttpRequest вбудованого в браузер для формування запиту на отримання/відправлення даних
- JavaScript та HTML DOM для відображення або використання даних

Застосунки , які використовують AJAX для представлення даних мають використовувати XML, проте так само для цих цілей можна використовувати звичайний текст або JSON (JavaScript Object Notation – текстовий формат обміну даними між комп'ютерами, який дає змогу описувати об'єкти та інші структури даних).

AJAX дозволяє веб-сторінкам оновлюватися асинхроно, обмінюватися даними з сервером, можна оновлювати частину завантаженої веб-сторінки без перезавантаження її повністю.



Розглянемо докладніше роботу AJAX

1. На веб-сторінці виникає подія (наприклад завантаження сторінки або натискання кнопки).
2. JavaScript створює об’єкт XMLHttpRequest.
3. Об’єкт XMLHttpRequest відправляє запит на сервер.
4. Сервер обробляє запит.
5. Сервер відсилає назад до веб-сторінки відповідь.
6. Ця відповідь читається JavaScript.
7. JavaScript виконує необхідні дії (наприклад, оновлення сторінки).

13.10 JS HTML DOM

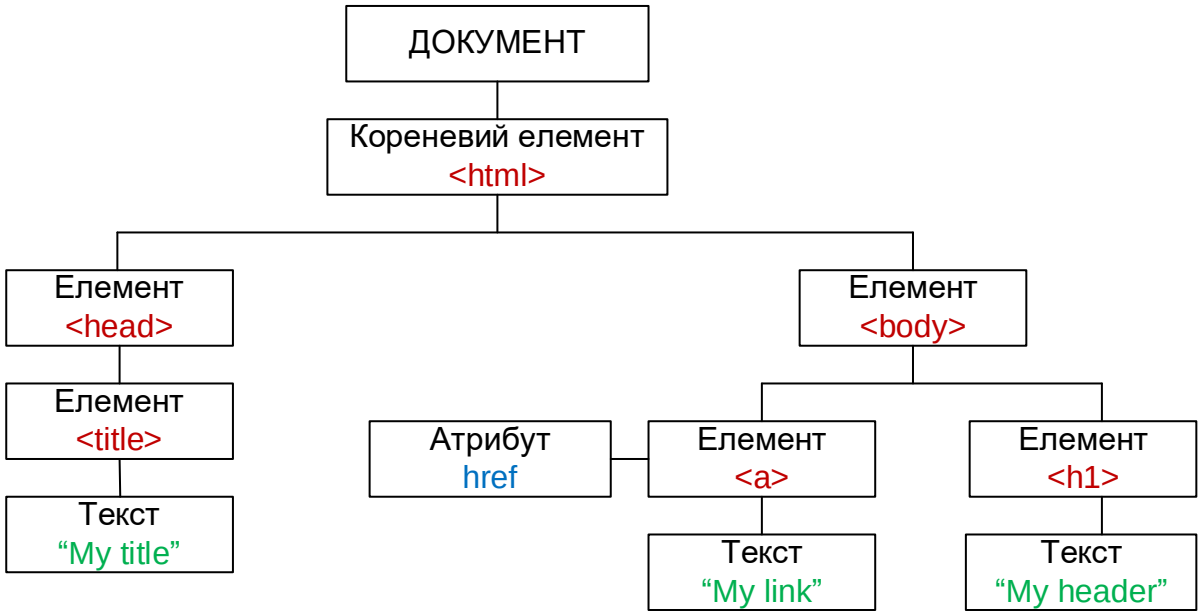
Завдяки HTML DOM програми на JavaScript можуть отримувати доступ та змінювати всі елементи HTML документа.

Об'єктна модель документа (Document Object Model, DOM) з'єднує веб-сторінки зі скриптами або мовами програмування, представляючи структуру документа, наприклад HTML, яка представляє веб-сторінку в пам'яті. Зазвичай це стосується JavaScript, хоча моделювання HTML, SVG або XML документів як об'єктів не є основними задачами програмування JavaScript.

DOM є документом з логічним деревом. Кожна гілка дерева закінчується у вузлі, який містить певний об'єкт. Методи DOM дозволяють програмувати доступ до дерева. За допомогою них можна змінити структуру документа, стиль або зміст.

Після завантаження веб-сторінки браузер створює об’єктную модель документа (анг. Document Object Model) цієї сторінки.

DOM модель HTML має вигляд дерева об’єктів:



Завдяки DOM моделі JavaScript отримує всі можливості, необхідна для створення динамічності HTML:

- JavaScript може змінювати всі HTML елементи на сторінці
- JavaScript може змінювати всі HTML атрибути на сторінці
- JavaScript може змінювати всі CSS стилі на сторінці
- JavaScript може видаляти існуючі HTML елементи та атрибути
- JavaScript може додавати нові HTML елементи та атрибути
- JavaScript може реагувати на всі можливі HTML події на сторінці
- JavaScript може створювати нові HTML події на сторінці

DOM є офіційним стандартом консорціуму W3C (World Wide Web Consortium), який визначає спосіб доступу до документів в Internet.

Стандарт W3C DOM розподіллено на 3 різні частини:

- Core DOM – стандартна модель для всіх типів документів
- XML DOM – стандартна модель для XML документів
- HTML DOM – стандартна модель для HTML документів

HTML DOM – це стандартна об’єктна модель та програмний інтерфейс для HTML документів., який визначає:

- HTML елементи як об’єкти
- Властивості всіх HTML елементів
- Методи для доступу до всіх HTML елементів
- Події для всіх HTML елементів

Іншими словами, HTML DOM – це стандарт того, як отримувати, змінювати, додавати або видаляти HTML елементи.

13.11 DOM - Методи та властивості

HTML DOM методи – це **дії**, які можна виконувати з елементами HTML.

HTML DOM властивості – це **значення** елементів HTML, які можна встановлювати або змінювати.

Отримати доступ до HTML DOM можна за допомогою JavaScript (або інших мов програмування).

В DOM всі HTML елементи визначаються як **об'єкты**.

Програмний інтерфейс – це властивості та методи кожного об'єкта.

Властивість – це значення, які можна прочитати або встановити (на кшталт зміни вмісту елемента HTML).

Метод – це дії, які можна виконувати (на кшталт додавання або видалення елемента HTML).

В наступному прикладі змінюється вміст (**innerHTML**) елемента **<p>** з атрибутом **id="demo"**:

```
<html>
<body>

<p id="demo"></p>

<script>
 document.getElementById("demo").innerHTML = "Hello World!";
</script>

</body>
</html>
```

В наступному прикладі **getElementById** – це метод, а **innerHTML** – властивість.

Метод getElementById

Найлегший спосіб отримання доступу до елемента HTML – використати ідентифікатор ***id*** елемента.

У попередньому прикладі метод **getElementById** використовує ***id="demo"***, для находження потрібного елемента.

Властивість innerHTML

Найлегший спосіб прочитати вміст елемента – це скористатися властивістю **innerHTML**, яка є корисною тим, що дозволяє прочитати або змінити вміст будь-якого елемента HTML, включаючи <html> або <body>.

DOM - Об'єкт документа

Об'єкт документа **document** є батьком всіх інших об'єктів веб-сторінки. Цей об'єкт представляє веб-сторінку.

Якщо потрібно отримати доступ до певного елемента на HTML сторінці, то перш за все слід звернутися до об'єкта **document**.

Поиск елементів HTML

Метод	**Опис**
document.getElementById(*id*)	Пошук елемента за ідентифікатором id
document.getElementsByTagName(*ім'я*)	Пошук елемента за ім'ям тега
document.getElementsByClassName(*ім'я*)	Пошук елемента за ім'ям класса

Зміна елементів HTML

Метод	Опис
елемент.innerHTML = *новий вміст*	Змінює внутрішній вміст елемента
елемент.атрибут = нове значення	Змінює значення атрибута елемента
елемент.setAttribute(*атрибут, значення*)	Змінює значення атрибута елемента
елемент.style.*властивість = новий стиль*	Змінює стиль елемента

Додавання та видалення елементів

Метод	Описание
document.createElement(*елемент*)	Створює HTML елемент
document.removeChild(*елемент*)	Видаляє HTML елемент
document.appendChild(*елемент*)	Додає HTML елемент
document.replaceChild(*елемент*)	Заміняє HTML елемент
document.write(*текст*)	Записує у зовнішній потік HTML

Додавання обробника події

Метод	Опис
document.getElementById(*id*).onclick = function(){*код*}	Додає код обробника для події onclick

Пошук об’єктів HTML

Перший HTML DOM рівень 1 (1998) визначал 11 HTML об’єктів, наборів об’єктів та властивостей. Усі вони ще є актуальними в HTML5.

До HTML DOM рівень 3 були додані нові об’єкти, набори та властивості.

Властивість	Опис	DOM
document.anchors	Повертає всі елементи <a>, які мають атрибут name	1
document.applets	Повертає всі елементи <applet> (Запрещено в HTML5)	1
document.baseURI	Повертає абсолютний базовий URI документа	3
document.body	Повертає елемент <body>	1
document.cookie	Повертає кукі документа	1
document.doctype	Повертає doctype документа	3
document.documentElement	Повертає елемент <html>	3

Властивість	Опис	DOM
document.documentMode	Повертає режим, який використовується браузером	3
document.documentURI	Повертає URI документа	3
document.domain	Повертає доменне ім’я сервера документа	1
document.domConfig	Устаріле. Повертає конфігурацію DOM	3
document.embeds	Повертає всі елементи <embed>	3
document.forms	Повертає всі елементи <form>	1
document.head	Повертає елемент <head>	3
document.images	Повертає всі елементи <img>	1
document.implementation	Повертає реалізацію DOM	3
document.inputEncoding	Повертає кодування документа (набір символів)	3
document.lastModified	Повертає дату та час оновлення документа	3
document.links	Повертає всі елементи <area> та <a>, які мають атрибут href	1

Властивість	Опис	DOM
document.readyState	Повертає статус завантаження документа	3
document.referrer	Повертає URL-адресу документа, за якою завантажено поточний документ	1
document.scripts	Повертає всі елементи <script>	3
document.strictErrorChecking	Повертає значення, чи виконується перевірка помилок чи ні	3
document.title	Повертає елемент <title>	1
document.URL	Повертає повну URL документа	1

13.12 DOM – Зміна HTML

HTML DOM дозволяє JavaScript *змінювати вміст HTML елементів*.

JavaScript може створювати динамічний HTML контент.

Для прямого запису в потік вивода HTML в JavaScript можна застосовувати метод **document.write()** :

```
<!DOCTYPE html>
<html>
<body>

<script>
 document.write(Date());
```

```
</script>


</body>
</html>
```

Слід зауважити, якщо використати метод **document.write()** після повного завантаження документа, він перезапише вміст всього документа.

Найпростіший спосіб змінити вміст HTML елемента – скористуватися властивістю **innerHTML**.

Синтаксис:

```
document.getElementById(id).innerHTML = новий HTML код
```

Наступний приклад змінює вміст елемента **<p>**:

```
<html>
<body>


<p id="p1">Всім привіт!</p>


<script>
 document.getElementById("p1").innerHTML = "Новий текст!";
</script>


</body>
</html>
```

Розглянемо роботу приклада:

- HTML документ містить елемент **<p>** з атрибутом ***id="p1"***
- Використовуємо HTML DOM для отримання доступу до елемента з ***id="p1"***
- JavaScript замінює вміст (**innerHTML**) елемента рядком "Новий текст!"

Наступний приклад змінює вміст елемента **<h1>**:

```
<!DOCTYPE html>
<html>
<body>

<h1 id="id01">Попередній заголовок</h1>

<script>
var element = document.getElementById("id01");
element.innerHTML = "Новий заголовок";
</script>

</body>
</html>
```

Розглянемо роботу приклада:

- HTML документ містить елемент **<h1>** з атрибутом ***id="id01"***
- Використовуємо HTML DOMдля отримання доступу до елемента з ***id="id01"***
- JavaScript замінює вміст (**innerHTML**) елемента рядком"Новий заголовок"

HTML DOM дозволяє змінювати значення атрибутів HTML елементів.

Синтаксис:

```
document.getElementById(id).атрибут = нове значення
```

Наступний приклад змінює значення атрибута **src** елемента **<img>**:

```
<!DOCTYPE html>
<html>
<body>

<img id="myImage" src='smiley.gif'>

<script>
 document.getElementById("myImage").src = "landscape.jpg";
</script>

</body>
</html>
```

Розглянемо роботу приклада:

- HTML документ містить елемент **<img>** з атрибутом ***id="myImage"***
- Використовуємо HTML DOM для отримання доступа до елемента з ***id="myImage"***
- JavaScript змінює значення атрибута ***src*** цього елемента з "smiley.gif" на "landscape.jpg"

13.13 DOM – Зміна CSS

HTML DOM дозволяє JavaScript змінювати стиль HTML елементів.

Синтаксис:

```
document.getElementById(id).style.Властивість = новий стиль
```

В наступному прикладі змінюється стиль елемента <**p**>:

```
<html>
<body>

<p id="p2">Hello World!</p>

<script>
document.getElementById("p2").style.color = "blue";
</script>

</body>
</html>
```

Використання подій

HTML DOM дозволяє виконувати код за виникнення будь-якої події.

Події генеруються браузером, за певної реакції оператора на HTML елемент або за інших випадків:

- на елементі натиснули маніпулятором
- сторінку повністю завантажено
- поле ввода було змінено

В наступному прикладі змінюється стиль елемента HTML з ***id="id1"***, за натискання кнопки користувачем:

```
<!DOCTYPE html>
```

```
<html>

<body>


<h1 id="id1">Заголовок 1</h1>


<button type="button"
onclick="document.getElementById('id1').style.color =
'red'">

 Натисни тут!

</button>


</body>

</html>
```

13.14 DOM – Анімація

Розглянемо створення HTML анімаційних ефектів за допомогою JavaScript.

Для демонстрації створення HTML анімації за допомогою JavaScript будемо використовувати наступну веб-сторінку:

```
<!DOCTYPE html>

<html>

<body>


<h1>Перша JavaScript анімація</h1>


<div id="animation">Анімація буде відбуватися тут</div>
```

```
</body>
</html>
```

Анімаціямає відбуватися відносно певного елемента-контейнера.

```
<div id="container">
  <div id="animate"> Анімація буде відбуватися тут </div>
</div>
```

Елемент-контейнер має бути створений зі стилем "**position: relative**".

А сам елемент, в якому будет происходить анимация, должен быть создан со стилем "**position: absolute**".

```
#container {
    width: 400px;
    height: 400px;
    position: relative;
    background: yellow;
}
#animate {
    width: 50px;
    height: 50px;
    position: absolute;
    background: red;
}
```

Анімація JavaScript здійснюється шляхом програмування послідовних змін стилю елемента. Зміни викликаються за таймером. За незначних інтервалів таймера анімація виглядає неперервною.

Базовий код-схема:

```
var id = setInterval(frame, 5);

function frame() {
    if (/* перевірка для завершення */) {
        clearInterval(id);
    } else {
        /* код, який змінює стиль елемента */
    }
}
```

Повний код створення анімації JavaScript:

```
function myMove() {
    var elem =  document.getElementById("animate");
    var pos = 0;
    var id = setInterval(frame, 5);
    function frame() {
        if (pos == 350) {
            clearInterval(id);
        } else {
```

```
            pos++;
            elem.style.top = pos + 'px';
            elem.style.left = pos + 'px';
        }
    }
}
```

13.15 DOM – Події

HTML DOM дозволяє JavaScript реагувати на HTML події:

Так, наприклад для виконання певного кода за натискання користувачем на елемент, слід додати код JavaScript до атрибуту події HTML елемента:

```
onclick=код JavaScript
```

Приклади HTML подій:

- користувач натискає мишею
- веб-сторінку повністю завантажено
- зображення завантажено
- курсор миши наведено на елемент
- значення в полі ввода змінено
- HTML форму відправлено
- натиснено клавішу клавіатури

В наступному прикладі змінюється вміст змінюється вміст елемента **<h1>** за натискання на цьому елементі мишею:

```
<!DOCTYPE html>
<html>
<body>
```

```
<h1 onclick="this.innerHTML = Click!'">Click on this
text!</h1>


</body>

</html>
```

В наступному прикладі JavaScript функція викликається обробником події:

```
<!DOCTYPE html>

<html>

<body>


<h1 onclick="changeText(this)">Натисни на цей текст!</h1>


<script>

function changeText(id) {

    id.innerHTML = "Click!";

}

</script>


</body>

</html>
```

Для назначення HTML елементу обробника подій потрібно використовувати відповідний **атрибут події**.

В наступному прикладі назначається обробник події **onclick** (натискання) для елемента кнопки. За натискання користувачем на кнопку виконується функція *displayDate*:

```
<button onclick="displayDate()">Натисни сюди!</button>
```

HTML DOM дозволяє назначити HTML елементу обробника подій за допомогою JavaScript.

В наступному прикладі за обробник події **onclick** елемента с ***id="myBtn"*** назначається функція *displayDate*, яка буде виконуватися за натискання користувачем кнопки.

```
<script>
 document.getElementById("myBtn").onclick = displayDate;
</script>
```

Події onload та onunload

Події **onload** та **onunload** спрацьовують під час заходження користувача на сторінку або коли він полишає її.

Подія **onload** може використовуватися, наприклад, для перевірки типа та версії користувацького браузера та завантаження відповідного варіанта веб-сторінки, на підставі цієї інформації.

Також події **onload** та **onunload** можуть використовуватися для роботи з файлами cookie.

```
<body onload="checkCookies()">
```

Подія onchange

Подія **onchange** часто використовується разом з функціями перевірки полів ввода.

Наведемо приклад використання події **onchange**. Функція *upperCase()* викликатиметься за зміни користувачем вмісту поля ввода:

```
<input type="text" id="fname" onchange="upperCase()">
```

Події onmouseover та onmouseout

Події **onmouseover** та **onmouseout** можуть використовуватися для виклику певних функцій за наведення користувачем курсора миши на HTML елемент або відведення курсора від елемента:

Події onmousedown, onmouseup та onclick

Події **onmousedown**, **onmouseup** та **onclick** – це група подій, пов'язаних з натисканням кнопки миши. За натискання кнопки миши спочатку виникає подія **onmousedown**, потім, коли кнопка миши відтпускається, виникає подія **onmouseup**, та, коли всі ці події завершено, виникає подія **onclick**.

13.16 Цікаві презентації та додаткова інформація

https://faculty.ksu.edu.sa/sites/default/files/javascript-dom.pdf

http://latemar.science.unitn.it/segue_userFiles/2022WebArch/10a-DOM-JS.pdf

Browser Object Model

The **Browser Object Model** (BOM) is used to interact with the browser.

The default object of browser is window means you can call all the functions of window by specifying window or directly. For example:

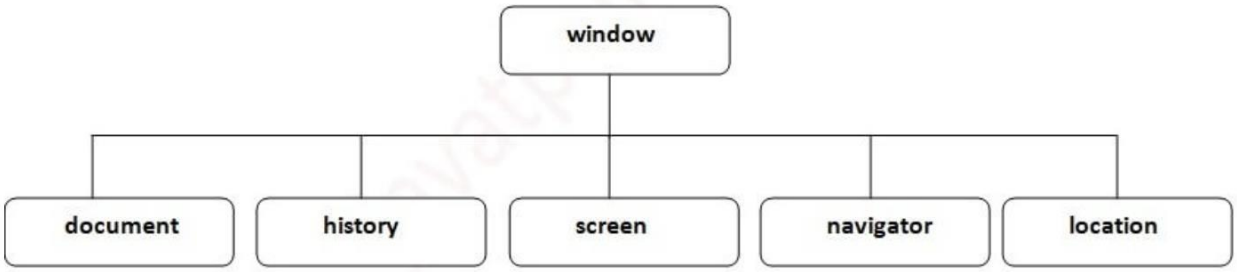
```
window.alert("hello javatpoint");
```

is same as:

```
alert("hello javatpoint");
```

You can use a lot of properties (other objects) defined underneath the window object like document, history, screen, navigator, location, innerHeight, innerWidth,

Note: The document object represents an html document. It forms DOM (Document Object Model).



Window Object

The window object represents a window in browser. An object of window is created automatically by the browser.

Window is the object of browser, it is not the object of javascript. The javascript objects are string, array, date etc.

Note: if html document contains frame or iframe, browser creates additional window objects for each frame.

Methods of window object

The important methods of window object are as follows:

Method	Description
alert()	displays the alert box containing message with ok button.
confirm()	displays the confirm dialog box containing message with ok and cancel button.
prompt()	displays a dialog box to get input from the user.
open()	opens the new window.
close()	closes the current window.

setTimeout()	performs action after specified time like calling function, evaluating expressions etc.

Example of alert() in javascript

```
<script type="text/javascript">
function msg(){
 alert("Hello Alert Box");
}
</script>
<input type="button" value="click" onclick="msg()"/>
```

Example of confirm() in javascript

It displays the confirm dialog box. It has message with ok and cancel buttons.

```
<script type="text/javascript">
function msg(){
var v= confirm("Are u sure?");
if(v==true){
alert("ok");
}
else{
alert("cancel");
}

}
</script>

<input type="button" value="delete record" onclick="msg()"/>
```

Example of prompt() in javascript

It displays prompt dialog box for input. It has message and textfield.

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

```
function msg(){
var v= prompt("Who are you?");
alert("I am "+v);

}
</script>

<input type="button" value="click" onclick="msg()"/>
```

Example of open() in javascript

It displays the content in a new window.

```
<script type="text/javascript">
function msg(){
open("http://www.javatpoint.com");
}
</script>
<input type="button" value="javatpoint" onclick="msg()"/>
```

Example of setTimeout() in javascript

It performs its task after the given milliseconds.

```
<script type="text/javascript">
function msg(){
setTimeout(
function(){
alert("Welcome to Javatpoint after 2 seconds")
},2000);

}
</script>

<input type="button" value="click" onclick="msg()"/>
```

JavaScript History Object

The **JavaScript history object** represents an array of URLs visited by the user. By using this object, you can load previous, forward or any particular page.

The history object is the window property, so it can be accessed by:

window.history

Or,

history

Property of JavaScript history object

There are only 1 property of history object.

No.	Property	Description
1	length	returns the length of the history URLs.

Methods of JavaScript history object

There are only 3 methods of history object.

No.	Method	Description
1	forward()	loads the next page.
2	back()	loads the previous page.
3	go()	loads the given page number.

Example of history object

Let's see the different usage of history object.

```
history.back();//for previous page
history.forward();//for next page
```

```
history.go(2);//for next 2nd page
history.go(-2);//for previous 2nd page
```

JavaScript Navigator Object

The **JavaScript navigator object** is used for browser detection. It can be used to get browser information such as appName, appCodeName, userAgent etc.

The navigator object is the window property, so it can be accessed by:

```
window.navigator
```

Or,

```
navigator
```

Property of JavaScript navigator object

There are many properties of navigator object that returns information of the browser.

No.	Property	Description
1	appName	returns the name
2	appVersion	returns the version
3	appCodeName	returns the code name
4	cookieEnabled	returns true if cookie is enabled otherwise false
5	userAgent	returns the user agent
6	language	returns the language. It is supported in Netscape and Firefox only.
7	userLanguage	returns the user language. It is supported in IE only.

8	plugins	returns the plugins. It is supported in Netscape and Firefox only.
9	systemLanguage	returns the system language. It is supported in IE only.
10	mimeTypes[]	returns the array of mime type. It is supported in Netscape and Firefox only.
11	platform	returns the platform e.g. Win32.
12	online	returns true if browser is online otherwise false.

Methods of JavaScript navigator object

The methods of navigator object are given below.

No.	Method	Description
1	javaEnabled()	checks if java is enabled.
2	taintEnabled()	checks if taint is enabled. It is deprecated since JavaScript 1.2.

Example of navigator object

Let’s see the different usage of history object.

```
<script>
document.writeln("<br/>navigator.appCodeName: "+navigator.appCodeName)
;
document.writeln("<br/>navigator.appName: "+navigator.appName);
document.writeln("<br/>navigator.appVersion: "+navigator.appVersion);
document.writeln("<br/>navigator.cookieEnabled: "+navigator.cookieEnabled)
;
```

```
document.writeln("<br/>navigator.language: "+navigator.language);
document.writeln("<br/>navigator.userAgent: "+navigator.userAgent);
document.writeln("<br/>navigator.platform: "+navigator.platform);
document.writeln("<br/>navigator.onLine: "+navigator.onLine);
</script>
```

```
navigator.appCodeName: Mozilla
navigator.appName: Netscape
navigator.appVersion: 5.0 (Windows NT 6.2; WOW64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/37.0.2062.124 Safari/537.36
navigator.cookieEnabled: true
navigator.language: en-US
navigator.userAgent: Mozilla/5.0 (Windows NT 6.2; WOW64)
AppleWebKit/537.36
(KHTML, like Gecko) Chrome/37.0.2062.124 Safari/537.36
navigator.platform: Win32
navigator.onLine: true
```

JavaScript Screen Object

The **JavaScript screen object** holds information of browser screen. It can be used to display screen width, height, colorDepth, pixelDepth etc.

The navigator object is the window property, so it can be accessed by:

```
window.screen
```

Or,

```
screen
```

Property of JavaScript Screen Object

There are many properties of screen object that returns information of the browser.

No.	Property	Description
1	width	returns the width of the screen

2	height	returns the height of the screen
3	availWidth	returns the available width
4	availHeight	returns the available height
5	colorDepth	returns the color depth
6	pixelDepth	returns the pixel depth.

Example of JavaScript Screen Object

Let’s see the different usage of screen object.

```
<script>
document.writeln("<br/>screen.width: "+screen.width);
document.writeln("<br/>screen.height: "+screen.height);
document.writeln("<br/>screen.availWidth: "+screen.availWidth);
document.writeln("<br/>screen.availHeight: "+screen.availHeight);
document.writeln("<br/>screen.colorDepth: "+screen.colorDepth);
document.writeln("<br/>screen.pixelDepth: "+screen.pixelDepth);
</script>
```

```
screen.width: 1366
screen.height: 768
screen.availWidth: 1366
screen.availHeight: 728
screen.colorDepth: 24
screen.pixelDepth: 24
```

https://msiter.ru/tutorials/javascript/js_htmldom_events