1. Структура програмного забезпечення Internet

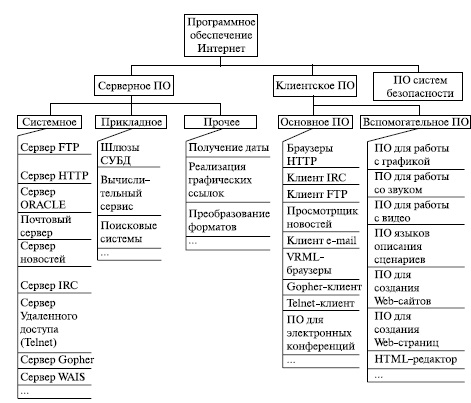
Программное обеспечение Internet состоит из трех видов программ: серверное ПО, клиентское ПО и ПО систем безопасности. Серверное ПО устанавливается на хост-компьютерах, клиентское - на локальных ЭВМ (т. е. на ЭВМ клиентов). ПО систем безопасности может не соответствовать архитектуре "клиент-сервер".

Системы безопасности либо устанавливаются на клиентской ЭВМ или только на хост-ЭВМ, либо для них выделяется отдельная ЭВМ, на которой устанавливаются специализированные программы, обеспечивающие безопасность, - "брандмауэры", или "firewall". Программы безопасности являются аналогами такой стены между ЭВМ пользователей и Интернетом. Они могут быть настроены так, чтобы полностью разрывать связь между ЭВМ и Интернетом, могут допускать одностороннюю связь (например, разрешена только отправка почты с локальной ЭВМ) или разрешать двустороннюю связь только для определенных видов работ. Входная и выходная информация в таких программах проходит через фильтры, которые, например, могут быть настроены на выявление вирусов, на пропуск файлов, не превышающих заданных размеров, или файлов определенного типа, на запрет связи с определенными IP-адресами и т.д.

В клиентском программном обеспечении необходимо обратить внимание на VRML-браузеры, позволяющие просматривать виртуальные миры, работать в трех измерениях. VRML-браузеры - это клиентское программное обеспечение. В Internet есть несколько сайтов с серверами VRML и своеобразными виртуальными мирами.

Как на клиентских ЭВМ, так и на хостах могут быть размещены программы, расширяющие возможности серверов и клиентов. Для написания таких программ применяются специальные алгоритмические языки: HTML, PHP, система программирования CGI, Java, Java-script, Perl, SSI и др. При использовании таких программ на сервере должно быть установлено соответствующее программное обеспечение.

Программирование глобальных вычислительных сетей - сложное направление, в котором применяются приведенные выше алгоритмические языки на основе правил работы, определяемых протоколами TCP/IP и соответствующих сервисов Интернета.



1. Структура HTML-документу

Язык HTML состоит из **тегов**. Каждый тег является элементом разметки гипертекста, и выполняет свою определенную функцию. Одни теги нужны для создания структуры html-документа, указания различной информации о веб-странице, другие – для форматирования и создания элементов на веб-странице, например, чтобы создать кнопку, необходим тег <button>Кнопка</button>, а чтобы перенести текст на новую строку – тег <br>.

Теги бывают **парными**, например тег <button>...</button>, и **одиночными**, например, тег <br>. **Парные теги** состоят из *начального* или *открывающего* тега, и *конечного* или *закрывающего* тега. Закрывающий тег записывается со слэшем.

**Каркас html-документа**

<!doctype html>

<html>

<head>

<meta charset="utf-8">

<title>Название веб-страницы</title>

</head>

<body>

</body>

</html>

### Тип документа

В первой строке html-документа принято указывать тип документа. Делать это нужно обязательно, потому что существует несколько версий языка html, и у каждой есть свои правила и стандарты. Если браузеру не сообщить о версии HTML, то он не будет знать какому стандарту нужно следовать при отображении веб-страницы, что может привести к ее некорректному отображению.

При вёрстке веб-страниц удобнее всего использовать пятую версию языка – **HTML5**. Для этой версии тип документа записывается таким образом: <!doctype html>.

### Теги верхнего уровня

Каркас html-документа образуют теги верхнего уровня <html>, <head>и <body>.

Теги <head>и <body>определяеют заголовок и тело html-документа.

### Теги заголовка документа

В заголовке документа размещается различная информация о веб-странице: кодировка, название, описание, ключевые слова, пути к файлам со стилями, скриптами и т.д. Эта информация, кроме названия, на веб-странице не отображается.

#### Кодировка html-документа

<meta charset="utf-8">

Кодировку html-документа указывают при помощи тега <meta>и его атрибута charset.

В качестве значения атрибута charset принято использовать кодировку **utf-8**, так как она включает в себя символы всех языковых групп.

#### Название html-документа

Название или заголовок html-документа записывается между тегами <title>. Данный тег является обязательным, а также очень полезным как для пользователей, так и для самой веб-страницы.

### Теги тела документа

В теле документа можно размещать теги, предназначенные для разметки и форматирования содержимого веб-страницы. Все что будет записано в этом контейнере, отобразится на веб-странице в окне браузера.

1. Службові теги

В HTML присутствует ряд тегов, называемых служебными тегами. Они используются для хранения сторонних скриптов, стилей или метаданных, т.е. данных предназначенных для браузеров и поисковых систем. Так благодаря служебным элементам 'style' и 'script' появляется возможность размещать таблицы стилей CSS или писать программный код JavaScript непосредственно в исходном html-коде страницы. А метаданные, содержащиеся в элементах 'base', 'link' и 'meta', сообщают браузеру служебную информацию, необходимую для работы с данными самой страницы, а также связи с внешними объектами, данные которых используются при формировании конечного документа. Хранятся служебные элементы в заголовке документа 'head'. Исключение составляет только элемент 'script', который при необходимости может располагаться практически в любом месте html-кода.

1. Управління створенням HTML-документів

**HTML документ** - это обычный текстовой файл c расширением .html, который содержит HTML-код. HTML документ иначе ещё называют **HTML файлом**.

Чтобы из обычного текстового файла (с расширением .txt) сделать HTML документ, нужно [сменить расширение](https://puzzleweb.ru/html/00_teacher2.php#a1) с .txt на .html.

Открыть файл, сохраненный с расширением .html, для просмотра в браузере можно двумя способами:

1. Кликнуть по нему два раза левой кнопкой мыши, в этом случае HTML документ откроется в том браузере, который выбран по умолчанию.
2. Кликнуть по нему ПКМ → навести курсор на открыть с помощью... → выбрать нужный браузер.

При получении HTML документа браузер будет отображать его в виде веб-страницы (**веб-страница** - это то, что отображается в окне браузера).

Открывая HTML файл браузером, в окне браузера мы не увидим отображения самого HTML-кода, потому что он будут интерпретирован для отображения содержимого страницы в нужном виде.

5. **Події та їх обробка браузером в сценаріях**

События (events) - это действия программного обеспечения, которые происходят при выполнении определѐнных условий Событие, по сути, сообщает об изменении состояния некоторого объекта.

ПРИМЕРЫ СОБЫТИЙ

1. Пользователь кликает мышью на определенный элемент или наводит курсор на определенный элемент.
2. Пользователь нажимает клавишу на клавиатуре.
3. 3. Пользователь изменяет размер или закрывает окно браузера.
4. 4. Завершение загрузки веб-страницы.
5. 5. Воспроизведение видео, пауза или завершение воспроизведения.
6. 6. Произошла ошибка.

**Основные события**

|  |  |
| --- | --- |
| onkeypress | Нажатие любой клавиши (кроме системных) |
| onkeydown | Нажатие любой клавиши |
| onkeyup | Отпускание клавиши |
| onhelp | Вызов подсказки (F1) |

**Фокусные события**

|  |  |
| --- | --- |
| onfocus | Приобретение фокуса |
| onblur | Потеря фокуса |

**Прочие события**

|  |  |
| --- | --- |
| onload | Загрузка документа или рисунка |
| onunload | Выгрузка документа |
| onerror | Возникновение ошибки в процессе загрузки |
| onreset | Сброс значений, введенных пользователем посредством формы |
| onchange | Изменение значения поля формы |
| onsubmit | Передача данных, введенных через форму, на сервер |
| onselectstart | Выделение фрагмента документа |
| ondragstart | Перемещение фрагмента |
| onfilterchange | Смена фильтра |
| onscroll | "Прокрутка" документа в окне |
| onresize | Изменение размеров окна браузера |

|  |  |
| --- | --- |
| **Сценарии обработки событий** | |
| **ПЕРВЫЙ СПОСОБ.** | **ВТОРОЙ СПОСОБ.** |
| события="перечень действий или вызов функции" | <script FOR='имя фрагмента - источника события'  EVENT='название события'  LANGUAGE='язык сценария'>  текст сценария  </script> |
| Как только документ будет загружен, начнет выполняться функция cnange\_bg();, при выгрузке документа (например, при закрытии окна браузера) появится системное окно с текстом: ПОКА! ЗАХОДИТЕ ЕЩЕ!. | При наступлении различных событий в плавающем фрейме, расположенном справа от источника события появляется соответствующая надпись.  Здесь mouse - имя плавающего фрейма, содержимое которого формируется динамически (это переменная top.s) при наступлении того или иного события. |
| **Пример:**  <body onload="cnange\_bg();" onunload="alert('ПОКА! ЗАХОДИТЕ ЕЩЕ!');">...</body> | <script FOR='spiders' EVENT='onmouseover'>  top.s=""+"<html><font color=Red>"+'НАВЕЛИ МЫШКУ'+"  </font></html>";  document.mouse.location='javascript:top.s';  </script> |

6. **Особливості управління з урахуванням подій маніпулятора "миша"**

|  |  |
| --- | --- |
| Событие | Комментарии |
| onclick | Щелчок левой кнопкой мыши. Это событие происходит после щелчка мышью в определенном месте страницы. Его рекомендуется применять, когда вы хотите создать сценарий, взаимодействующий с пользователем. |
| ondblclick | Двойной щелчок левой кнопкой мыши |
| onmouseover | Наведение курсора мыши на заданную область  Это событие похоже на событие onClick, но происходит не после щелчка мышью, а после наведения курсора на определенный элемент страницы. Событие onMouseover можно связать практически с любым объектом Web-страницы (текстом, изображением, кнопками, гиперссылками и т.д.). Его также можно использовать, чтобы повысить уровень интерактивности сценария. |
| onmouseout | Выведение курсора мыши из заданной области  Это событие подобно событию onMouseover, но происходит в тех случаях, когда курсор мыши отводится от объекта. Они возникают только один раз на концевых узлах документа и поднимаются вверх по иерархии элементов вместо возникновения при всех граничных условиях. |
| onmousemover | Перемещение курсора мыши в пределах заданной области  Они возникают только один раз на концевых узлах документа и поднимаются вверх по иерархии элементов вместо возникновения при всех граничных условиях. |
| onmousedown | Нажатие любой кнопки мыши |
| onmouseup | Отпускание кнопки мыши |

Поскольку события onclick и ondblclick могут возникнуть на элементе, который является общим для элементов, на которых кнопка мыши была нажата и отпущена, то данные, два события могут возникнуть на элементах, ***которые не являются концевыми узлами в дереве документа. Все остальные события мыши и клавиатуры всегда начинаются на концевых узлах и поднимаются (всплывают) вверх по иерархии.***

* 1. **Форми та їх використання**

**Створення**[**форми**](http://vo.ukraine.edu.ua/mod/forum/view.php?id=35570)**в HTML**

Форми **використовуються** для організації діалогу між користувачем веб-сервера та програмами обробки даних, розміщених на сервері www. Форми дозволяють бачити набір полей для введення значень користувачами і надсилання їх на веб-сервер.

У складі документа може бути кілька форм, але дані відправляються з поточної форми, що містить кнопку submit(надіслати). При відсутності кнопки дані можуть бути відправленні тільки з однієї форми, використовуючи кнопку Enter.

Приклади: всілякі тести, для реєстрації відвідувачів на сайті, для інтерактивних опитувань і голосувань, для відправлення повідомлення і так далі.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Пояснення | | Атрибути | Приклад |
| **Тег**  **FORM** | <FORM> … . </FORM>  Його вмістом є теги INPUT, TEXTAREA, SELECT | | **name** - ім'я [форми](http://vo.ukraine.edu.ua/mod/forum/view.php?id=35570)  **action** - визначає URL-адресу  **method** - визначає спосіб відправки інформації |  |
| **Текстове**  **поле** | Просте однорядкове текстове поле, в яке можна вводити і редагувати текст задається тегом INPUT | | **name** - ім'я елементу,  **type** - тип елементу  **size** - розмір текстового поля в символах,  **maxlength** - максимальна кількість символів, які можна ввести в поле, **value** - текст, який буде відображатися (його можна стерти), за відсутності цього параметра поле буде порожнім. | <input type = "text"> |
| **Текстове поле для введення пароля** | Всі параметри такі ж, як у простого текстового поля, крім параметра type = "password". | | <input type = "password"> |
| [**Форми**](http://vo.ukraine.edu.ua/mod/forum/view.php?id=35570)**з вибором** | | | | | |
| **Прапорці** | |  | **type** - тип елементу (в даному випадку - checkbox),  **name** - ім'я елемента, вказує програмою оброблювачу [форми](http://vo.ukraine.edu.ua/mod/forum/view.php?id=35570), який пункт вибрав користувач,  **value** - значення елемента, вказує програмою оброблювачу [форми](http://vo.ukraine.edu.ua/mod/forum/view.php?id=35570) значення пункту, який вибрав користувач.  У нашому прикладі обраний пункт англійська, отже, програма-оброблювач отримає: lan1 = "english",  **checked** - ним зазвичай позначають найбільш ймовірні для вибору пункти, користувач клацанням миші може вибрати інші пункти. | <input type = "checkbox "> |
| **Перемикачі** | | можна вибрати тільки один пункт -> значення параметра **name** повинні бути однакові для всіх елементів групи | <input type=" radio "> |

|  |  |  |  |
| --- | --- | --- | --- |
|  | Пояснення | Парамети | |
| **Кнопки** | | |
| **submit** | кнопка відправки вмісту [форми](http://vo.ukraine.edu.ua/mod/forum/view.php?id=35570) web-серверу | * type = "submit" - тип кнопки, * name - ім'я кнопки, * value - напис на кнопці. | |
| **image** | графічна кнопка відправки вмісту [форми](http://vo.ukraine.edu.ua/mod/forum/view.php?id=35570) web-серверу. | * type = "image" - тип графічної кнопки, * nam - ім'я кнопки, * src - адреса картинки для кнопки. | |
| **reset** | кнопка, що дозволяє відновити всі значення за замовчуванням у формі. | * type = "reset" - тип кнопки очищення, * name - ім'я кнопки, * value - напис на кнопці. | |
| **button** | довільна кнопка, її дії призначаються користувачем, тобто сама вона робити нічого не вміє. | **type** - тип кнопки, може приймати значення:   * reset - кнопка очищення [форми](http://vo.ukraine.edu.ua/mod/forum/view.php?id=35570), * submit - кнопка відправки даних, * button - кнопка довільної дії.   **name** - ім'я кнопки,  **value** - напис на кнопці. | |

**Багаторядкове текстове поле(тег textarea )**

Для об'ємних текстів, наприклад для поштових повідомлень, зручно використовувати саме цей елемент.

**Списки, що розкриваються( тег select)**

* 1. **Методи об’єктів браузера**

СПИСОК САМЫХ ИСПОЛЬЗУЕМЫХ МЕТОДОВ DOM API

* document.getElementById(id) - возвращает ссылку на элемент по его идентификатору (ID); идентификатор является строкой. Этот метод возвращает объект, имеющий указанный идентификатор.
* document.getElementsByTagName(name) - метод возвращает живую
* коллекцию элементов HTMLCollection , учитывая имя тэга.
* document.createElement(name) – создание нового элемента.
* parentNode.appendChild(node) - возвращает ссылку на добавленный

узел(элемент)

* element.innerHTML - устанавливает или получает всю разметку и

содержание внутри данного элемента

* element.setAttribute - устанавливает значение указанного атрибута
* элемента
* element.getAttribute - возвращает значение указанного атрибута элемента

element.addEventListener - регистрирует определенный обработчик

**9. Языки сценариев в HTML-документах и их характеристики (скриптовые языки)**

Perl, Python, JS, Bash.

Сценарий, грубо говоря тег <script> </script> в хтмл документе, позволяет отделить сценарную часть от остальной информации.

Характеристики языков:

Javascript: комментарий ставиться с помощью //

Tcl: #

Vbscript: одиночная кавычка '

Общие характеристики скриптовых языков:

1. Компиляция на лету, это значит что читается сразу весь текст программы, а не построчно.
2. Экономичность выражений, избежание развернутых объявлений и высокоуровневой структуры. Меньше пунктуации и минимум шаблонов.
3. Отсутствие объявлений, простые правила видимости данных, все конструкции считаются глобальными, редкие объявления задают для переменной вложенную область видимости.
4. Гибкая динамическая типизация. Значит что тип переменной определяется прямо перед использованием.
5. Легкий доступ к другим программам. Большинство языков программирования предоставляет возможность посылки из одной программы запроса на запуск другой программы. Этот запрос адресуется операционной системе. Чтение/запись файлов, управление процессами, доступ к базе данных, все это предлагают скриптовые языки.
6. Высокоуровневые типы данных, библиотеки, встроены в сам язык, списки, массивы, словари все это сразу встроено в скриптовый язык.

**10. Особенности использования стилей в HTML-документах**

CSS – cascade style sheet, каскадная таблица стилей. Используется для управления видом, положением, элементом веб страницы.

Есть три вида подключения CSS к хтмл документу:

1. Первый это подключения файла связанных стилей.

*<link rel="stylesheet" href="http://htmlbook.ru/mysite.css">*

Здесь в параметре линк мы указываем вид подключаемого файла, и в href ссылку на него.

Это самый удобный, быстрый и универсальный метод подключения стилей.

2. Глобальные стили, это когда мы в head документа, в теге <style> также как и в отдельном ксс файле прописываем стили для элементов. Этот вариант плох, так как увеличивается время загрузки страницы, стиль влияет только на эту страницу.

*<html>*

*<head>*

*<meta charset="utf-8">*

*<title>Глобальные стили</title>*

*<style>*

*H1 {*

*font-size: 120%;*

*font-family: Verdana, Arial, Helvetica, sans-serif;*

*color: #333366;*

*}*

*</style>*

*</head>*

*<body>*

*<h1>Hello, world!</h1>*

*</body>*

*</html>*

3. Внутренние стили, те же самые стили, но их нужно прописывать каждому отдельному элементу.

*<p style="font-size: 120%; font-family: monospace; color: #cd66cc">Пример текста</p>*

Быстро можно что-то исправить, очень неудобно и долго

**11. Атрибуты шрифтов текста HTML документа и способы их замены в сценариях**

Для шрифтов существуют такие атрибуты:

font-family – Семейство шрифтов, то есть выбор самого шрифта

font-style – выбор стиля шрифта, обычное, курсив или наклонный

font-weight – жирность шрифта, от 100 до 900. Bold, lighter, normal

font-size – размер шрифта, в пикселях, пунктах или процентах.

Изменять в сценариях их можно с помощью языка JS и библиотеки JQuery, у этой библиотеке есть методы для изменения атрибутов, размеров и свойств элементов хтмл страницы.

.attr(имя атрибута, значение), для изменения атрибутов, имя это целевой атрибут, второй аргумент это значение.

Есть второй вариант, получаем элемент через один из методов document.getElementBy\*\*\* и далее производим модификацию его атрибутов: document.getElementById(‘newInput’).fontSize = '12px'.

**12. Объект document, методы и описание**

Для работы со структурой DOM в JavaScript предназначен объект document, который определен в глобальном объекте window. Объект document предоставляет ряд свойств и методов для управления элементами страницы.

* getElementById(value): выбирает элемент, у которого атрибут id равен value
* getElementsByTagName(value): выбирает все элементы, у которых тег равен value
* getElementsByClassName(value): выбирает все элементы, которые имеют класс value
* querySelector(value): выбирает первый элемент, который соответствует css-селектору value
* querySelectorAll(value): выбирает все элементы, которые соответствуют css-селектору value

Все методы предназначены для получения элементов с хтмл страницы и дальнейшей ее обработки.

Например чтобы получить элемент с id=newInput, вызываем метод document.getElementById(‘newInput’).

**13. Функції в сценаріях**

Чтобы не повторять один и тот же код во многих местах, придуманы функции. Функции являются основными «строительными блоками» программы.

Пример объявления функции:

function showMessage() {

alert( 'Привет всем присутствующим!' );

}

Вначале идет ключевое слово function, после него *имя функции*, затем *список параметров* в скобках (в примере выше он пустой) и *тело функции* – код, который выполняется при её вызове.

Функция может содержать локальные переменные, объявленные через var. Такие переменные видны только внутри функции.

При вызове функции ей можно передать данные, которые та использует по своему усмотрению. **Параметры копируются в локальные переменные функции.**

Функцию можно вызвать с любым количеством аргументов.

Если параметр не передан при вызове – он считается равным undefined.

Функция может возвратить результат, который будет передан в вызвавший её код. **Для возврата значения используется директива**return**.**

Она может находиться в любом месте функции. Как только до неё доходит управление – функция завершается и значение передается обратно.

**14. Обробка графічних об'єктів**

Пока хз что тут писать. Нужно спрашивать что он имел ввиду. Ну эт относится и к остальным мои вопросам.

**15. Безумовне виконання сценарію**

При подключении  JS скрипта к документу весь код глобальной области выполняется безусловно(объявление функиций, переменных и тд). Так же в языке представлен инструмент само вызывающихся функций.

Самовызывающиеся функции- это функции которые вызывают сами себя, сразу же  после объявления. Анонимные самовызывающиеся функции чаще всего используются для того чтобы "уйти" от глобальной области видимости. Весь код написанный внутри выполняется.

 (function() {

   //body;

  }());

**16. Об'єкти в сценаріях та їх використання**

2 варианта:

1.

|  |
| --- |
| Объекты — это единственный составной тип данных в JavaScript. |

var o = {};

var o = new Object()

Объект представляет собой неупорядоченный набор набор пар вида «ключ-значение». Каждая такая пара называется свойством объекта (функции называются методами), каждое свойство должно иметь уникальное имя, которое может быть строкой или числом. Значение свойства может быть любым: как значением простого типа, так и другим объектом.

Главное отличие объектов от других типов данных заключается в том, что все операции с ними осуществляются по ссылке.

В JavaScript существует три способа создания объектов: с помощью литерала объекта, с помощью конструктора Object и с помощью метода Object.create (последний способ предусмотрен только в стандарте ECMAScript 5).

Пример создания объекта:

var emptyObject = {};

var emptyObject = Object.create(null);

var cat = new Object();

2 .

DOM – это представление документа в виде дерева объектов, доступное для изменения через JavaScript. Большинство действий в javascript выполняется с HTML-страницей. В javascript страница представлена в виде объектной модели DOM.

Любые действия со страницей требуют вызова соответствующего метода DOM.

Язык Js дает ряд функций для работы с DOM.

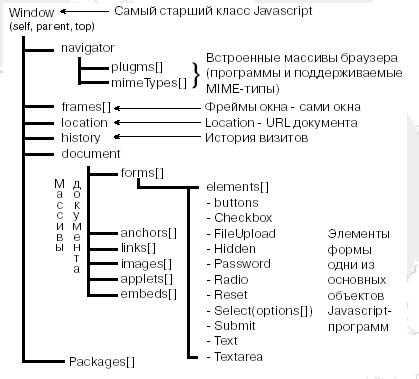
Пример:

**var** container = document.getElementById("container");

container.innerHTML = "Абра-кадабра!";

**17. Ієрархія об'єкту Window, його властивості, методи та використання**

Объектно-ориентированный язык программирования предполагает наличие иерархии классов объектов. В JavaScript такая иерархия начинается с класса объектов ***Window***, т.е. каждый объект приписан к тому или иному окну. Для обращения к любому объекту или его свойству указывают полное или частичное имя этого объекта или свойства объекта, начиная с имени объекта старшего в иерархии, в который входит данный объект:



Window

* Navigator
  + Plagins[]
  + mimeTypes[] - Встроенные массивы браузера поддерживающие Mime типы
* frames[]= фреймы окна – cами окна
* location – Location – Url documenta
* history - история визитов
* Иерархия document

Forms[] = elements[] = { buttons, checkbox , FileUpload , Hidden , Password , Radio , Reset , Select , Submit , Text , TextArea } , Anchors[] , Links[] , images[], applets[] , embeds[] .

* Packeges[]

Разумеется, мы должны иметь возможность получать информацию о различных объектах в этой иерархии и управлять ею. Для этого мы должны знать, как в языке JavaScript организован доступ к различным объектам. Как видно, каждый объект иерархической структуры имеет свое имя. Следовательно, если Вы хотите узнать, как можно обратиться к первому рисунку на нашей HTML-странице, то обязаны сориентироваться в иерархии объектов. И начать нужно с самой вершины. Первый объект такой структуры называется document. Первый рисунок на странице представлен как объект images[0]. Это означает, что отныне мы можем получать доступ к этому объекту, записав в JavaScript document.images[0]. Если же, например, Вы хотите знать, какой текст ввел читатель в первый элемент формы, то сперва должны выяснить, как получить доступ к этому объекту. И снова начинаем мы с вершины нашей иерархии объектов. Затем прослеживаем путь к объекту с именем elements[0] и последовательно записываем названия всех объектов, которые минуем. В итоге выясняется, что доступ к первому полю для ввода текста можно получить, записав:

document.forms[0].elements[0]

**18 . Особливості стилів CSS3**

**CSS (Cascading Style Sheets), или каскадные таблицы стилей**, используются для описания внешнего вида документа, написанного языком разметки. Обычно CSS-стили используются для создания и изменения стиля элементов веб-страниц и пользовательских интерфейсов, написанных на языках HTML и XHTML,

Виды селекторов

**Селекторы** представляют структуру веб-страницы. С их помощью создаются правила для форматирования элементов веб-страницы. Селекторами могут быть элементы, их классы и идентификаторы, а также псевдоклассы и псевдоэлементы.

2.1. Универсальный селектор

Соответствует любому HTML-элементу. Например, \* {margin: 0;} обнулит внешние отступы для всех элементов сайта. Также селектор может использоваться в комбинации с псевдоклассом или псевдоэлементом: \*:after {CSS-стили}, \*:checked {CSS-стили}.

2.2. Селектор элемента

Селекторы элементов позволяют форматировать все элементы данного типа на всех страницах сайта. Например, h1 {font-family: Lobster, cursive;} задаст общий стиль форматирования всех заголовков h1.

2.3. Селектор класса

Селекторы класса позволяют задавать стили для одного и более элементов с одинаковым именем класса, размещенных в разных местах страницы или на разных страницах сайта.

#### 2.4. Селектор идентификатора

Селектор идентификатора позволяет форматировать **один** конкретный элемент. Идентификатор id должен быть уникальным и на одной странице может встречаться только один раз.

#### Дочерний селектор

Дочерний элемент является прямым потомком содержащего его элемента. У одного элемента может быть несколько дочерних элементов, а родительский элемент у каждого элемента может быть только один

### Группировка селекторов

Один и тот же стиль можно одновременно применить к нескольким элементам. Для этого необходимо в левой части объявления перечислить через запятую нужные селекторы:

##### **Как задаются и работают CSS-стили**

1) Стили могут наследоваться от родительского элемента (наследуемые свойства или с помощью значения inherit);

2) Стили, расположенные в таблице стилей ниже, отменяют стили, расположенные в таблице выше;

3) К одному элементу могут применяться стили из разных источников. Проверить, какие стили применяются, можно в режиме разработчика браузера

#### Каскад

**Каскадирование** — это механизм, который управляет конечным результатом в ситуации, когда к одному элементу применяются разные CSS-правила. Существует три критерия, которые определяют порядок применения свойств — правило !important, специфичность и порядок, в котором подключены таблицы стилей.

##### **Правило !important**

Вес правила можно задать с помощью ключевого слова !important, которое добавляется сразу после значения свойства, например, span {font-weight: bold!important;}. Правило необходимо размещать в конец объявления перед закрывающей скобкой, без пробела. Такое объявление будет иметь приоритет над всеми остальными правилами. Это правило позволяет отменить значение свойства и установить новое для элемента из группы элементов в случае, когда нет прямого доступа к файлу со стилями.

##### **Специфичность**

Для каждого правила браузер вычисляет **специфичность селектора**, и если у элемента имеются конфликтующие объявления свойств, во внимание принимается правило, имеющее наибольшую специфичность. Значение специфичности состоит из четырех частей: 0, 0, 0, 0. Специфичность селектора определяется следующим образом:

для id добавляется 0, 1, 0, 0;  
для class добавляется 0, 0, 1, 0;  
для каждого элемента и псевдоэлемента добавляется 0, 0, 0, 1;  
для встроенного стиля, добавленного непосредственно к элементу — 1, 0, 0, 0;

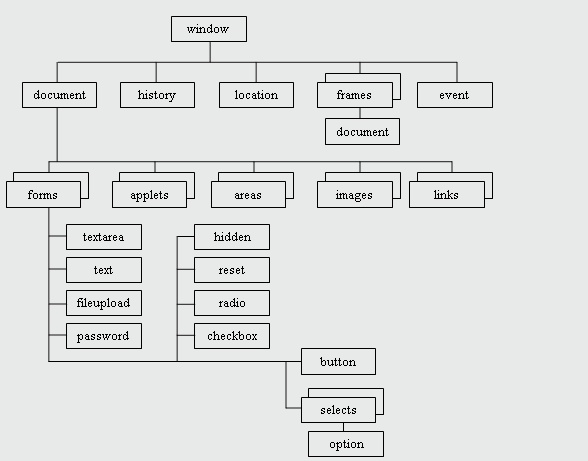
В результате к элементу применятся те правила, специфичность которых больше. Например, если на элемент действуют две специфичности со значениями 0, 0, 0, 2 и 0, 1, 0, 1, то выиграет второе правило.

##### **Порядок подключённых таблиц**

Вы можете создать несколько внешних таблиц стилей и подключить их к одной веб-странице. Если в разных таблицах будут встречаться разные значения свойств одного элемента, то в результате к элементу применится правило, находящееся в таблице стилей, идущей в списке ниже.

**19.Структура та використання ієрархії об'єктів при обробці HTML-документів**

При интерпретации страницы html, браузер создает объекты JS, свойства которых представляют значения параметров тэгов языка html. Объекты хранятся в виде иерархической структуры, отражая структуру документа. Некоторые тэги html являются контейнерами, в которых могут размещаться другие тэги.



**Свойства и методы ключевых объектов.** 

**Window (окно)**. Создается автоматически при запуске браузера.   
Методы:

open () - создание окна . close () - закрытие окна

**Document**. Содержит информацию о загруженной странице. Для каждой страницы создается один объект document. Некоторые его свойства соответствуют параметрам тэга body. Свойство URL содержит адрес загруженного документа.   
Методы:   
write, writeln записывают в документ информацию и тем самым позволяют динамически его создавать.

**Location.** Связан с текущим URL адресом. Его свойства позволяют получать информацию о host-машине, с которой в данный момент связан браузер.   
Свойства:

hostname - содержит имя хоста

port - номер порта, к которому подсоединен браузер на хост-машине

Методы:

reload () - перезагружает в браузер текущую страницу

replace () - загружает в окно браузера страницу, адрес которой задан в качестве его параметра.

**History**. Содержит список адресов html документов, ранее загруженных в браузер.   
Свойства:

current - текущая страница

next - следующая страница

previous - предыдущая страница

Метод:

go () - загружает страницу из списка посещенных.

Текущая страница имеет индекс 0, предыдущая индексируется отрицательными целыми числами, последующая - положительным целым числом.

history.go(-3)

**Form**. Порождается объектом document и сам порождает подчиненные объекты. Ссылка на этот объект осуществляется с помощью переменной, определенной в параметре name тэга form.   
В документе может быть несколько форм, поэтому введено свойство-массив forms, в котором содержатся ссылки на все формы документа.

Пример:

По номеру:

document.forms[0]

document.forms[1]

По имени:

document.forms["form1"]

Если в документе одна форма (с именем form1)

document.form1

Все элементы формы порождают соответствующие объекты, подчиненные объекту родительской формы.

<input type="text" name="txt1">

Варианты обращения к полю txt1:

document.forms[0].txt1

document.forms["form1"].txt1

document.forms.txt1

Каждый объект form имеет также свойство-массив elements, содержащий ссылки на все подчиненные форме элементы в том порядке, в котором они определены в документе html.   
Объекты формы имеют свойство name, значение которого равно значению параметра name тэга input, а также свойство value, значение которого определяется смыслом параметра value элемента формы.

**Объект frame**. В html тэг-контейнер <frameset> … </frameset> задает специальный тип окна, называемый набором фреймов.   
Это окно отображает несколько независимых (каждый со своими полосами прокрутки) фреймов на одном экране.   
Каждый фрейм, в свою очередь, может отображать определенный документ, расположенный по адресу, указанному в параметре src тэга <frame>.   
Набор фреймов образует страницу, поэтому не нужно задавать тэг body.

Свойства-массивы объектов.   
Некоторые объекты имеют свойства, которые являются массивами. Они используются для хранения информации о подчиненных объектах, когда их количество заранее не известно.

**20.Об'єкт EVENT та його використання**

Объект event. Каждое событие в JS порождает ассоциированный с ним объект **event**. Этот объект содержит всю информацию о событии и ее можно передать процедуре обработки события. Эта информация зависит от типа события.   
Например, event MouseDown содержит:

* информацию о типе события (свойство type)
* какая кнопка мыши была нажата (which)
* координаты курсора (screenx, screeny)

Совместно с обработчиками событий, объекты event позволяют производить достаточно тонкую обработку событий.   
Процедуру обработки события можно вызвать двумя способами:

1. Явно, назначив ссылку на процедуру обработки события, в соответствующем свойстве объекта.
2. Неявно - в параметре обработки события тэга соответствующего элемента.

Пример:

<form name="form1">

<input type="button" name="button1" value="Узнай событие">

<script>

document.form1.button1.onMouseDown=showEventType

</script>

</form>

function showEventType(e){

alert ("Произошло событие: "+e.type)

}

В объявлении функции **showEventType** присутствует параметр **e**, свойство **type** которого выводится в диалоговом окне.   
При явном вызове процедуры обработки события, объект **event** передается ей по умолчанию, поэтому в данном случае печатается значение свойства **type** объекта **event**, т.е. тип события **MouseDown**.   
При неявном вызове требуется задание обращения к процедуре в параметре **onMouseDown** тэга **input**, при этом необходимо явно указывать параметр **event**.

**21. Формування НТМL-документу в залежності від типу браузера**

При использовании одних и тех же HTML-тегов в старых браузерах и современных, придерживающихся стандартов (Mozilla и браузеры на его основе, Mac IE 5, Windows IE 6, Safari, Opera 7+) можно обнаружить два основных режима отображения web-страницы. В одном случае браузер пытается представить страницу в соответствии с рекомендациям CSS и (X)HTML. В другом случае отображение страницы происходит как в старых браузерах. В Mozilla эти два состояния известны как "the Standards mode" и "the Quirks mode", т.е. как стандартный режим и неопределенный соответственно.

**Quirks Mode - Неопределенное состояние**

При Quirks Mode в браузерах нарушаются рекомендации W3C для обеспечения нормального отображения страниц. Для разных браузеров существуют различные варианты совместимости с предыдущими версиями. Следовательно, "Quirks mode" не единственная цель. Например, Mozilla берет свое начало с Netscape 4.x, а Windows IE 6 с Windows IE 5.

Собственно цель Quirks mode - корректное отображение страниц, написанных для старых браузеров. При создании новых, современных страниц следует выполнять требования стандартов и рекомендаций ([CSS 2.1](http://www.w3.org/TR/CSS21/) в частности) и придерживаться Standards mode.

**Standards Mode - Соответствует стандартам**

При Standards Mode современные браузеры должны одинаково правильно представлять соответствующую стандартам страницу.

**Almost Standards Mode - Почти соответствует стандартам**

В браузерах Mozilla, Safari и Opera 7.5 еще существует третий вариант "the Almost Standards mode" (почти соответствующий стандартам), который не достаточно строго следует рекомендациям W3C. У Mac IE 5, Windows IE 6, Opera до 7.5 и Konqueror отсутствует Almost Standards mode, потому что они и так не могут работать в соответствии с CSS2, поэтому они будут находиться в этом случае в Standards modes. Фактически, их Standards modes это Almost Standards mode браузера Mozilla, а не полноценный Standards modes. Важно помнить: браузеры имеют одинаковые основные состояния, но даже при одинаковых состояния они все равно работают по-разному.

Старые браузеры кое-как еще обрабатывают нестандартные теги. В современных браузерах некорректный HTML приведет и к неправильной работе таблиц стилей и скриптов.

Раньше, применением различных трюков удавалось достичь более-менее схожего отображения страниц в разных браузерах, поэтому со временем, подавляющее большинство документов text/html в интернете оказались синтаксически неправильными, потому что они не следовали единому стандарту.

**Исследование Doctype**

Mozilla, Mac IE 5, Windows IE 6, Safari и Opera 7+ используют doctype для того, чтобы оценить как рассматривать разметку документа text/html. Этот метод основан на определении объявления типа документа (или его отсутствия) в начале HTML документа. (Документы, которые используют определение типа контента XML, всегда рассматриваются в Standards mode.)

Объявление типа документа (doctype) включает название (характерное для распознавания) корневого элемента и ссылку на DTD (document type definition - определение типа документа), которое в свою очередь, содержит информацию о синтаксических правилах для документа. (Теоретически, весь DTD может быть вставлен в объявление типа документа, но на практике используется лишь ссылка на него.) В спецификации HTML объявление типа документа содержит [информацию о версии](http://www.w3.org/TR/html4/struct/global.html#h-7.2) HTML. Отсутствие объявления типа документа и информации о его версии не позволяет правильно обрабатывать этот документ.

Ни спецификация HTML 4.01, ни ISO 8879 (SGML) не говорят что-либо об использовании объявления типа документа как о переключателе режимов отображения разметки. Пренебрежение к doctype основано на его несоблюдении, а также на том, что большинство устаревших нестандартных документов не включают объявление типа документа, либо ссылаются на старые DTD.

Типичное объявление типа документа содержит строку (все ее части разделены пробелами): "<!DOCTYPE", общий идентификатор корневого элемента ("HTML"), строку "PUBLIC", общий идентификатор DTD в кавычках, может быть системный идентификатор (URL) того же DTD и символ ">". Объявление типа документа должно размещаться перед начальным тегом корневого элемента.

**22. Доступ до властивостей елементів управління в HTML-документах**

Перечислим все элементы управления, описанные в стандарте HTML. Как мы знаем, большинство их создается одинарным тегом <INPUT> и отличается значением атрибута TYPE этого тега.

Область редактирования текста создается с помощью парного тега <TEXTAREA>. Внутри этого тега помещается содержимое области редактирования по умолчанию.

Список, обычный или раскрывающийся, создается с помощью парного тега <SELECT>. Внутри этого тега помещаются парные теги <OPTION>, создающие пункты этого списка. Название каждого пункта помещается внутрь соответ-ствующего ему тега <OPTION>.

В табл. 14.3 приведены все свойства, поддерживаемые элементами управле-ния. Элементы управления, поддерживающие эти свойства, указаны либо в виде значения атрибута TYPE тега <INPUT>, либо в виде создающего их тега.

Кроме того, элементы управления поддерживают все "обычные" свойства, присущие остальным элементам страницы. Это уже знакомые нам свойства id, style и др.

* **HTML-формы**. Формы используются для группирования вводимой пользователем информации и отправки ее обратно на сервер. Формы полностью доступны для сценариев и таким образом могут быть использованы Для обработки на клиентской стороне. В данном разделе представлено введение в формы HTML и элементы Input.
* **Программирование текстовых элементов Input**. Текстовые элементы Input создают текстовое окно для запроса информации от пользователя. В HTML определены четыре типа текстовых окон: текстовое окно с одной строкой, текстовое окно, содержащее множество строк, текстовое окно для ввода пароля и текстовое окно с именем файла. В данном разделе внимание сфокусировано на методах использования событий и объектной модели для проверки и форматирования вводимой пользователем информации.
* **Программирование элементов Select (список)**. Элементы Select используются для предоставления пользователю определенного набора вариантов. С помощью внутренних элементов управления могут быть созданы списки двух типов: *поле со списком* (list box) и *раскрывающееся поле со списком*(combo box). Для обоих типов списков используется одна модель программирования. В данном разделе рассмотрены методы программирования списков с помощью сценариев для динамического добавления и удаления элементов списков.
* **Программирование списков с помощью кнопок-переключателей и флажков**. Альтернативным способом предоставления пользователю возможности выбора из списка элементов является создание набора *флажков* (check boxes) или *кнопок-переключателей* (radio buttons). Флажки полезны для простых ответов да/нет. Кнопки-переключатели используются для выбора одного элемента из списка. В данном разделе обсуждаются преимущества использования списков с кнопками по сравнению с использованием окна со списком и объясняются методы написания сценариев.
* Программирование элементов командных кнопок. В HTML могут быть созданы четыре типа командных кнопок: *текстовые кнопки*(plain-text buttons), *кнопки в формате HTML* (rich HTML buttons), *кнопки отправки*(submit buttons) и *кнопки сброса* (reset buttons). Кнопки отправки на сервер и кнопки сброса имеют определенную модель поведения, когда пользователь находится в формах, и ведут себя как кнопки **Default** (по умолчанию) и **Cancel** (отмена). Кнопка **Default** отличается наличием дополнительной рамки и получает событие нажатия кнопки, когда пользователь нажимает клавишу <Enter>. Кнопка **Cancel** активизируется в том случае, если пользователь нажимает клавишу <Esc>. Модели поведения командных кнопок других типов должны быть определены с помощью сценария. В данном разделе показано, как использовать преимущества командных кнопок.
* **Программирование элементов Label и Fieldset**. *Надписи* (label) и *наборы полей* (fieldset) добавлены в HTML и необходимы для создания форм с форматированием. Элемент Label используется для определения отношений между элементом Input и содержанием. Элемент Fieldset используется для определения взаимоотношений между группами элементов управления.

**23. Об'єкти управління процесами браузера та їх використання в сценаріях**

16ый вопрос с похожей формулировкой

**24. Обробка подій у сценаріях**

**Включение сценариев в гипертекст**

Сценарий на JavaScript или VBScript является расширением HTML и встраивается в гипертекстовые страницы. Обычно определения процедур и функций сценария помещаются в контейнер заголовка HEAD, а код сценария - в тело документа BODY. При этом предложения должны быть внутри контейнера сценария SCRIPT. Кроме того, группа операторов сценария, заключенная в кавычки, может располагаться в определении обработчика событий элементов формы. Атрибут LANGUAGE задает язык сценария, а SRC определяет файл для включения.

В приведенном примере на JavaScript с помощью метода write() объекта document выводится приветствие на текущей странице.

<HTML>

<BODY>

<SCRIPTLANGUAGE="JavaScript">

<!- Пример простого сценария

document.write("HellofromJavaScript")

// ->

</SCRIPT>

</BODY>

</HTML>

**Обработка событий в сценариях**

В основном языки сценариев применяются для создания интерактивных документов. Возможно динамически формировать гипертекст с включением тегов HTML.

Язык является событийно-управляемым. Диалоговые элементы формы расширены и могут содержать указание на событие и код его обработки.

Основной синтаксис, используемый для указания обработчиков событий:

<TAGeventHandler="код\_сценария">

Здесь TAG это некоторый тег HTML, а eventHandler - имя обработчика событий. Как правило, обработчик событий имеет имя, совпадающее с именем события, с приставкой "on" (см. табл. событий в учебном пособии). В языке VBScript вместо двойных кавычек код сценария обрамляется апострофами.

Заметим, что многие объекты имеют методы, эмулирующие события, но эти методы не запускают обработчик события.

**25.Методи об’єктів HTML-документу**

1. Об'єкт window

Об'єкт *window*займає особливе, привілейоване положення в наборі об'єктів, оскільки він є основним контейнером, у якому розміщується все те, чим можна керувати за допомогою браузера.

У сценарії посилання на властивості і методи об'єкта *window*задається безпосереднім звертанням до об'єкта *window:*

window.властивість

Метод, за допомогою якого можна генерувати нові вікна, - *window.open().*У цьому методі використовується до трьох параметрів, що визначають такі характеристики вікна, як URL-адресу документа, що завантажується, його назву, що вказується в атрибутах *TARGET*HTML-тегів, а також фізичні параметри (розмір і вміст).

Наприклад, щоб закрити прямо зі сценарію головного вікна щойно створене нове вікно, можна скористатися методом *close():*

Властивість window.status

Рядок стану в нижній частині вікна браузера при наведенні на нього курсора миші зазвичай відображає URL-адресу посилання, хоча у ньому можуть відображаються інші повідомлення.

 Метод window.alert()

Цей метод ґенерує діалогове вікно-попередження, що відображає той текст, який передається методові як параметр. Єдина кнопка OK, напис якої не можна змінити, призначена для того, щоб користувач міг закрити вікно-попередження.

Змінювати можна тільки вміст повідомлень.

Метод window.confirm()

У діалоговому вікні методу *wmdowconfirmQ*використовується вже дві кнопки. Для більшості версій браузерів і платформ це кнопки *ОК*і *Cancel (Скасування).*Називається таке вікно діалоговим вікном підтвердження.

Метод window.prompt()

Цей метод ґенерує діалогове вікно запиту. У ньому друкується призначене розроблювачем сторінки повідомлення і виводиться текстове поле для введення відповіді.

Подія onLoad

Об'єкт *window*реагує на численні події, що генеруються системою або користувачем. Але з однією подією доводиться працювати частіше від інших. Вона виникає тоді, коли всі елементи сторінки успішно завантажені. Перевага використання події *onLoad*полягає у тому, що вона дозволяє використовувати функції і забезпечує наявність об'єктів документа в об'єктній моделі.

Властивості об'єкта window

*parent*- повертає батьківське по відношенню до поточного вікно; *self*- повертає посилання на поточне вікно; *top*- повертає посилання на головне вікно; *name*- назва вікна;

*returnValue*- визначає змінну, що була повернута   в результаті виконання події або діалогового вікна;

*document*- посилання "тільки для перегляду" на об'єкт вікна document;

*event*- посилання "тільки для перегляду" на глобальний об'єкт event;

*screen*- посилання "тільки для перегляду " на ґлобальний об'єкт screen.

Методи об'єкта window

Зазначимо, що поняття «вікно у фокусі» означає, що це вікно є активним;

*open*- відкриває нове вікно браузера; *close*- закриває поточне вікно браузера;

*navigate*- завантажує іншу сторінку з зазначеною адресою;

*setinterval*- вказує процедурі виконуватися періодично через задану кількість миллисекунд;

*setTimeout*- запускає програму через задану кількість мілісекунд після завантаження сторінки;

Події об'єкта window

*onblur*- вихід вікна з фокуса;

*onfocus*- вікно стає активним;

*onhelp*- натискання користувачем клавіш <F1>;

*onload*- сторінка цілком завантажена;

*onunload*- безпосередньо перед вивантаженням сторінки.

2.      Об'єкт document

В об'єкті документа *document*зберігається весь реальний вміст сторінки. Властивості і методи об'єкта *document,*в основному, впливають на зовнішній вигляд і вміст сторінки, завантаженої у вікні.

Розглянемо властивості цього об'єкта.

*acttveElement*- ідентифікує активний елемент;

*location*- повна URL-адреса документа;

*parentWtndow*- повертає батьківське вікно для документа;

*readyState*- визначає поточний стан об'єкта, що завантажується;

*referer*- URL-адреса сторінки, що викликала поточну;

*title*-       визначає   довідкову   інформацію   для   елемента , використовувану при завантаженні або в спливаючій підказці;

*vltnk*- колір відвіданих посилань на сторінці.

Колекції об'єкта document

*all*- колекція всіх тегів і елементів в основній частині документа;

*images*- колекція всіх графічних елементів на сторінці;

*links*- колекція всіх посилань і блоків <AREA> на сторінці;

*scripts*- колекція всіх розділів <SCRlPT> на сторінці;

*stylesheets*-   колекція   всіх   конкретних   властивостей стилю, визначених у документі.

Методи об'єкта document

*clear*- очищає виділену ділянку ;

*close*- закриває поточне вікно браузера;

*createEiement*- створює екземпляр елемента для виділеного тега;

*elementFromPoint*- повертає елемент із заданими координатами;

*writeln*- записує текст і код HTML, що закінчується поверненням каретки;

*open*- відкриває документ як потік для опрацювання результатів застосування методів *write*і *writeln.*

Події об'єкта document

*onafterupdate*- виникає при закінченні передавання даних;

*onbeforeupdate*- виникає перед вивантаженням сторінки;

*onerror*- помилка при передаванні;

*onkeyup*- виникає, коли користувач відпускає клавішу;

*onload*- виникає при повному завантаженні елемента;

*onmousedown*- відбувається, коли користувач натискає кнопку миші;

*omousemove*- відбувається, коли користувач переміщує мишу;

*onmouseout*-  відбувається,  коли покажчик миші  виходить з елемента;

Об'єкт history

Об'єкт *history*містить інформацію про адреси, які браузер відвідав під час поточного сеансу. Ми можемо пересуватися за цим списком за допомогою сценарію, завантажуючи сторінки, які він містить. Об'єкт *history*має одну властивість і три методи.

**26. Формування HTML-документів на підставі даних про об’єкти**

Для работы со структурой DOM в JavaScript предназначен объект document, который определен в глобальном объекте window. Объект document предоставляет ряд свойств и методов для управления элементами страницы.

Поиск элементов

Для поиска элементов на странице применяются следующие методы:

getElementById(value): выбирает элемент, у которого атрибут id равен value

getElementsByTagName(value): выбирает все элементы, у которых тег равен value

getElementsByClassName(value): выбирает все элементы, которые имеют класс value

querySelector(value): выбирает первый элемент, который соответствует css-селектору value

querySelectorAll(value): выбирает все элементы, которые соответствуют css-селектору value

Каждый отдельный узел, будь то html-элемент, его атрибут или текст, в структуре DOM представлен объектом Node. Этот объект предоставляет ряд свойств, с помощью которых мы можем получить информацию о данном узле:

childNodes: содержит коллекцию дочерних узлов

firstChild: возвращает первый дочерний узел текущего узла

lastChild: возвращает последний дочерний узел текущего узла

previousSibling: возвращает предыдущий элемент, который находится на одном уровне с текущим

nextSibling: возвращает следующий элемент, который находится на одном уровне с текущим

ownerDocument: возвращает корневой узел документа

parentNode: возвращает элемент, который содержит текущий узел

nodeName: возвращает имя узла

nodeType: возвращает тип узла в виде числа

nodeValue: возвращает или устанавливает значение узла в виде простого текста

Используя свойства nextSibling и previousSibling можно также пройтись по узлам в прямом или обратном порядке.

Для создания элементов объект document имеет следующие методы:

createElement(elementName): создает элемент html, тег которого передается в качестве параметра. Возвращает созданный элемент

createTextNode(text): создает и возвращает текстовый узел. В качестве параметра передается текст узла.

Иногда элементы бывают довольно сложными по составу, и гораздо проще их скопировать, чем с помощью отдельных вызовов создавать из содержимое. Для копирования уже имеющихся узлов у объекта Node можно использовать метод cloneNode()

Для удаления элемента вызывается метод removeChild() объекта Node. Этот метод удаляет один из дочерних узлов.

Для замены элемента применяется метод replaceChild(newNode, oldNode) объекта Node. Этот метод в качестве первого параметра принимает новый элемент, который заменяет старый элемент oldNode, передаваемый в качестве второго параметра.

Кроме методов и свойств объекта Node в JavaScript мы можем использовать свойства и методы объектов Element. Важно не путать эти два объекта: Node и Element. Node представляет все узлы веб-станицы, в то время как объект Element представляет непосредственно только html-элементы. То есть объекты Element - это фактически те же самые узлы - объекты Node, у которых тип узла (свойство nodeType) равно 1.

Одним из ключевых свойств объекта Element является свойство tagName, которое возвращает тег элемента.

Свойства innerText и innerHTML

Для получения или установки текстового содержимого элемента мы можем использовать свойство innerText, а для получения или установки кода html - свойство innerHTML

Методы объекта Element

Среди методов объекта Element можно отметить методы управления атрибутами:

getAttribute(attr): возвращает значение атрибута attr

setAttribute(attr, value): устанавливает для атрибута attr значение value. Если атрибута нет, то он добавляется

removeAttribute(attr): удаляет атрибут attr и его значение

Размеры и позиция элементов

Элементы имеют ряд свойств, которые позволяют определить размер элемента. Но важно понимать разницу между всеми этими свойствами. Свойства offsetWidth и offsetHeight определяют соответственно ширину и высоту элемента в пикселях. В ширину и высоту включается граница элемента. Свойства clientWidth и clientHeight также определяют ширину и высоту элемента в пикселях, но уже без учета границы

Для определения позиции элемента наиболее эффективным способом является метод getBoundingClientRect(). Этот метод возвращает объект со свойствами top, bottom, left, right, которые указывают на смещение элемента относительно верхнего левого угла браузера

**27. Основні конструкції мови JavaScript**

В javaScript условие осуществляет оператор if. В качестве условия может использоваться логическое выражение или переменная.

Тернарный оператор языка javascript - это сокращенная форма конструкции if ... else. Данный оператор включает в себя три операнда: ( условие ? Если условие ИСТИННО : Если условие ЛОЖНО )

Оператор switch javascript служит для проверки переменной на множество значений. Блок, начинающийся со служебного слова default можно опустить. Операторы блока будут выполнены в случае, если ни одно из перечисленных значений не подходит.

switch (выражение) {

case вариант1:

*//..блок операторов..*

break

case вариант2:

*//..блок операторов..*

break

[default:

*//..блок операторов..]*

}

Циклические операторы языка javaScript – For В качестве начального значения счетчика итераций используется выражение присваивания: например, i=0 - счетчик цикла начинается с нуля. В качестве приращения счетчика указывается шаг, с которым должен увеличиваться счетчик: например, i++ указывает на то, что каждая итерация цикла будет сопровождаться его увеличением на 1. Условие цикла - это и есть конечное значение счетчика: например, i<10 - счетчик, достигнув значения 10, останавливает цикл.

for (var i=0;i<10;i++)

{

document.write(i+"<br>");

}

Оператор break прерывает выполнение всего тела цикла, т.е. осуществляет выход из цикла в javaScript. В то время как оператор continue прерывает выполнение текущей итерации цикла, но продолжая при этом выполнение цикла со следующий итерации.

for (var i=0;i<10;i++)

{

if (i==4) continue;

document.write(i+"<br>");

if (i==8) break; }

Оператор Exit. В языке javasctipt предусмотрен оператор выхода из программного кода - оператор exit. Чаще всего оператор используется для исключения ошибки ввода пользователя.

Цикл while имеет вид:

while (условие) {

// код, тело цикла

}

Пока условие верно – выполняется код из тела цикла. Проверку условия можно поставить под телом цикла, используя специальный синтаксис do..while:

do {

// тело цикла

} while (условие);

Цикл, описанный, таким образом, сначала выполняет тело, а затем проверяет условие. Циклические операторы языка javaScript - цикл For In Цикл for in предназначен для прохода по массивам, коллекциям и объектам

Оператор обработки исключений в javaScript - try..catch В некоторых случаях код на странице не работает непонятно по какой причине. Где искать ошибку? В таких случаях можно применить оператор try..catch, который пытается выполнить фрагмент кода, и, если в коде есть ошибка, то существует возможность выдать ошибку на экран.

try

{

sdfsdf; *// оператор с ошибкой*

}

catch(e)

{

alert("обработка ошибки: "+e.message);

}

finally

{

alert("завершающие действия");

}

**28. Особливості функцій в сценаріях JavaScript**

Функции представляют собой набор инструкций, выполняющих определенное действие или вычисляющих определенное значение.

function имя\_функции([параметр [, ...]]){

    // Инструкции

}

Определение функции начинается с ключевого слова function, после которого следует имя функции. Наименование функции подчиняется тем же правилам, что и наименование переменной: оно может содержать только цифры, буквы, символы подчеркивания и доллара ($) и должно начинаться с буквы, символа подчеркивания или доллара.

После имени функции в скобках идет перечисление параметров. Даже если параметров у функции нет, то просто идут пустые скобки. Затем в фигурных скобках идет тело функции, содержащее набор инструкций.

Необязательно давать функциям определенное имя. Можно использовать анонимные функции var display = function(){ } // определение функции. Функция может содержать локальные переменные, объявленные через var. Такие переменные видны только внутри функции.

Блоки if/else, switch, for, while, do..while не влияют на область видимости переменных.

При объявлении переменной в таких блоках, она всё равно будет видна во всей функции.

Неважно, где именно в функции и сколько раз объявляется переменная. Любое объявление срабатывает один раз и распространяется на всю функцию. Функция может обратиться ко внешней переменной. Значение возвращается оператором return .Вызов return тут же прекращает функцию.

Если return; вызван без значения, или функция завершилась без return, то её результат равен undefined

«Классическое» объявление функции, о котором мы говорили до этого, вида function имя(параметры) {...}, называется в спецификации языка «Function Declaration».

*Function Declaration* – функция, объявленная в основном потоке кода.

*Function Expression* – объявление функции в контексте какого-либо выражения, например присваивания.

**29 Таблиці стилів та їх використання**

Каскадні таблиці стилів (англійською Cascading Style Sheets, CSS) — спеціальна мова, яку використовують для запису оформлення сторінок, написаних мовами розмітки даних.

Таблиці стилів дають змогу спростити процес створення сторінок і поліпшити їх зовнішній вигляд. Концепція стилів подібна до ідеї стилів, яку реалізовано в сучасних текстових редакторах — текст спочатку вводять, а потім форматують, користуючись стилями. Застосування стилів дає змогу вводити на сторінку потрібні тексти та інші елементи, не задумуючись над їхнім зовнішнім виглядом і розташуванням.

Таблиці стилів зазвичай створють окремо від html-файлу. Під час створення html-файлу увагу концентрують на змісті сторінки, а не на її зовнішньому вигляді, а під час створення таблиці стилів — навпаки. Отже, стилі дають змогу розділити етапи створення html-файлу й удосконалення зовнішнього вигляду сторінки.

Cпособи взаємодії таблиці стилів з html-файлом:

* зв'язування;
* імпортування;
* вбудовування (використання стильових класів).

Зв'язування

Таблицю стилів створюють і зберігають в окремому файлі з розширенням css. Таку таблицю називають зовнішньою. Щоб зв'язати основний файл з такою таблицею стилів, у середині тега <head> застосовують одинарний тег <link> з інформацією про таблицю.

<head>

<link href="адреса таблиці стилів"

type="text/css"

rel="stylesheet">

</head>

Тут:

href="адреса таблиці стилів" — адреса розташування файлу,

type="text/css" — опис типу файлу,

rel="stylesheet" — вказівка на відношення основного файлу з файлом таблиці стилю.

Імпорт

Це те саме, що і зв'язування, але взаємодію файлів забезпечують засобами тегу <style> і такої вказівки:

@import URL(адреса таблиці стилів)

Вбудовування (використання стильових класів)

Якщо таблицю створено лише для деякого конкретного html-файлу, то її розташовують у цьому файлі в контейнері <style>. Таку таблицю стилів називають внутрішньою або вбудованою. Також стиль можна вбудувати безпосередньо в html-тег за допомогою атрибуту <style>. Наприклад, контейнер

<span style="color: blue">Blue </span>

створює блок з текстом синього кольору: Blue.

Таблиця стилів складаєтся з правил, а правило — з назви тега (селектора) чи списку назв тегів і описів стилів, які діятимуть у межах деякого html-файлу.

Опис стилю — це послідовність пар властивість: значення, які записують через крапку з комою у фігурних дужках.

Загальний вигляд правила такий:

Селектор{властивість1: значення1;

властивість2: значення2; …

властивістьN: значенняN;}

Назва властивості складається з одного чи декількох англійських слів, які записують через риску. Властивість діє лише у межах тега, зазначеного у відповідному правилі.

Правила можуть починатися або з назви тега без кутових дужок, або з деякого слова користувача, перед яким є крапка. Це слово стає назвою типу, який можна застосувати в контейнері body до іншого текстового блока, фрагменту тексту чи деякого елемента.

Внутрішні таблиці стилів описують у головному файлі в контейнері head за допомогою контейнера style.

Виокремлення текстового блоку, до якого буде застосовано стиль, здійснюють за допомогою контейнера

<div class=назва стилю>…</div>

де значення параметра class задає конкретний стиль.

Виокремлення у блоці фрагмента, для якого буде переозначено стиль, здійснюють за допомогою контейнера

<span>…</span>

**30 Модель DOM та властивості елементів управління в HTML-документах**

Основным инструментом работы и динамических изменений на странице является DOM (Document Object Model) – объектная модель, используемая для XML/HTML-документов.

Согласно DOM-модели, документ является иерархией, деревом. Каждый HTML-тег образует узел дерева с типом «элемент». Вложенные в него теги становятся дочерними узлами. Для представления текста создаются узлы с типом «текст».

DOM – это представление документа в виде дерева объектов, доступное для изменения через JavaScript.

DOM позволяет делать что угодно с HTML-элементом и его содержимым, но для этого нужно сначала нужный элемент получить.

Доступ к DOM начинается с объекта document. Из него можно добраться до любых узлов.

Формы и внутренние элементы управления HTML:

1. HTML-формы. Формы используются для группирования вводимой пользователем информации и отправки ее обратно на сервер. Формы полностью доступны для сценариев и таким образом могут быть использованы Для обработки на клиентской стороне.
2. Текстовые элементы Input создают текстовое окно для запроса информации от пользователя. В HTML определены четыре типа текстовых окон: текстовое окно с одной строкой, текстовое окно, содержащее множество строк, текстовое окно для ввода пароля и текстовое окно с именем файла
3. Элементы Select используются для предоставления пользователю определенного набора вариантов. С помощью внутренних элементов управления могут быть созданы списки двух типов: поле со списком (list box) и раскрывающееся поле со списком (combo box). Для обоих типов списков используется одна модель программирования.
4. Альтернативным способом предоставления пользователю возможности выбора из списка элементов является создание набора флажков (check boxes) или кнопок-переключателей (radio buttons). Флажки полезны для простых ответов да/нет. Кнопки-переключатели используются для выбора одного элемента из списка.
5. Командные конопки. В HTML могут быть созданы четыре типа командных кнопок: текстовые кнопки (plain-text buttons), кнопки в формате HTML (rich HTML buttons), кнопки отправки (submit buttons) и кнопки сброса (reset buttons). Кнопки отправки на сервер и кнопки сброса имеют определенную модель поведения, когда пользователь находится в формах, и ведут себя как кнопки Default (по умолчанию) и Cancel (отмена). Кнопка Default отличается наличием дополнительной рамки и получает событие нажатия кнопки, когда пользователь нажимает клавишу <Enter>. Кнопка Cancel активизируется в том случае, если пользователь нажимает клавишу <Esc>. Модели поведения командных кнопок других типов должны быть определены с помощью сценария.
6. Label и Fieldset. Надписи (label) и наборы полей (fieldset) добавлены в HTML и необходимы для создания форм с форматированием. Элемент Label используется для определения отношений между элементом Input и содержанием. Элемент Fieldset используется для определения взаимоотношений между группами элементов управления.

**31 Псевдокласи та їх використання**

Псевдоклассы определяют динамическое состояние элементов, которое изменяется с помощью действий пользователя, а также положение в дереве документа. Примером такого состояния служит текстовая ссылка, которая меняет свой цвет при наведении на неё курсора мыши.

Псевдоклассы дают возможность стилизовать элемент на основе не только отношений в DOM-дереве, но и основываясь на внешних факторах, таких как история посещений (например, :visited), состояние содержимого (вроде :checked у некоторых элементов формы) или позиции курсора мыши (например, :hover определяет, находится ли курсор мыши над элементом).

|  |  |
| --- | --- |
| **Псевдокласс** | **Описание** |
| :active | Определяет стиль активного элемента. |
| :blank | Представляет пустые элементы, т. е. те, которые не содержат дочерних элементов или текста. |
| :checked | Применяется к элементам интерфейса, таким как переключатели (checkbox) и флажки (radio), когда они находятся в положение «включено». |
| :default | Применяет стиль к элементам форм, которые установлены по умолчанию в группе похожих элементов. |
| :disabled | Применяет стиль к заблокированным элементам форм. |
| :empty | Представляет пустые элементы, т. е. те, которые не содержат дочерних элементов, текста, пробелов или переноса строк. |
| :enabled | Используется для применения стиля к доступным (не заблокированным) элементам форм. |
| :first-child | Задаёт стилевое оформление для первого дочернего элемента в группе братских элементов. |
| :first-of-type | Задаёт правила стилей для первого элемента определённого типа в группе братских элементов. |
| :focus | Определяет стиль для элемента, получающего фокус. |
| :focus-within | Определяет стиль элемента, когда он сам или элементы внутри него получают фокус. |
| :fullscreen | Псевдокласс :fullscreen применяется к элементам, когда браузер находится в полноэкранном режиме. |
| :hover | Определяет стиль элемента при наведении на него курсора мыши, но при этом элемент ещё не активирован. |
| :in-range | Применяется к элементам форм, у которых введённое пользователем значение находится в заранее заданном диапазоне. Сам диапазон устанавливается с помощью атрибутов min и max, они, соответственно, задают минимальное и максимальное значение. |
| :indeterminate | Задаёт стиль для элементов форм, таким как флажки и переключатели, когда они находятся в неопределённом состоянии. |
| :invalid | Применяется к полям формы, содержимое которых не соответствует указанному типу. |
| :lang | Определяет язык, который используется в документе или его фрагменте. |
| :last-child | Задаёт стилевое оформление последнего дочернего элемента в группе братских элементов. |
| :last-of-type | Задаёт правила стилей для последнего элемента определённого типа в группе братских элементов. |
| :link | Применяется к ссылкам, которые ещё не посещались пользователем. |
| :not | Задаёт правила стилей для элементов, которые не содержат указанный селектор. |
| :nth-child | Используется для добавления стиля к элементам на основе нумерации в дереве элементов. |
| :nth-last-child | Используется для добавления стиля к элементам на основе нумерации в дереве элементов, отсчёт ведётся с конца. |
| :nth-last-of-type | Используется для добавления стиля к элементам указанного типа на основе нумерации в дереве элементов, отсчёт ведётся от последнего элемента. |
| :nth-of-type | Используется для добавления стиля к элементам указанного типа на основе нумерации в дереве элементов. |
| :only-child | Применяется к дочернему элементу, только если он является единственным у своего родителя. |
| :only-of-type | Применяется к дочернему элементу указанного типа, только если он единственный у родителя. |
| :optional | Применяет стилевые правила к полю формы, у которого не задан атрибут required. |
| :out-of-range | Применяется к полям форм, у которых введённое пользователем значение выходит из заданного диапазона. Псевдокласс работает только для тех полей, где пользователь может сам ввести значение, несмотря на ограничения. |
| :placeholder-shown | Определяет стиль элемента &lt;input&gt; или &lt;textarea&gt;, который в данный момент отображает текст подсказки, заданной атрибутом placeholder. |
| :read-only | Применяется к полям формы, у которых задан атрибут readonly. |
| :read-write | Применяется к полям формы, доступных для изменения. |
| :required | Применяет стилевые правила к элементу input, у которого установлен атрибут required. |
| :target | Применяется к целевому элементу, иными словами, к идентификатору, который указан в адресной строке браузера. |
| :valid | Применяется к полям формы, содержимое которых проходит проверку в браузере на соответствие указанному типу. |
| :visited | Применяется к ссылкам, уже посещённым пользователем, и задаёт для них стилевое оформление. |

**32 Управління процесами браузера в сценаріях**

???

В браузере код JavaScript запускается на выполнение сразу после обработки разметки страницы. Код скачивается (если это необходимо) и запускается еще до отображения содержимого веб-страницы. Это важно, так как ваш скрипт может отвечать за выполнение различных функций: загружать последующий код, удалять каждый DOM-элемент, перенаправлять посетителя на другой URL и т. д. Но это блокирует дальнейшую загрузку страницы, и она перестает отвечать на запросы.

Часто требуется, чтобы JavaScript-код был запущен как можно скорее, так как он инициализирует виджеты и контроллеры событий. Но есть множество менее важных фоновых задач, остановка которых никак не сказываются на взаимодействии ресурса с пользователем:

* Считывание данных аналитики;
* Отправка данных в социальные сети;
* Предварительное извлечение контента, находящегося за пределами окна просмотра;
* Предварительная обработка и вывод HTML-элементов.

Эти задачи влияют на скорость загрузки несущественно. Но для того, чтобы страница сохраняла свои адаптивные свойства, не стоит запускать эти задачи во время взаимодействия пользователя с контентом или прокручивания им страницы.

Решение этой проблемы очень простое – использовать Web Workers – объекты, которые способны обрабатывать код отдельным потоком. Они отлично подходят для предварительного извлечения и обработки, но с их помощью вы не сможете напрямую осуществлять доступ или обновлять DOM.

Есть еще одна опция – использовать setTimeout (или setTimeout(doSomething, 1). Браузер выполнит функцию doSomething() лишь после того, как будут завершены другие, более важные задачи. В результате запуск функции получает самое последнее место в очереди. При этом следует учитывать, что функция будет запущена в любом случае.

**37. Селекторы**

**Селекторы тегов**

В качестве селектора может выступать любой элемент HTML, для которого определяются правила форматирования, такие как: цвет, фон, размер и др.

E { Описание правил стиля }

Здесь E имя произвольного элемента. Следует понимать, что хотя правила стиля можно применить к любому элементу, результат будет заметен только для тегов, которые непосредственно отображаются в контейнере <body>.

**p** {color: green; /\* Зелёный цвет текста \*/}

**Идентификаторы**

E#Имя\_идентификатора { Описание правил стиля }  
#Имя\_идентификатора { Описание правил стиля }

Е должно содержать символы EN алфавита, - или \_

Обращение к идентификатору происходит с помощью атрибута id, значением которого выступает имя идентификатора (id="Имя\_идентификатора"). Символ решётки при этом уже не указывается.

#help { color: green; }

**Классы**

Классы применяют, когда необходимо определить стиль для индивидуального элемента веб-страницы или задать разные стили для одного тега.

E.Имя\_класса { Описание правил стиля }  
.Имя\_класса { Описание правил стиля }

p.cite { color: green; } или .cite{ color: green; }

**Мультиклассы**

К любому тегу одновременно можно добавить несколько классов, перечисляя их в атрибуте class через пробел. В этом случае к элементу применяется стиль, описанный в правилах для каждого класса.

<E class="class1 class2 ...">

.class1.class2 { Описание правил стиля }

**Универсальный селектор**

\* { Описание правил стиля }

**Вложенные селекторы**

E F { Описание правил стиля }

Здесь E это родительский тег, а F — дочерний тег, расположенный в контейнере <E>. В этом случае стиль будет применяться к тегу <F>, когда соблюдается следующий код <E><F></F></E>. Не обязательно должно быть два тега, допускается произвольный уровень вложения. При этом конструкция может записываться так: div div ul li {...}.

**Дочерние селекторы**

E > F { Описание правил стиля }

Стиль применяется к элементу F, только когда он является дочерним для элемента E.

**Соседние селекторы**

Соседними называются элементы веб-страницы, когда они следуют непосредственно друг за другом в коде документа.

E + F { Описание правил стиля }

**Родственные селекторы**

E ~ F { Описание правил стиля }

Для управления стилем родственных элементов используется символ тильды (~), он добавляется между двумя селекторами E и F. Пробелы вокруг тильды не обязательны. Стиль при такой записи применяется к элементу F в том случае, если он имеет того же родителя, что и элемент E и следует сразу после него.

**[атрибут$="значение"]**

Устанавливает стиль для элемента в том случае, если значение атрибута оканчивается указанным текстом.

[атрибут$="значение"] { Описание правил стиля }   
E[атрибут$="значение"] { Описание правил стиля }

**[атрибут\*="значение"]**

Значение атрибута содержит указанный текст.

[атрибут\*="значение"] { Описание правил стиля }   
E[атрибут\*="значение"] { Описание правил стиля }

**[атрибут="значение"]**

Устанавливает стиль для элемента в том случае, если задано определённое значение специфичного атрибута.

[атрибут="значение"] { Описание правил стиля }   
E[атрибут="значение"] { Описание правил стиля }

**[атрибут^="значение"]**

Устанавливает стиль для элемента в том случае, если значение атрибута тега начинается с указанного текста.

[атрибут^="значение"] { Описание правил стиля }   
E[атрибут^="значение"] { Описание правил стиля }

**[атрибут|="значение"]**

В именах идентификаторов и классов разрешено использовать символ дефиса (-), что позволяет создавать значащие значения атрибутов id и class.

[атрибут|="значение"] { Описание правил стиля }   
E[атрибут|="значение"] { Описание правил стиля }

**[атрибут~="значение"]**

Чтобы задать стиль при наличии в списке требуемого значения требуется использовать конструкцию ~=.

[атрибут~="значение"] { Описание правил стиля }   
E[атрибут~="значение"] { Описание правил стиля }

**Селекторы атрибутов**

Устанавливает стиль для элемента, если задан специфичный атрибут тега. Его значение в данном случае не важно.

[атрибут] { Описание правил стиля }  
E[атрибут] { Описание правил стиля }

**38.фавита, - или \_ символы Об’єкт Window, властивості та методи**

Объект Window представляет собой открытое окно браузера.

Данный объект является корневым и все свойства, методы, функции и даже переменные являются его свойствами и методами.

**Свойства объекта Window**

|  |  |
| --- | --- |
| [closed](http://www.wisdomweb.ru/JSd/wiclosed.php) | Возвращает логическое значение (true или false) в зависимости от того,  закрыто указанное окно или отрыто. |
| [frames](http://www.wisdomweb.ru/JSd/wiframes.php) | Возвращает массив всех фреймов на странице (включая iframe). |
| [document](http://www.wisdomweb.ru/HDOMd/document.php) | Возвращает объект Document данного окна. |
| [history](http://www.wisdomweb.ru/JSd/history.php) | Возвращает объект History данного окна. |
| [length](http://www.wisdomweb.ru/JSd/wilength.php) | Возвращает количество фреймов (включая iframe), которые находятся в данном окне. |
| [location](http://www.wisdomweb.ru/JSd/location.php) | Возвращает объект Location данного окна. |
| [name](http://www.wisdomweb.ru/JSd/winame.php) | Устанавливает или возвращает имя данного окна. |
| [navigator](http://www.wisdomweb.ru/JSd/navigator.php) | Возвращает объект Navigator данного окна. |
| [opener](http://www.wisdomweb.ru/JSd/wiopener.php) | Возвращает ссылку на окно, которое открыло данное. |
| [parent](http://www.wisdomweb.ru/JSd/wiparent.php) | Возвращает родительское окно данного окна. |
| [screen](http://www.wisdomweb.ru/JSd/screen.php) | Возвращает объект Screen данного окна. |
| [self](http://www.wisdomweb.ru/JSd/wiself.php) | Возвращает текущее окно. |
| [top](http://www.wisdomweb.ru/JSd/witop.php) | Возвращает верхнее браузерное окно для данного окна. |

**Методы объекта Window**

|  |  |
| --- | --- |
| [alert()](http://www.wisdomweb.ru/JSd/wialert.php) | Вызывает окно оповещения, которое содержит текст сообщения и  клавишу ОК. |
| [blur()](http://www.wisdomweb.ru/JSd/wiblur.php) | Делает окно неактивным. |
| [clearInterval()](http://www.wisdomweb.ru/JSd/wiclinterval.php) | Прекращает повторное выполнение кода заданного setInterval(). |
| [clearTimeout()](http://www.wisdomweb.ru/JSd/wicltimeout.php) | Отменяет запланированное методом setTimeout() выполнение кода. |
| [close()](http://www.wisdomweb.ru/JSd/wiclose.php) | Закрывает окно. |
| [confirm()](http://www.wisdomweb.ru/JSd/wiconf.php) | Вызывает окно подтверждения содержащее текст сообщения и клавиши ОК и Отмена. |
| [focus()](http://www.wisdomweb.ru/JSd/wifocus.php) | Делает окно активным. |
| [moveBy()](http://www.wisdomweb.ru/JSd/wimoveby.php) | Смещает окно относительно его текущей позиции. |
| [moveTo()](http://www.wisdomweb.ru/JSd/wimoveto.php) | Перемещает окно на указанную позицию. |
| [open()](http://www.wisdomweb.ru/JSd/wiopen.php) | Открывает новое окно браузера. |
| [print()](http://www.wisdomweb.ru/JSd/wiprint.php) | Распечатывает содержимое текущего окна. |
| [prompt()](http://www.wisdomweb.ru/JSd/wiprompt.php) | Вызывает окно запроса, побуждающее посетителя ввести в него определенные данные. |
| [scrollBy()](http://www.wisdomweb.ru/JSd/wiscrollby.php) | Прокручивает содержимое окна на указанное количество пикселей. |
| [scrollTo()](http://www.wisdomweb.ru/JSd/wiscrollto.php) | Прокручивает содержимое окна до указанных координат. |
| [setInterval()](http://www.wisdomweb.ru/JSd/wisetint.php) | Вызывает функцию или выполняет код через определенные промежутки времени (указанные в миллисекундах). |
| [setTimeout()](http://www.wisdomweb.ru/JSd/wisetimeout.php) | Вызывает функцию или выполняет код после указанного количества миллисекунд один раз. |

**39. Об’єкт Screen, властивості та методи**

**Свойства**

[Screen.availTop](https://developer.mozilla.org/ru/docs/Web/API/Screen/availTop)

Возвращает координату по оси y первого пикселя по оси y, но не относящегося к пользовательским элементам интерфейса.

[Screen.availLeft](https://developer.mozilla.org/ru/docs/Web/API/Screen/availLeft)

Возвращает первый доступный пиксель от левой стороны экрана.

[Screen.availHeight](https://developer.mozilla.org/ru/docs/Web/API/Screen/availHeight)

Возвращает высоту экрана в пикселях минус высота пользовательских элементов интерфейса, таких как панель задач в Windows.

[Screen.availWidth](https://developer.mozilla.org/ru/docs/Web/API/Screen/availWidth)

Возвращает ширину экрана в пикселях.

[Screen.colorDepth](https://developer.mozilla.org/ru/docs/Web/API/Screen/colorDepth)

Возвращает глубину цвета экрана.

[Screen.height](https://developer.mozilla.org/ru/docs/Web/API/Screen/height)

Возвращает высоту экрана в пикселях.

[Screen.left](https://developer.mozilla.org/ru/docs/Web/API/Screen/left)

Возвращает расстояние в пикселях от левой стороны основного экрана до левой стороны текущего экрана.

[Screen.orientation](https://developer.mozilla.org/ru/docs/Web/API/Screen/orientation)

Возвращает текущую ориентацию экрана.

[Screen.pixelDepth](https://developer.mozilla.org/ru/docs/Web/API/Screen/pixelDepth)

Возвращает количество битов на пиксель экрана.

[Screen.top](https://developer.mozilla.org/ru/docs/Web/API/Screen/top)

Возвращает расстояние в пикселях от верхней стороны экрана до текущего экрана.

[Screen.width](https://developer.mozilla.org/ru/docs/Web/API/Screen/width)

Возвращает ширину экрана в пикселях.

[Screen.mozEnabled](https://developer.mozilla.org/ru/docs/Web/API/Screen/mozEnabled)

Булево. Установка на false выключит экран устройства.

[Screen.mozBrightness](https://developer.mozilla.org/ru/docs/Web/API/Screen/mozBrightness)

Управляет яркостью экрана устройства. Ожидается значение между 0 и 1.0.

Методы

[Screen.lockOrientation](https://developer.mozilla.org/ru/docs/Web/API/Screen/lockOrientation)

Блокировка ориентации экрана (работает только в полноэкранном режиме или для установленных приложений)

[Screen.unlockOrientation](https://developer.mozilla.org/ru/docs/Web/API/Screen/unlockOrientation)

Разблокировка ориентации экрана (работает только в полноэкранном режиме или для установленных приложений)

**40. Об’єкт Location, властивості та методи**

Метод toString этого объекта возвращает URL, а различные свойства позволяют получить/установить отдельные компоненты адреса.

Для некоторых строковых операций необходимо явно преобразовать Location к строке:

|  |
| --- |
| window.location.toString().charAt(17) |

**Свойства объекта Location**

Все следующие свойства являются строками.  
Колонка "Пример" содержит их значения для URL:

http://www.google.com:80/search?q=javascript#test

|  |  |  |
| --- | --- | --- |
| Свойство | Описание | Пример |
| hash | часть URL, которая идет после символа решетки '#', включая символ '#' | #test |
| host | хост и порт | www.google.com:80 |
| href | весь URL | http://www.google.com:80/search?q=javascript#test |
| hostname | хост (без порта) | www.google.com |
| pathname | строка пути (относительно хоста) | /search |
| port | номер порта | 80 |
| protocol | протокол | http: |
| search | часть адреса после символа ?, включая символ ? | ?q=javascript |

В Firefox есть баг: если hash-компонент адреса содержит закодированные (см. [encodeURIComponent](https://javascript.ru/encodeURIComponent)) символы, свойство hash возвращает раскодированный компонент. Например, вместо %20 будет пробел и т.п. Другие браузеры ведут себя корректно и не раскодируют hash.

**Методы объекта Location**

assign(url)

загрузить документ по данному url

reload([forceget])

перезагрузить документ по текущему URL. Аргумент forceget - булево значение, если оно true, то документ перезагружается всегда с сервера, если false или не указано, то браузер может взять страницу из своего кэша.

replace(url)

заменить текущий документ на документ по указанному url. Разница, по сравнению с assign() заключается в том, что после использования replace()страница не записывается в истории посещений. В частности, это значит, что посетитель не сможет использовать для возврата кнопку браузера "Назад".

toString()

возвращает строковое представление URL для объекта Location

**41. Заміна текстових елементів HTML-документу при його обробці браузером.**

Рассмотрим последовательность работы браузера при отображении документа:

1. Из полученного от сервера HTML-документа формируется DOM (Document Object Model).
2. Загружаются и распознаются стили, формируется CSSOM (CSS Object Model).
3. На основе DOM и CSSOM формируется дерево рендеринга ( render tree — набор объектов). Render tree дублирует структуру DOM, но сюда не попадают невидимые элементы (например — <head>, или элементы со стилем display:none;). Также, каждая строка текста представлена в дереве рендеринга как отдельный renderer. Каждый объект рендеринга содержит соответствующий ему объект DOM (или блок текста), и рассчитанный для этого объекта стиль. *Проще говоря, render tree описывает визуальное представление DOM.*
4. Для каждого элемента render tree рассчитывается положение на странице — происходит layout. Браузеры используют поточный метод (flow), при котором в большинстве случаев достаточно одного прохода для размещения всех элементов (для таблиц проходов требуется больше).
5. Наконец, происходит отрисовка в браузере.

В процессе взаимодействия пользователя со страницей, а также выполнения скриптов, она меняется, что требует повторного выполнения некоторых из вышеперечисленных операций.

**42. Особливості HTML5.**

**HTML** (*HyperText Markup Language*) – наиболее популярный инструмент разметки, который уже долгое время не теряет свою актуальность среди разработчиков. HTML5 –  новая версия HTML стандарта и языка разметки, которая выпущена для улучшение языка, для работы с новыми мультимедийными элементами и приложениями. HTML5 обеспечивает удобную читаемость программ для человека и обработки для браузеров. Имеет поддержку части тэгов HTML4 и языка JavaScript.

***Основные преимущества  HTML5:***

* применения более простого вида кода, например, div заменены более совершенными элементами.
* дизайнерские решение, которые позволяют сделать индивидуальное решения для сайта. Также улучшен пользовательский интерфейс, он стал более понятным и функциональным. Валидацию форм можно осуществлять с помощью классических средств HTML-разметки.
* можно использовать новые поля ввода для разных целей, например, поиска.
* новая более совершенная семантика HTML5 дает возможность быстрее и проще различать футер, заголовки, панель навигации. Для этого используются разработанные по новому теги для упрощенного назначения основных элементов в разметке.
* удобное юзабилити. С HTML5 можно использовать новые элементы, в том числе <canvas>, <audio>, <video>. Сделаны улучшения для интеграции с SVG. Теперь можно не использовать плагины, к примеру, Flash.
* установку музыки, чартов, видео и рисунков можно производить без использования сторонних программ.
* улучшена структура документов, для этого используют новые теги: <header>, <section> и <article>.
* восприятие разметки страницы:  исходный код стал простым и понятным.
* разработаны доски общений, вики и drag-n-drop, для того чтобы улучшить клиентскую часть приложения..
* значительно увеличена скорость отклика страницы (используются хранилища localStorage и sessionStorage, которые частично заменили куки).
* используется Geolocation API, а полученные данные могут применяться в приложениях.
* HTML5 имеет поддержку  MathML (математическая разметка для формул) и SVG (форма  изображений, масштабируемая векторная графика).
* HTML5 поддерживается популярными и широко используемыми мобильными устройствами
* HTML5 помогает поисковикам найти ценную информацию на сайте, например: <articlе>  показывает поисковикам ценную часть контента на сайте, более важную чем то, что указано в <footer>. Эти теги позволяют легко найти важные части контента на страницах ресурса.

Также среди нововведений HTML5 стоит выделить следующие элементы: **Figure** – рисунок; **Header**– заголовок страницы;**Section** – крупный блок страницы; **Footer** – нижняя часть страницы; **meta charset = «UTF-8»** – обновление кодировки страницы; **Nav** – навигация по сайту;**Aside** – дополнительный контент в виде боковой колонки; **Article** – статья, основная часть контента.

**43. Особливості форм в HTML5.**

С появлением HTML5 формы получили новые типы полей ввода и новые атрибуты для полей введения форм. Далее кратко описано, что из себя представляют тринадцать новых типов полей.

Color

<input id="color" name="color" type="color"/>

Этот тип поля очень прост для понимания: он обеспечивает выбор цвета для пользователя, и генерирует шестнадцатеричное значение. Палитра цветов будет нативной для каждой операционной системы или браузера, как и большинство новых полей.

Date

<input id="date" name="date" type="date"/>

Поле Date позволит вам выбрать дату из раскрывающегося списка.

Date-time

<input id="departure-day-time" name="departure-day-time" type="date-time"/>

Если вы хотите, чтобы пользователь выбрал одновременно не только дату, но и время.

Datetime-local

<input id="arrival-time" name="arrival-time" type="datetime-local"/>

Этот тип поля точно такой же как "date-time", но он предоставляет вам время в локальной временной зоне.

Email

<input type="email" name="email" id="email"/>

Этот тип поля создан специально для ввода адреса электронной почты, теперь нет необходимости создавать регулярное выражение на JS или PHP для проверки вводимой строки. На смартфонах оно будет вызывать специальную клавиатуру для удобного ввода адреса.

Month

<input id="expiry" name="expiry" type="month"/>

Это поле ввода позволит вам выбрать месяц года.

Number

<input type="number" [min](http://www.php.net/min)="48" [max](http://www.php.net/max)="84" step="0.5" value="48" name="height-inches"/>

Поле Number, как вы могли догадаться, предназначено только для ввода чисел. Вы можете указать разрешенный диапозон для чисел, если вы укажете атрибуты min и max как в приведенном выше примере, а также указать шаг в этом диапазоне.

Range

<input type="range" [min](http://www.php.net/min)="1" [max](http://www.php.net/max)="100" value="0" id="happiness"/>

В поле Range браузер выводит ползунок, который пользователь может перетаскивать влево-вправо. Можно указать минимальное и максимальное значение, а также значение по умолчанию.

Tel

<input type="tel" name="tel" id="tel"/>

Этот тип поля очень удобно использовать в смартфонах, так как оно вызывает цифровую клавиатуру телефона, чтобы можно было ввести номер телефона максимально просто. Оно, однако, не проверяет правильность ввода, в отличие от Email или URL.

Time

<input name="start-time" id="start-time" type="time"/>

Этот тип ввода позволит пользователям выбирать только определенное время, например, часы и минуты.

Url

<input type="url" name="url" id="url"/>

Это поле будет проверять на правильность ввода URL-адреса, а на смартфоне откроется URL-клавиатура.

Week

<input id="conference" name="conference " type="week"/>

Это поле позволяет пользователю возможность выбрать определенную неделю.

**44. Атрибути полів введення форм в HTML5.**

С появлением HTML5 формы получили новые типы полей ввода и новые атрибуты для полей введения форм. Далее кратко описано, что из себя представляют некоторые новые атрибуты, которые могут быть использованы в сочетании с новыми или ранее существовавшими полями.

Autocomplete

<input type="text" name="fullname" id="fullname" autocomplete="on"/>

Автозаполнение - этот атрибут помогает пользователям заполнить поле на основе предыдущих записей.

Autofocus

<input type="text" name="username" id="username" autofocus/>

Этот атрибут ставит курсор в нужное поле при загрузке страницы, так что пользователь может сразу начать печатать.

Form

<input type="button" name="filter" form="filter/>

Form связывает поле с формой по идентификатору, поэтому элемент не обязательно должен быть в пределах соответствующей формы, чтобы быть её частью.

Formaction

<input type="submit" value="Send" formaction="email.php"/>

Указывает обработчик формы.

Formenctype

<input type="submit" value="Send" formenctype="application/x-www-form-urlencoded"/>

Устанавливает способ кодирования данных формы при их отправке на сервер.

Formmethod

<input type="submit" value="Send" formmethod="POST"/>

Определяет, какой метод HTTP использовать, чтобы отправить форму, например, POST или GET.

Formtarget

<input type="submit" value="Send" formtarget="\_self"/>

Определяет окно или фрейм в которое будет загружаться результат, возвращаемый обработчиком формы.

Pattern

<input pattern="[0-9]{5}" name="zip" type="text" title="Zipcode"/>

Pattern позволяет определить регулярное выражение в HTML-разметке поля, а не в JS.

Placeholder

<input type=" password " name="password" id=" password " placeholder="Введите ваш пароль"/>

Placeholder обычно отображается в виде светло-серого текста в поле, до тех пор пока поле не активно. Обычно этот атрибут содержит подсказку, что именно необходимо ввести в это поле.

Required

<input type="text" name="username" id="username" required/>

Атрибут 'required' делает поле обязательным к заполнению и форма не может быть отправлена, пока требуемые данные не будут введены. И нет необходимости использовать JS.

46 Создает область, в которой при помощи JavaScript можно рисовать разные объекты, выводить изображения, трансформировать их и менять свойства. При помощи тега <canvas>можно создавать рисунки, анимацию, игры и др.

49. Веб-страница хочет получить данные о местоположении посетителя. Посетитель может разрешить предоставлять эти данные или же не предоставлять их. Такое поведение браузера Chrome не является собственной инициативой его разработчиков, а официальным правилом стандарта геолокации, требующим, чтобы каждый веб-сайт, пытающийся получить данные о местоположении посетителя, получил для этого его разрешение.

Чтобы вычислить местоположение посетителя, браузер заручается помощью поставщика услуг определения местоположения (location provider), например, для Chrome это сервис Google Location Services. Определение местоположения является задачей не из легких, и поставщик применяет несколько разных подходов для ее решения.

Возможность геолокации предельно проста. Она состоит из трех методов объекта navigator.geolocation: getCurrentPosition(), watchPosition() и clearWatch().

Объект navigator — это сравнительно незначительная часть JavaScript. Его несколько свойств предоставляют информацию о текущем браузере и его возможностях. Наиболее полезным из них является свойство navigator.userAgent, которое предоставляет информационную строку, содержащую подробные данные о браузере, его версии, а также операционной системы, в которой он выполняется.

Для получения местоположения посетителя вызывается метод getCurrentPosition(). Но следует понимать, что процесс определения местоположения занимает определенное время, в течение которого никакой уважающий себя браузер не остановит всю деятельность на странице, пока ожидает эти данные. Поэтому метод getCurrentPosition() исполняется асинхронно, т.е. после его запуска продолжается выполнение следующего за ним кода. Когда определение местоположения завершится, для обработки результатов активируется другой фрагмент кода.

50 **Апле́т** ([англ.](https://uk.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D1%96%D0%B9%D1%81%D1%8C%D0%BA%D0%B0_%D0%BC%D0%BE%D0%B2%D0%B0) *Applet*) — коротка комп'ютерна програма, що функціонально розширює можливості основної програми, або інтернет-застосування. Наприклад, додає в [веб-сторінку](https://uk.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D1%81%D1%82%D0%BE%D1%80%D1%96%D0%BD%D0%BA%D0%B0) функцію анімації. Для створення аплетів в мережі Інтернет використовується мова програмування [Java](https://uk.wikipedia.org/wiki/Java).

[**Java**](https://uk.wikipedia.org/wiki/Java)**-аплет**

забезпечує можливість використовувати в World Wide Web «виконуваний вміст» шляхом виконання в гіпертекстовому документі міні-програм, відомих як **аплети**. Незвичайним є формат готового до виконання коду, у який транслюються аплети Java, — це апаратно-незалежний побайтовий формат. Завдяки використанню такого нейтрального формату відкомпільовані Java-програми можна переміщати по [Internet](https://uk.wikipedia.org/wiki/%D0%86%D0%BD%D1%82%D0%B5%D1%80%D0%BD%D0%B5%D1%82), оскільки вони не залежать від [операційної системи](https://uk.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D1%96%D0%B9%D0%BD%D0%B0_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0), у якій виконуються.

48 Основная идея в основе проверки форм HTML5 состоит в том, что разработчик указывает данные для валидации, но не реализует все необходимые для этого трудоемкие подробности. Например, допустим, что определенное поле нельзя оставлять пустым, и посетитель должен ввести в него хоть что-то. В HTML5 это осуществляется с помощью атрибута required в соответствующем поле:

<input id="name" placeholder="Иван Иванов" autofocus required>

45 Посмотрим, как выглядит самая обычная интернет-страница изнутри.

* **Структура HTML** по большому счету лежит в основе каждой web - страницы.
* Сегодня существует несколько типов **HTML** документов. Каждому типу документа соответствует его собственная **HTML структура** (подробности внизу страницы).
* **Правильная начальная структура HTML** является залогом кросс-браузерности сайта.

Пример структуры документа:

|  |
| --- |
| <html> <head> <title>HTML структура документа</title> </head> <body> <h3>Мой первый заголовок</h3> <p>Мой первый параграф</p> <p>Мой второй параграф</p> </body> </html> |

Рассмотрим HTML теги, определяющие структуру страницы

* <html> </html> – обязательные, определяют **HTML** документ.
* <head> </head> – определяют секцию со служебной информацией, содержат инструкции для поисковиков, для браузеров, скрипты.
* <title> </title> – определяют основной заголовок web - страницы.
* <body> </body> – обязательные, определяют видимую часть документа.
* <h3> </h3> – определяют заголовок 3-го уровня.

<p> </p> – определяют параграф. Смотрите урок [**HTML параграфы**](http://ab-w.net/HTML/html_paragraf.php)