Навчальний посібник
«Технології мультимедійних гіпервидань»

Київ 2021

ЗМІСТ

ПЕРЕДМОВА

JavaScript – мова, на якому сьогодні працює все – від веб-додатків до мобільних додатків і серверів. Його популярність різко зросла за останні кілька років, обійшовши такі мови, як Java і PHP.

JavaScript був створений Бренданом Ейхом в 1995 році. Брендан був прийнятий в компанію Netscape Communication, завданням якої було зробити веб більш динамічним. Через 10 днів, він створив прототип мови, з синтаксисом, дуже схожим на Java. Це був день народження JavaScript.

У 1996 JavaScript був переданий ECMA (Європейська асоціація виробників комп'ютерів), щоб зареєструвати новий стандарт мови. Це призвело до офіційного випуску ECMA-262. Незважаючи на те, що JavaScript вживається частіше, офіційна назва стандарту – ECMAScript.

Сьогодні кожен браузер підтримує його, тим самим роблячи JavaScript мовою інтернету.

Що таке JavaScript насправді? JavaScript – це повноцінна динамічна мова програмування, яка застосовується до HTML документу, і може забезпечити динамічну інтерактивність на веб-сайтах.

Базовою особливістю цієї мови відзначається те, що на неї вплинули інші (Python, Java тощо) мови програмування з метою надання максимального комфорту JavaScript і легкості в розумінні його тими користувачами, які не мають відповідної освіти і глибинних знань – не програмістів. JavaScript неймовірно універсальний і доброзичливий до новачків. Володіючи ж більшим досвідом, можна створювати ігри, анімовану 2D і 3D графіку, повномасштабні програми з базами даних і багато іншого!

JavaScript сам по собі досить компактний, але дуже гнучкий. Розробниками написано велику кількість інструментів поверх основної мови JavaScript, які розблоковують величезну кількість додаткових функцій з дуже невеликим зусиллям. До них відносяться: програмні інтерфейси додатка (API), вбудовані в браузери, що забезпечують різні функціональні можливості, такі як динамічне створення HTML і установку CSS стилів, захоплення і маніпуляція відеопотоком, робота з веб-камерою користувача або генерація 3D графіки і аудіо семплів.

Сторонні API дозволяють розробникам впроваджувати функціональність в свої сайти від інших розробників, таких як Twitter або Facebook.

Також можна застосувати до HTML сторонні фреймворки і бібліотеки, що дозволить прискорити створення сайтів і додатків.

Що може JavaScript в браузері? Сучасний JavaScript – це «безпечна» мова програмування. Вона не надає низькорівневий доступ до пам'яті або процесору, тому що від початку була створена для браузерів, які не потребують цього.

Можливості JavaScript сильно залежать від оточення, в якому він працює. Наприклад, Node.JS підтримує функції читання / запису довільних файлів, виконання мережевих запитів

і т.д. У браузері для JavaScript є все, що пов'язано з маніпулюванням веб-сторінками, взаємодією з користувачем і веб-сервером.

Наприклад, в браузері JavaScript може:

- Додавати новий HTML-код на сторінку, змінювати існуючий вміст, модифікувати стилі.
- Реагувати на дії користувача, клацання миші, перемістити вказівник, натискання клавіш.
- Відправляти мережеві запити на віддалені сервера, завантажувати і завантажувати файли (технології AJAX і COMET).
- Отримувати і встановлювати куки, задавати питання відвідувачеві, показувати повідомлення.
- Запам'ятовувати дані на стороні клієнта («local storage»).

Чого НЕ може JavaScript в браузері? Його можливості в браузері обмежені заради безпеки користувача. Мета полягає в запобіганні доступу недобросовісної веб-сторінки до особистої інформації або нанесення шкоди даним користувача. JavaScript може легко взаємодіяти з сервером, з якого прийшла поточна сторінка. Але його здатність отримувати дані з інших сайтів / доменів обмежена.

Що робить JavaScript особливим? Як мінімум, три сильні сторони JavaScript: повна інтеграція з HTML / CSS, прості речі робляться просто, підтримується всіма основними браузерами і включений за замовчуванням. JavaScript – це єдина браузерна технологія, що поєднує в собі все це. Ось чому це найпоширеніший інструмент для створення інтерфейсів в браузері.

ОСНОВИ JAVASCRIPT

JavaScript – це мова програмування, який активно використовується в HTML і при розробці сайтів.

Програми на мові JavaScript зазвичай називають сценаріями. Вони вставляються безпосередньо в HTML код веб-сторінки і виконуються браузером користувача. Сценарії JavaScript дозволяють додати веб-сторінці динамічність і зробити її інтерактивною.

JavaScript це один з 3 мов програмування, які повинен знати кожен, хто займається веб-розробкою:

HTML – визначає вміст веб-сторінки

CSS – визначає стилі відображення вмісту веб-сторінки

JavaScript – програмує поведінку веб-сторінки

При цьому область застосування JavaScript не обмежується веб-додатками. Багато десктопних і серверних програм також використовують JavaScript. Наприклад, програмна платформа Node.js, бази даних MongoDB і CouchDB.

Що таке JavaScript

JavaScript був створений для надання динамічності HTML сторінок, це скриптова мова або мова сценаріїв.

Мови сценаріїв – це спрощені мови програмування, створювані для роботи в певних середовищах.

Код, написаний на JavaScript, можна вставляти прямо в HTML-код веб-сторінки.

Сценарій JavaScript являє собою текстовий файл, тому написати його можна в простому текстовому редакторі, а для його роботи достатньо запустити в вікні браузера.

Завдяки скриптів JavaScript статичні HTML документи можна зробити динамічними і інтерактивними: різні візуальні ефекти, на зразок слайдеров, галерей картинок і динамічного тексту; перевірка призначених для користувача даних форми до їх відправки на сервер; виведення інформації в нових вікнах в автоматичному режимі; зміна вмісту вікна браузера, в залежності від дій користувача тощо.

Варто пам’ятати, що JavaScript і Java абсолютно різні, як за концепцією, так і по реалізації, мови програмування.

JavaScript був придуманий Бренданом Ейхом (Brendan Eich) в 1995 році і став стандартом ECMA в 1997 році. Офіційна назва стандарту – ECMA-262. А офіційне назви мови – ECMAScript.

Куди вставляти JavaScript-код

Щоб код Javascript спрацював в браузері користувача, його необхідно вставити між тегами `<script>` і `</script>`.

```
<html>
<head>
<title> Приклад JavaScript </title>
</head>
<body>
<script>
document.getElementById("demo").innerHTML = "Перший сценарій на JavaScript";
</script>
<noscript>
Ваш браузер не підтримує JavaScript або підтримка відключена в налаштуваннях.
</noscript>
<div id = "demo"> </ div>
</body>
</html>
```

В даному прикладі, як тільки HTML сторінка буде завантажена, браузер запустить команду `document.getElementById("demo").InnerHTML = "Перший сценарій на JavaScript"`, яка шукає елемент з ідентифікатором "demo" і, знайшовши його, поміщає в нього рядок " перший сценарій на JavaScript ".

Насправді повний запис тега `<script>` має такий вигляд:

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

В атрибуті `type` вказується використовувана мова скриптів. Однак в даний час існує не так вже й багато таких мов, і в HTML мову Javascript встановлений як мова скриптів за замовчуванням. Тому атрибут `type` використовувати не потрібно.

Також, зверніть увагу на теги `<noscript>` і `</noscript>`. Цей тег спрацьовує, коли з тієї чи іншої причини, наприклад, виконання сценаріїв відключено в налаштуваннях браузера, неможливо виконати сценарій JavaScript.

Функція JavaScript – це блок коду, який виконується за "викликом".

Наприклад, функція може викликатися у разі виникнення будь-небудь події, на зразок натискання користувачем на кнопку миші.

В HTML документ можна вставляти будь-яке число скриптів.

На HTML сторінці скрипти можна розміщувати всередині секції `<body>` або `<head>`, або в обох відразу.

Розміщення скриптів в нижній частині елемента `<body>` збільшує швидкість відображення HTML документа, так як компіляція скриптів уповільнює рендеринг веб-сторінки.

Зовнішній JavaScript

Скрипти також можна розміщувати в зовнішніх файлах:

```
Зовнішній файл: myScript.js
function myFunction () {
   document.getElementById("demo").innerHTML = "Параграф змінений.";
}
```

Зовнішні скрипти виправдані тоді, коли потрібно один і той же сценарій використовувати в декількох HTML документах.

Зазвичай у файлів JavaScript розширення .js.

Щоб використовувати зовнішній скрипт, потрібно помістити ім'я файлу Javascript в атрибут src тега `<script>`:

```
<script src = "// msiter.ru/myScript.js"> </ script>
```

Підключати зовнішні скрипти можна всередині тега `<head>` або `<body>`.

Скрипт буде вести себе так само, як якщо б він був вставлений безпосередньо всередині тега `<script>`.

Увага! Зовнішні скрипти не можуть містити теги `<script>`.

У розміщенні скриптів в зовнішніх файлах є ряд переваг:

- Розділяється HTML і Javascript код.
- Стає простіше читати і обслуговувати HTML і JavaScript.
- Завдяки кешування файлів JavaScript збільшується швидкість завантаження веб-сторінки.

Щоб додати кілька файлів скриптів на HTML сторінку, досить вставити потрібне число тегів `<script>`:

```
<script src = "// msiter.ru/myScript1.js"> </ script>
<script src = "// msiter.ru/myScript2.js"> </ script>
```

Зовнішні посилання

Як посилань на зовнішні скрипти може використовуватися або абсолютний URL, або відносний шлях до поточної веб-сторінці.

У наступному прикладі для підключення зовнішнього скрипта використовується повний URL:

```
<script src = "// msiter.ru/js/myScript1.js"> </ script>
```

Виведення даних Javascript

У JavaScript є кілька різних способів "відобразити" дані:

- Запис в HTML елемент за допомогою властивості `innerHTML`.
- Запис в висновок HTML за допомогою методу `document.write()`.
- Запис у вікно попереджень за допомогою методу `window.alert()`.
- Запис в консоль браузера за допомогою методу `console.log()`.

Використання innerHTML

Щоб отримати доступ до HTML елементу, JavaScript повинен скористатися методом `document.getElementById(id)`. Атрибут `id` визначає ідентифікатор HTML елемента. Властивість innerHTML визначає виведений HTML контент.

Зміна властивості innerHTML елемента HTML це звичайний спосіб виведення даних в HTML.

Приклад:

```
<html>
<body>
<h1>Моя веб-сторінка</h1>
<p>Перший параграф</p>
<p id="demo"></p>
<script>
 document.getElementById("demo").innerHTML = 5 + 6;
</script>
</body>
</html>
```

Використання document.write ()

У тестових цілях для виведення даних можна використовувати метод `document.write()`:

Слід пам'ятати, що використання методу `document.write()` після повного завантаження HTML документа видалить весь існуючий HTML код, тому метод document.write () слід використовувати тільки для тестування.

Приклад:

```
<html>
<body>
<h1>Моя веб-сторінка</h1>
<p>Перший параграф</p>
<script>
document.write(5 + 6);
```

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

Використання window.alert ()

Для відображення даних також можна використовувати вікно повідомлень. Для цього потрібно скористатися методом `window.alert()`:

Приклад:

```
<html>
<body>
<h1>Моя веб-стрінка</h1>
<p>Перший параграф</p>
<script>
window.alert(5 + 6);
</script>
</body>
</html>
```

Використання console.log ()

Під час налагодження скрипта, щоб побачити якісь дані, ви можете вивести їх в консоль браузера. Для цього використовується метод `console.log()`:

Приклад:

```
<html>
<body>
<script>
 console.log(5 + 6);
</script>
</body>
</html>
```

Вирази JavaScript

Комп'ютерна програма – цей набір інструкцій, які повинен виконати комп'ютер. У мовах програмування такі програмні інструкції називаються виразами. Програма на JavaScript – це набір програмних виразів.

Приклад:

```
var x, y, z; // Вираз 1
x = 5; // Вираз 2
y = 6; // Вираз 3
z = x + y; // Вираз 4
```

В HTML програма на JavaScript виконується веб-браузером.

Вирази JavaScript складається з: значень, операторів, операндів, ключових слів і коментарів.

Вираз, наведений в наступному прикладі, говорить браузеру, що потрібно вивести рядок "Привіт!" всередині HTML елемента з ідентифікатором `id = "demo"`:

```
document.getElementById("demo").innerHTML = "Привіт!";
```

Вирази в програмі на JavaScript виконуються послідовно, один за одним, в тому порядку, в якому вони написані.

Крапка з комою (;)

Крапка з комою (;) розділяє вирази JavaScript.

Додавайте крапку з комою після кожного виконуваного виразу:

```
var a, b, c; // Декларує три змінні
a = 5; // Присвоює значення 5 змінної a
b = 6; // Присвоює значення 6 змінної b
c = a + b; // Обчислює суму значень змінних a і b і результат привласнює змінної c
```

При використанні крапки з комою, можна розміщувати декілька виразів на одному рядку: a = 5; b = 6; c = a + b;

Використання крапки з комою в кінці виразу не обов'язкове, але наполегливо рекомендується.

Пропуски в JavaScript

JavaScript ігнорує множинні пропуски. Проте, для поліпшення читаності коду ви можете за бажанням додавати пропуски в свій скрипт.

Наступні рядки еквівалентні:

```
var person  =  "Петро";
var person="Петро";
```

Вважається доброю практикою додавати пропуски перед та після операторів (`= + - * /`): `var x = y + z;`

Довжина рядка і перенесення на новий рядок

Для кращої читання програмісти часто уникають писати рядки коду довше 80 символів. Якщо вираз JavaScript виходить занадто довгим, то його можна перенести на новий рядок. Зазвичай це роблять відразу після оператора:

```
 document.getElementById("demo").innerHTML =
"Вітання!";
```

Блоки коду JavaScript

Вирази JavaScript можуть групуватися в блоки коду. Це робиться шляхом розміщення группіруемих виразів всередині фігурних дужок {...}.

Блоки коду призначені для визначення виразів, які виконуються разом.

Одне з місць де можна зустріти вирази, згруповані в блоки й це функції JavaScript:

```
function myFunction () {
    document.getElementById("demo1").innerHTML = "Привіт!";
    document.getElementById("demo2").innerHTML = "Як справи?";
}
```

Зазвичай для візуального виділення блоків коду перед згрупованими виразами ставлять кілька пробілів.

Більш докладно про функції ви дізнаєтеся трохи пізніше.

Ключові слова JavaScript

Дуже часто у виразах JavaScript використовуються ключові слова, які наказують JavaScript виконати якісь дії.

Наприклад, ключове слово `var` говорить браузеру, що потрібно створити змінні:

```
var x, y;
x = 5 + 6;
y = x * 10;
```

У наступній таблиці представлені деякі ключові слова:

Ключове слово	**Опис**
`break`	Перериває цикл або перевірку по команді `switch`
`continue`	Припиняє поточну ітерацію циклу і продовжує виконання з наступного ітерації
`debugger`	Зупиняє виконання скрипта JavaScript і викликає (якщо можливо) функцію налагодження
`do ... while`	Створює цикл з перевіркою умови після кожної ітерації

Ключове слово	Опис
`for`	Створити цикл, вказавши початковий стан, умова і операцію оновлення стану
`function`	Декларує функцію
`if ... else`	Виконує той чи інший блок коду в залежності від того, чи правильно умова
`return`	Завершує функцію
`switch`	Порівнює значення виразу з різними варіантами і при збігу виконує відповідний код
`try ... catch`	Реалізує обробку помилки при виконанні блоку виразів
`var`	Декларує змінну

Ключові слова зарезервовані за мовою JavaScript. Ці слова не можна використовувати як імена змінних або функцій.

Ідентифікатори JavaScript

Ідентифікатори – це імена.

В JavaScript ідентифікатори використовуються для називання змінних (а також ключових слів, функцій, міток). У більшості мовах програмування правила складання коректних імен одні й ті ж.

В JavaScript першим символом ідентифікатора повинна бути буква, символ підкреслення (_) або символ долара ($). Наступними ж символами можуть бути літери, цифри, символи підкреслення або символи долара.

В якості першого символу ідентифікатора можна використовувати цифри. Так JavaScript відрізняє ідентифікатори від числових значень.

Всі ідентифікатори в JavaScript чутливі до регістру. Змінні `lastName` і `lastname` це різні змінні.

```
var lastname, lastName;
lastName = "Doe";
lastname = "Peterson";
```

Таким чином, ідентифікатори `VAR` або `Var` НЕ будуть інтерпретуватися JavaScript як ключове слово `var`.

JavaScript і "Горбатий Регістр"

Так склалося історично, що програмісти для імені однієї змінної використовували з'єднання декількох слів. Це дозволяє відразу ж зробити зрозумілим для чого призначена ця змінна. Щоб візуально розділити ці слова, використовуються різні способи.

1) Тире: Використання символу тире в ідентифікаторах в JavaScript заборонено. Тире зарезервований для операції віднімання.

2) Символ підкреслення: `first_name, last_name, master_card, inter_city`.

3) Верхній "горбатий регістр" (стиль мови Pascal). "Горбатий регістр" називається так за схожість написання ідентифікаторів з горбами двогорбої верблюда – окремі слова, складові ідентифікатор, починаються з великої літери, нагадуючи горб верблюда: `FirstName, LastName, MasterCard, InterCity`.

4) Нижній "горбатий регістр". Як правило, програмісти на JavaScript воліють використовувати "горбатий регістр", що починається з маленької літери: `firstName, lastName, masterCard, interCity`.

Коментарі Javascript

Коментарі використовуються для пояснення JavaScript коду, щоб зробити його більш зрозумілим.

Також, коментарі дозволяють закрити окремі ділянки JavaScript коду від виконання під час тестування альтернативних алгоритмів.

Однорядкові коментарі

Щоб визначити однорядковий коментарі, необхідно перед текстом коментарю написати подвійний прямий слеш (`//`). Будь-який текст між подвійним слешем (`//`) і кінцем рядка буде ігноруватися (не виконуватиметься) оброблювачем JavaScript.

У наступному прикладі перед кожним рядком коду визначається однорядковий коментар:

```
// Змінимо заголовок:
document.getElementById("myH").innerHTML = "Моя перша веб-сторінка";
// Змінимо параграф:
document.getElementById("myP").innerHTML = "Мій перший параграф.";
```

У наступному прикладі однорядковий коментар використовується в кінці кожного рядка для пояснення JavaScript коду:

```
var x = 5; // Декларуємо змінну x і присвоюємо їй значення 5
var y = x + 2; // Декларуємо змінну y і присвоюємо їй значення x + 2
```

Багаторядкові коментарі

Для визначення багаторядкового коментаря використовується конструкція `/*...*/`. Будь-який текст, що знаходиться між `/*` і `*/` буде ігноруватися оброблювачем JavaScript.

У наступному прикладі використовується багаторядковий коментар (блок коментаря) для пояснення JavaScript коду:

```
/*
Код нижче змінить
на моїй веб-сторінці
```

```
заголовок з id = "myH"
і параграф з id = "myP":
*/
document.getElementById("myH").innerHTML = "Моя перша веб-сторінка";
document.getElementById("myP").innerHTML = "Мій перший параграф.";
```

Як правило, частіше використовують однорядкові коментарі. Блоки коментарів зазвичай використовуються для офіційної документації.

Для того щоб протестувати роботу скрипта, дуже зручно закривати певні ділянки коду за допомогою коментарів, перевіряючи різні альтернативи.

Значення даних в JavaScript

В JavaScript існує два види значень: фіксовані значення (константи) та змінні значення.

1) Константи JavaScript

Головне, що потрібно знати при написанні фіксованих значення це те, що числа можуть записуються як з десятковою крапкою, так і без неї: 10.50, 1001

Рядки це текст, укладений в подвійні або одинарні лапки: "John Doe", 'John Doe'

2) Змінні JavaScript

У мовах програмування змінні використовуються для зберігання різних даних. Змінні JavaScript можуть зберігати як числа, так і текст.

У програмуванні текстові значення називаються текстові рядки.

Рядки записуються в подвійних або одинарних лапках. Числа записуються без лапок. Якщо помістити число в лапки, то воно буде сприйматися оброблювачем JavaScript, як текстовий рядок.

```
var pi = 3.14;
var person = "Іван Петров";
var answer = 'Так, це я!';
```

Декларування (створення) змінних JavaScript

Процес створення змінної в JavaScript називається "декларування" змінної.

Змінна JavaScript декларується за допомогою ключового слова `var`:

```
var carName;
```

Відразу після декларування у змінної немає ніякого значення.

Щоб привласнити змінної значення, слід скористатися оператором присвоювання:

```
 carName = "Volvo";
```

Також привласнити значення можна і під час декларування змінної:

```
var carName = "Volvo";
```

У наступному прикладі ми створюємо змінну з ім'ям `carName` і відразу ж присвоюємо їй значення `"Volvo"`. Потім ми "виводимо" значення змінної в HTML параграфі з атрибутом `id = "demo"`:

```
<p id = "demo"> </ p>
<script>
var carName = "Volvo";
document.getElementById("demo").innerHTML = carName;
</script>
```

Можна декларувати в одному виразі кілька змінних. Для цього потрібно почати вираз з ключового слова `var` і розділяти декларовані змінні коми:

```
var person = "John Doe", carName = "Volvo", price = 200;
```

Декларація може розташовуватися на декількох рядках:

```
var person = "John Doe",
carName = "Volvo",
price = 200;
```

Значення = undefined

У комп'ютерних програмах змінні часто декларуються без визначення значень. У цьому випадку передбачається, що значення в подальшому має бути обчислено, чи надано пізніше в результаті, наприклад, призначеного для користувача введення. При цьому у змінної, декларованої без значення, насправді буде значення undefined.

Якщо ви повторно декларуєте змінну JavaScript, то вона не втратить значення, яке у неї було до цього.

Оператори Javascript

В JavaScript існує безліч різних операторів: арифметичні оператори, оператори присвоювання, рядкові оператори, оператори порівняння, логічні оператори, оператори типів.

Арифметичні оператори JavaScript

Арифметичні оператори використовуються для виконання арифметичних операцій з числами:

Оператор	**Опис**
+	Додавання
-	Віднімання
*	Множення
/	Ділення
%	Залишок від ділення
++	Збільшення на 1 (інкремент)
--	Зменшення на 1 (декремент)

Оператори присвоювання JavaScript

Оператори присвоювання привласнюють значення змінним JavaScript.

Оператор	**Приклад**	**Аналог**
=	x = y	x = y
+ =	x + = y	x = x + y
− =	x − = y	x = x − y
* =	x * = y	x = x * y
/ =	x / = y	x = x / y
% =	x% = y	x = x% y

Рядкові оператори JavaScript

Оператор додавання (+) також може використовуватися для об'єднання (конкатенації) рядків.

```
txt1 = "John";
txt2 = "Doe";
txt3 = txt1 + " " + txt2;
```

В результаті в змінної `txt3` буде рядок – `"John Doe"`.

Також, для конкатенації рядків можна використовувати оператор присвоювання `+ =`:

```
txt1 = "Який теплий ";
txt1 + = "і чудовий день";
```

В результаті в змінної `txt1` буде рядок – `"Який теплий і чудовий день"`.

При використанні з рядками оператор + ще називають оператором конкатенації.

Додавання двох чисел поверне суму цих чисел, однак додавання числа і рядки поверне рядок:

```
x = 5 + 5;
y = "5" + 5;
z = "Hello" + 5;
```

В результаті в змінних `x`, `y`, `z` ми отримаємо `10`, `55`, `Hello5` відповідно.

Оператори порівняння JavaScript

Оператор	**Опис**
==	Дорівнює
===	Дорівнює за значенням і типом
! =	Не дорівнює
! ==	Не дорівнює за значенням і типом
>	Більше ніж
<	Менше ніж
> =	Більше або дорівнює
<=	Менше або дорівнює
?	Тернарний оператор

Логічні оператори JavaScript

Оператор	**Опис**
&&	Логічне І
||	Логічне АБО
!	Логічне НЕ

Оператори типів JavaScript

Оператор	**Опис**
typeof	Повертає тип змінної
instanceof	Повертає `true`, якщо об'єкт є екземпляром певного об'єктного типу

Типи даних Javascript

Щоб можна було оперувати змінними, вкрай важливо знати про їх типах.

Без типів даних комп'ютер не буде знати як безпечно обчислити, наприклад, такий вираз:

```
var x = 16 + "Volvo";
```

Чи має сенс додавати до рядка "Volvo" число 16? Це призведе до помилки або якому-небудь розумному результату?

JavaScript буде інтерпретувати наведений вище приклад наступним чином:

```
var x = "16" + "Volvo";
```

При додаванні числа і рядки JavaScript буде сприймати число як рядок.

JavaScript обчислює вирази зліва направо. Таким чином, різні послідовності можуть привести до різних результатів:

JavaScript	**Результат**
`var x = 16 + 4 + "Volvo";`	`20Volvo`
`var x = "Volvo" + 16 + 4;`	`Volvo164`

У першому прикладі JavaScript сприймає 16 і 4 як числа, поки не досягне рядки "Volvo".

У другому прикладі, так як перший операнд – рядок, всі наступні операнди також вважаються рядками.

В JavaScript всі типи даних є динамічними. Це означає, що одна і та ж змінна може використовуватися для зберігання даних різних типів:

```
var x; // x має тип undefined
x = 5; // тепер x – число
x = "John"; // тепер x – рядок
```

Рядки в JavaScript

Рядок (або текстовий рядок) це послідовність символів, наприклад, "Іван Петров". Рядки повинні записуватися всередині лапок. Це можуть бути подвійні або одинарні лапки:

```
var carName = "Volvo XC60"; // Використовуються подвійні лапки
var carName = 'Volvo XC60'; // Використовуються одинарні лапки
```

Усередині рядка можна використовувати лапки тільки в тому випадку, якщо вони відрізняються від лапок, в які укладений рядок:

```
var answer = "It's alright"; // Одинарні лапки всередині подвійних
var answer = "He is called 'Johnny'"; // Одинарні лапки всередині подвійних
var answer = 'He is called "Johnny"'; // Подвійні лапки всередині одинарних
```

Числа в JavaScript

В JavaScript існує тільки один тип числових даних.

Числа можуть записуватися як з десятковою крапкою, так і без неї:

```
var x1 = 34.00; // З десятковою крапкою
var x2 = 34; // Без десяткового дробу
```

Дуже великі і дуже малі числа можуть записуватися за допомогою експоненціальної нотації (має вид Мер, де М – число, е – означає "помножити на 10 у степені ...", р – порядок або степінь, до якої зводиться 10) :

```
var y = 123e5; // 12300000
var z = 123e-5; // 0.00123
```

Логічні дані в JavaScript

Є два логічних значення: true (істина) і false (брехня).

```
var x = 5;
var y = 5;
var z = 6;
(X == y) // Поверне true
(X == z) // поверне false
```

Логічні значення часто використовуються в перевірках різних умов.

Масиви в JavaScript

Масиви в JavaScript записуються за допомогою квадратних дужок.

Елементи масиву розділяються комами.

У наступному прикладі декларується (створюється) масив з ім'ям `cars`, що містить три елементи:

```
var cars = [ "Saab", "Volvo", "BMW"];
```

Індексація елементів масиву починається з нуля. Це означає, що перший елемент має індекс [0], другий [1] і так далі.

Об'єкти в JavaScript

Об'єкти в JavaScript записуються за допомогою фігурних дужок.

Властивості об'єктів записуються у вигляді пар ім'я: значення, розділених комами.

```
var person = {firstName: "John", lastName: "Doe", age: 50, eyeColor: "blue"};
```

У наведеному вище прикладі об'єкт з ім'ям `person` має 4 властивості: `firstName, lastName, age, eyeColor`.

Оператор typeof

Щоб визначити тип даних змінної в JavaScript використовується оператор `typeof`. Оператор `typeof` повертає тип даних змінної або виразу:

```
typeof "" // поверне "string"
typeof "John" // поверне "string"
typeof "John Doe" // поверне "string"
typeof 0 // поверне "number"
typeof 314 // поверне "number"
typeof 3.14 // поверне "number"
typeof (3) // поверне "number"
typeof (3 + 4) // поверне "number"
```

Події в Javascript

HTML подія може бути дією, яку робить браузер або користувач. При використанні на HTML сторінці скрипта JavaScript він може реагувати на ці події. Кілька прикладів HTML подій:

1) HTML сторінка закінчила завантажуватися

2) Поле введення було змінено

3) Користувач натиснув на HTML кнопку

Часто, при виникненні HTML події необхідно щось зробити. JavaScript дозволяє при виявленні потрібного події виконати необхідний код. Для цього у HTML елементів є спеціальні атрибути обробники подій, в які і можна додати JavaScript код:

1) З одинарними лапками: `<Елемент подія = 'код JavaScript'>`;

2) З подвійними лапками: `<Елемент подія = "код JavaScript">`.

У наступному прикладі елементу `button` доданий атрибут `onclick` з JavaScript кодом:

```
<button onclick = "document.getElementById ( 'demo'). InnerHTML = Date ()">
```

```
Яка зараз дата і час?
</button>
```

У наведеному прикладі при натисканні користувачем на кнопку код JavaScript змінить вміст елемента з `id = "demo"` і виведе в нього поточну дату і час.

У наступному прикладі JavaScript код змінить вміст самого елемента (використовується команда `this.innerHTML`):

```
<button onclick = "this.innerHTML = Date ()">
Яка зараз дата і час?
</button>
```

Однак, дуже рідко можна побачити код JavaScript, що складається з одного виразу. Тому зазвичай в атрибут події записують виклик JavaScript функції:

```
<button onclick = "displayDate ()">
Яка зараз дата і час?
</ button>
```

Нижче наводиться список деяких часто використовуваних HTML подій:

Подія	**Опис**
`onchange`	HTML елемент був змінений
`onclick`	Користувач клікнув мишкою на HTML елемент
`onmouseover`	Користувач навів мишку на HTML елемент
`onmouseout`	Користувач вивів мишку за межі HTML елемента
`onkeydown`	Користувач натиснув на клавішу клавіатури
`onload`	Браузер закінчив завантажувати сторінку

Оброблювач подій можна використовувати для обробки і перевірки користувальницького введення, дій користувача і браузера:

- Виконувати дії повторюються після завантаження сторінки;
- Виконувати дії повторюються після закриття сторінки;
- Дії, які повинні виконуватися при натисканні користувачем на кнопку;
- Перевірка даних, введених користувачем в поле форми тощо.
- В JavaScript існує безліч способів працювати з подіями:
- HTML атрибути-події можуть безпосередньо виконувати код JavaScript;
- HTML атрибути-події можуть викликати функції JavaScript;

- Ви можете встановити власну функцію обробки події;
- Ви можете заборонити відправку і обробку події тощо.

Умовні оператори If і Switch

Дуже часто потрібно, щоб в залежності від деяких умов виконувалися різні дії. Реалізувати все це дозволяють умовні оператори.

Умовні оператори використовуються для виконання певних дій в залежності від заданих умов.

В JavaScript є такі умовні оператори:

1) Оператор `if` використовується для визначення блоку коду, який буде виконуватися, якщо задана умова дотримується (повертає true).

```
if (умова) {
   блок коду, що виконується якщо умова повертає true
}
```

2) Оператор `else` є частиною і продовженням оператора `if` і використовується для визначення блоку JavaScript коду, який буде виконуватися, якщо задана умова не дотримується (повертає false).

```
if (умова) {
   блок коду, що виконується якщо умова повертає true
} else {
   блок коду, що виконується якщо умова повертає false
}
```

3) Оператор `else if` є частиною і продовженням оператора `if` і використовується для визначення нового умови, якщо перша умова не дотримується (повертає false).

```
if (умова 1) {
   блок коду, що виконується якщо умова 1 повертає true
} else if (умова 2) {
   блок коду, що виконується якщо умова 1 повертає false, а умова 2 true
} else {
   блок коду, що виконується якщо умова 1 і умова 2 повертають false
}
```

4) Оператор `switch` використовується для порівняння одного значення з безліччю інших і вибору відповідного блоку коду для виконання.

```
switch (вираз) {
   case n:
     блок коду
     break;
   case n1:
     блок коду
     break;
   default:
     блок коду
}
```

Спочатку обчислюється вираз в операторі `switch`.

Потім його значення порівнюється зі значеннями кожного оператора `case`.

Якщо знайдено збіг, то виконується відповідний блок коду. Коли збіг знайдено, і робота зроблена, приходить час припинити перевірку, так як в ній більше немає необхідності. Для цього і існує ключове слово `break`. Коли інтерпретатор JavaScript досягає ключове слово `break`, він перериває виконання блоку оператора `switch`. Весь код блоку оператора `switch`, розташований після нього, ігнорується.

Ключове слово `default` визначає блок коду, який виконується в тому випадку, якщо жодна з умов не співпала.

Блок вибору за умовою `default` не обов'язково повинен бути останнім у блоці оператора switch, однак тоді його потрібно закінчувати ключовим словом `break`.

Іноді виникає необхідність за різними умовами використовувати один і той же код.

У наступному прикладі умова 4 і 5 визначають загальний блок коду, а 0 і 6 визначають інший загальний блок коду:

```
switch (new Date (). getDay ()) {
    case 4:
    case 5:
        text = "Скоро вихідні";
        break;
    case 0:
    case 6:
        text = "Сьогодні вихідний";
        break;
    default:
        text = "Будемо чекати вихідних";
}
```

Оператори циклу For і While. Оператори Break і Continue

Оператори циклу виконують блок коду заданий число раз, або до тих пір, поки не виконається задана умова. Цикли особливо зручні, коли потрібно виконувати один і той же код багато разів, кожен раз з різним значенням.

Часто оператори циклу використовуються при роботі з масивами.

Так, замість того щоб писати:

```
text + = cars [0] + "<br>";
text + = cars [1] + "<br>";
text + = cars [2] + "<br>";
text + = cars [3] + "<br>";
text + = cars [4] + "<br>";
text + = cars [5] + "<br>";
```

Можна написати:

```
var i;
for (i = 0; i <cars.length; i ++) {
    text + = cars [i] + "<br>";
}
```

JavaScript підтримує різні види циклів:

1) Оператор `for` це найбільш часто використовуваний інструмент для створення циклу.

```
for (вираз 1; вираз 2; вираз 3) {
   виконуваний блок коду
}
```

Вираз 1 виконується до початку циклу (до початку виконання блоку коду). Зазвичай, вираз 1 використовується для ініціалізації змінної, яка буде використовуватися всередині циклу, як правило в якості лічильника (`i = 0`). При цьому вираз 1 є необов'язковим. Можна в вираженні 1 форматувати кілька змінних (розділяючи їх комами). Можна пропустити вираз 1 (і визначити всі необхідні значення до самого циклу), однак при цьому знак `;` необхідно залишити.

Вираз 2 визначає умову продовження циклу. Часто вираз 2 використовується для обчислення стану змінної-лічильника. При цьому вираз 2 також є необов'язковим. Якщо вираз 2 повертає true, то почнеться новий цикл. Якщо воно поверне false, то цикл закінчується.

Увага! Якщо вираз 2 не визначається, то всередині циклу повинен бути заданий оператор break. Інакше цикл ніколи не закінчиться, що призведе до краху браузера.

Вираз 3 виконується після кожного проходу циклу. Зазвичай, в виразі 3 змінюється значення змінної-лічильника. При цьому вираз 3 також є необов'язковим. Вираз 3 може виконувати будь-які дії, наприклад, зменшення (`i--`), збільшення (`i = i + 15`) і т.д. Вираз 3 може бути пропущено (наприклад, якщо обчислення зі змінною-лічильником здійснюються всередині циклу):

2) Оператор циклу `for / in` використовується для обходу в циклі властивостей об'єкта:

```
var person = {fname: "Іван", lname: "Петров", age: 25};

var text = "";
var x;

for (x in person) {
    text + = person [x];
}
```

3) Оператор циклу `while` виконує блок коду до тих пір, поки заданий умова дорівнює true.

```
while (умова) {
   виконуваний блок коду
}
```

Увага! Якщо ви забудете збільшити змінну, зазначену в умови, то цикл ніколи не закінчиться. Це призведе до краху браузера.

4) Оператор циклу `do / while` є варіантом циклу `while`. Цей цикл один раз виконає блок коду перед перевіркою умови завершення і потім буде повторювати цикл до тих пір, поки умова не дорівнюватиме true.

```
do {
   виконуваний блок коду
}
while (умова);
```

Не забувайте змінювати значення змінної, використовуваної в умови завершення циклу, інакше цикл ніколи не закінчиться!

Оператор `break` перериває цикл і передає виконання коду, наступного після оператора циклу (якщо є):

```
for (i = 0; i <10; i ++) {
    if (i === 3) {break; }
    text + = "Число:" + i + "<br>";
}
```

Оператор `continue` припиняє поточний прохід циклу, якщо виконується задана умова, і починає наступну ітерацію циклу.

У наступному прикладі пропускається значення 3:

```
for (i = 0; i <10; i ++) {
    if (i === 3) {continue; }
    text + = "Число:" + i + "<br>";
}
```

Щоб визначити мітку в JavaScript, ви повинні перед виразом вказати ім'я мітки з двокрапкою:

```
мітка:
   вираз
```

Оператори `break` і `continue` єдині оператори в JavaScript, які можуть "вистрибувати" з блоку коду.

```
break імя_мітки;
continue імя_мітки;
```

Оператор `continue` (з посиланням на мітку або без) єдиний спосіб пропустити один прохід циклу.

Оператор `break` (без посилання на мітку) єдиний спосіб вийти з циклу або умовного оператора `switch`.

Оператор `break` з посиланням на мітку дозволяє вийти з будь-якого блоку коду і перейти в конкретне місце програми:

Об'єкт Math

JavaScript об'єкт Math дозволяє вирішувати різні математичні завдання з числами.

Приклад:

```
pi = Math.PI; // повертає 3.141592653589793 (число пі)
```

На відміну від інших глобальних об'єктів, у об'єкта Math немає конструктора. Його методи і властивості є статичними.

Всі методи і властивості (константи) об'єкта Math можна використовувати без попереднього створення самого об'єкта.

Методи об'єкта Math

Метод	**Опис**
`abs (x)`	Повертає абсолютне значення від x
`acos (x)`	Повертає арккосинус кута x, в радіанах
`asin (x)`	Повертає арксинус кута x, в радіанах
`atan (x)`	Повертає арктангенс кута x, як числове значення в діапазоні від -PI / 2 до PI / 2 в радіанах
`atan2 (y, x)`	Повертає арктангенс приватного своїх аргументів
`ceil (x)`	Приводить число x до найближчого більшого цілого
`cos (x)`	Повертає косинус кута x (x повинен бути в радіанах)
`exp (x)`	Повертає експоненту від x (Ex)
`floor (x)`	Приводить число x до найближчого меншого цілого
`log (x)`	Повертає натуральний логарифм (по підставі E) числа x
`max (x, y, z, ..., n)`	Повертає найбільше значення в списку
`min (x, y, z, ..., n)`	Повертає найменше значення у списку
`pow (x, y)`	Зводить значення x в ступінь y
`random ()`	Повертає випадкове число між 0 і 1
`round (x)`	Повертає округлене значення x
`sin (x)`	Повертає синус кута x (x повинен бути в радіанах)
`sqrt (x)`	Повертає квадратний корінь від x
`tan (x)`	Повертає тангенс кута

Питання для самоперевірки:

1. Надати загальну характеристику JavaScript.
2. Пояснити, як відбувається оброблення подій методами JavaScript.
3. Пояснити, якими способами JavaScript вставляється до гіпервидання.
4. Назвати та описати методи виведення даних JavaScript.
5. Надати характеристику виразів у JavaScript.
6. Надати характеристику ідентифікаторів у JavaScript.
7. Пояснити, як декларуються (створюються) змінні у JavaScript.
8. Назвати та надати коротку характеристику операторів JavaScript: арифметичні, порівняння, рядкові та присвоєння. Навести приклади.

9. Назва ти та надати опис типів даних у JavaScript.

10. Надати характеристику умовних операторів у JavaScript. Навести приклади.

11. Надати характеристику операторів циклу у JavaScript. Навести приклади.

12. Надати характеристику операторів Break і Continue у JavaScript. Навести приклади.

13. Пояснити поняття функції JavaScript, їх синтаксис, виклик та повернення результату. Навести приклади.

ФУНКЦІЇ ТА ОБ’ЄКТИ JAVASCRIPT

Функції JavaScript

Функція JavaScript це блок коду, призначений для виконання певного завдання. По суті функції JavaScript це те ж саме, що і процедури або підпрограми в інших мовах програмування.

Функція JavaScript виконується тільки тоді, коли "щось" звертається до неї (викликає її).

```
function myFunction (p1, p2) {
    return p1 * p2; // Ця функція повертає похідне p1 і p2
}
```

Синтаксис функції JavaScript

Функція JavaScript декларується за допомогою ключового слова `function`, за яким слідує ім'я і круглі дужки ().

Ім'я функції може складатися з букв, цифр, символу підкреслення і символу долара (ті ж правила, що і зі змінними).

Усередині круглих дужок можуть визначатися імена параметрів, розділених комами: (параметр1, параметр2, ...).

Виконуваний код функції поміщається всередині фігурних дужок {}:

```
function ім'я (параметр1, параметр2, параметр3) {
    виконуваний код
}
```

Параметри функції перераховуються всередині круглих дужок () в заголовку функції.

Аргументи функції це значення, одержувані функцією при її виклику. Усередині функції аргументи (параметри) поводяться як локальні змінні.

Виклик функції

Код всередині функції виконується тільки тоді, коли "щось" звертається до неї (викликає її):

- Коли виникає якась подія (наприклад, користувач натискає на кнопку);
- Коли функція викликається з JavaScript коду;
- Автоматично (самовиклик).

Повернення результату з функції

Коли інтерпретатор JavaScript досягає виразу `return`, функція припиняє виконуватися. Якщо функція була викликана з коду, інтерпретатор поверне виконання коду, розташованого після виразу, що викликав функцію.

Часто функції повертають деяке значення, яке вони вираховували. Повернене значення "повертається" назад тому, хто її викликав. Виклик функції без дужок () поверне визначення функції, а не результат її роботи.

Приклад. Обчислимо добуток двох чисел і повернемо результат:

```
// Викликається функція, значення, що повертає
// зберігається в змінній x
var x = myFunction (4, 3);
function myFunction (a, b) {
    return a * b; // Функція повертає твір від a і b
}
```

В результаті в змінну `x` буде отримано значення 12.

Функції дозволяють повторно використовувати код: одного разу визначається блок коду і потім використовується скільки завгодно раз з будь-якого місця скрипта з різними аргументами, щоб отримати різні результати.

Функції можуть використовуватися так само як змінні у всіх типах формул, в операторах присвоювання і в будь-яких обчисленнях. Ви можете використовувати саму функцію, як значення змінної:

Об'єкти JavaScript

У реальному житті все що нас оточує є об'єктами. Наприклад, машина.

У машини є властивості – вага і колір, і методи – поїхати і зупинитися:

Об'єкт	**Властивості**	**Методи**

Машина
 
	`car.name = Fiat`
	`car.model = 500`
	`car.weight = 850kg`
	`car.color = white`
		`car.start ()`
		`car.drive ()`
		`car.brake ()`
		`car.stop ()`

У всіх машин одні й ті ж властивості, однак значення цих властивостей у різних машин різні. У всіх машин одні й ті ж методи, однак ці методи виконуються в різний час.

Змінні в JavaScript це контейнери для зберігання даних. Об'єкти це теж змінні. Однак об'єкти можуть зберігати кілька значень.

У наступному коді змінної з ім'ям car присвоюється кілька значень (Fiat, 500, white):

```
var car = {type: "Fiat", model: "500", color: "white"};
```

Тут значення записуються у вигляді пар ім'я: значення (ім'я та значення розділяються двокрапкою).

Об'єкти в JavaScript це контейнери для іменованих значень, які називаються властивості або методи.

Об'єкти в JavaScript визначаються (створюються) за допомогою фігурних дужок:

```
var person = {firstName: "Іван", lastName: "Петров", age: 50, eyeColor: "блакитні"};
```

Пропуски і переклади рядка не мають значення. Визначення об'єкта може займати кілька рядків:

```
var person = {
    firstName: "Іван",
    lastName: "Петров",
    age: 50,
    eyeColor: "блакитні"
};
```

Властивості об'єкта

Пари ім'я: значення в об'єктах JavaScript називаються властивостями:

Властивість	Значення
firstName	Іван
lastName	Петров
age	50
eyeColor	блакитні

Звернутися до властивості об'єкта можна двома способами: імяОб'єкта.імяВластивості, або імяОб'єкта ["імяВластивості"].

```
Приклад 1:  person.lastName;
Приклад 2:  person [ "lastName"];
```

Методи об'єктів

Крім властивостей у об'єктів так само можуть бути методи.

Методи – це дії, які можуть бути виконані з об'єктом.

Методи зберігаються у властивостях у вигляді визначень функцій.

Властивість	**Значення**
firstName	Іван
lastName	Петров
age	50
eyeColor	блакитні
fullName	function () {return this.firstName + " " + this.lastName;}

Таким чином, метод – це функція, що зберігається як властивість об'єкта.

```
var person = {
    firstName: "Іван",
    lastName: "Петров",
                id: 5566,
    fullName: function () {
        return this.firstName + "" + this.lastName;
};
```

У коді функції ключове слово `this` посилається на "власника" цієї функції.

У наведеному вище прикладі ключове слово `this` вказує на об'єкт `person`, який "володіє" функцією `fullName`. Іншими словами вираз `this.firstName` означає властивість `firstName`, що належить даному об'єкту.

Звернення до методів об'єкта

Звернутися до методу об'єкта можна наступним чином:

```
імяОб'єкта.імяМетода ()
```

```
Приклад:  name = person.fullName ();
```

Якщо ви звернетеся до методу без дужок (), то вираз поверне визначення функції:

```
name = person.fullName;
```

В JavaScript якщо змінна декларується з ключовим словом "new", то в цієї змінної буде створено об'єкт:

```
var x = new String (); // Змінна x декларується як об'єкт String
var y = new Number (); // Змінна y декларується як об'єкт Number
var z = new Boolean (); // Змінна z декларується як об'єкт Boolean
```

Намагайтеся уникати створення об'єктів `String`, `Number` і `Boolean`. Вони ускладнюють код скрипта і уповільнюють швидкість його роботи.

Рядок (об'єкт String)

Рядки в JavaScript використовуються для зберігання і маніпулювання текстовими даними. Рядком в JavaScript є нуль або більше символів, укладених в лапки.

```
var x = "Іван Петров";
```

Можна використовувати як одинарні, так і подвійні лапки:

```
var carname = "Volvo XC60"; // Подвійні лапки
var carname = 'Volvo XC60'; // Одинарні лапки
```

Усередині рядка також можна використовувати лапки, якщо вони відрізняються від лапок, в які укладена рядок:

```
var answer = "Його звали 'Іван'";
var answer = 'Його звали "Іван"';
```

Спеціальні символи

У зв'язку з тим, що рядок повинен полягати в лапки, JavaScript не зрозуміє наступний рядок:

```
var x = "Ми, так звані, "вікінги" з півночі.";
```

Цей рядок буде обрізана до "`Ми, так звані,`".

Для вирішення цієї проблеми потрібно скористатися екрануючим символом.

Екранує символ (\) перетворює спеціальні символи в символи рядка:

Код	Результат	Опис
\'	'	Одинарна лапки
\"	"	Подвійна лапки
\\	\	Зворотний слеш

Так, послідовність \" вставляє в рядок символ подвійної лапки:

```
var x = "Ми, так звані, \"вікінги\" з півночі.";
```

Крім цього в JavaScript існує ще шість екранованих послідовностей:

Код	Результат
\b	Повернення на крок (Backspace)
\f	Подача сторінки
\n	Новий рядок
\r	Повернення каретки
\t	Горизонтальна табуляція
\v	Вертикальна табуляція

Всі ці 6 екранованих символів спочатку були введені для управління телетайпних, друкованих та факс пристроїв. В HTML ці символи не мають сенсу.

Перенесення довгих рядків

Для кращої читання програмісти зазвичай дуже рідко пишуть рядки коду довше 80 символів. Якщо строкове вираження JavaScript не поміщається на одному рядку, то його можна перенести на новий рядок. Робити перенесення найкраще після оператора:

```
document.getElementById ( "demo"). innerHTML =
"Hello Dolly!";
```

Також, можна розбити рядок за допомогою символу зворотного слеша (\):

```
document.getElementById ( "demo"). innerHTML = "Hello \
Dolly! ";
```

Однак це не кращий спосіб, так як у нього немає універсальної підтримки. Деякі браузери не допускають використання пропусків після символу \.

Найбільш безпечним способом перенесення рядків є оператор рядкового додавання:

```
document.getElementById ( "demo"). innerHTML = "Hello" +
"Dolly!";
```

Не можна використовувати символ зворотного слеша (\) для перенесення на новий рядок коду:

```
document.getElementById ( "demo"). innerHTML = \
"Hello Dolly!";
```

Рядки можуть бути об'єктами.

Зазвичай, рядки JavaScript це примітивні значення, створені за допомогою літералів. Однак, за допомогою ключового слова new рядки також можна визначити і як об'єкти.

Приклад:

```
var x = "John";
var y = new String ( "John");
// typeof x поверне тип string
// typeof y поверне тип object
```

Увага! Не визначайте рядки як об'єкти. Це уповільнює швидкість виконання скрипта. Крім цього, ключове слово new в даному випадку ускладнює код і може призвести до несподіваних результатів:

При використанні оператора порівняння ==, однакові рядки рівні:

```
var x = "John";
var y = new String ( "John");
// (x == y) буде істинним (true), тому що x і y мають рівні значення
```

Однак, при використанні оператора порівняння ===, однакові рядки не будуть рівними, тому що оператор === очікує збіги як за значенням, так і за типом.

```
var x = "John";
var y = new String ( "John");

// (x === y) буде хибним (false), тому що x і y не рівні по типу (string і object)
```

Або ще гірше. Об'єкти не порівнюються:

```
var x = new String ( "John");
```

```
var y = new String ( "John");

// (x == y) – false, тому що x і y різні об'єкти
// (x === y) – false, тому що x і y різні об'єкти
```

Зверніть увагу на різницю між (`x == y`) і (`x === y`).

При порівнянні двох об'єктів JavaScript завжди повертається хиба (`false`).

Методи об'єкта String

У примітивних значень або літералів, типу рядка "Іван Петров", не може бути властивостей або методів (бо вони не об'єкти). Однак, в JavaScript все влаштовано так, що навіть у примітивних значень доступні свої методи і властивості, тому що JavaScript інтерпретує будь-які значення, як об'єкти з виконуваними методами і властивостями.

Вбудовані методи і властивості рядків допомагають працювати з цими самими рядками.

Довжина рядка

За допомогою властивості `length` можна дізнатися довжину рядка:

```
var txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
var sln = txt.length;
```

Пошук підрядка в рядку

Метод `indexOf()` повертає індекс (позицію) першого входження заданого тексту в рядку:

```
var str = "Please locate where 'locate' occurs!";
var pos = str.indexOf("locate");
```

Метод `lastIndexOf()` повертає індекс останнього входження заданого тексту в рядку:

```
var str = "Please locate where 'locate' occurs!";
var pos = str.lastIndexOf("locate");
```

Якщо підрядок не знайдено, то обидва методи, `indexOf()` і `lastIndexOf()`, повернуть -1.

Увага! JavaScript вважає позицію від нуля. 0 це перший символ в рядку, 1 – другий, 2 – третій і т.д.

Обидва методи в якості другого параметра приймають початкову позицію пошуку:

```
var str = "Please locate where 'locate' occurs!";
```

```
var pos = str.indexOf ( "locate", 15);
```

Крім цього, для пошуку підрядка існує метод `search()`, який повертає позицію знайденого підрядка:

```
var str = "Please locate where 'locate' occurs!";
var pos = str.search("locate");
```

Метод `search ()` не приймає другий, пов'язаний з позицією, параметр.

Метод `indexOf ()` не приймає "посилені" пошукові значення (регулярні вирази).

Витягування частини рядка

Існує три методи, що дозволяють отримати частину рядки:

```
slice (початок, кінець)
substring (початок, кінець)
substr (початок, довжина)
```

Метод `slice()` витягує частину рядка і повертає витягнуту частина в новому рядку.Цей метод приймає 2 параметра: початковий і кінцевий індекс (позицію).

У наступному прикладі вирізається частина рядка, починаючи з позиції 7 і закінчуючи позицією 13:

```
var str = "Apple, Banana, Kiwi";
var res = str.slice(7, 13);
```

В результаті в змінній `res` буде рядок `"Banana"`.

Якщо в параметрі задані від’ємні значення, то позиція буде відраховуватися з кінця рядка. У наступному прикладі вирізається частина рядка з позиції – 12 до позицією – 6:

```
var str = "Apple, Banana, Kiwi";
var res = str.slice (-12, -6);
```

В результаті в змінної `res` буде рядок `"Banana"`.

Якщо другий параметр не вказано, то вирізається весь підрядок, починаючи з заданої позиції. Задавати також можна додатні і від’ємні значення.

Увага! Від’ємні значення параметрів не працюють в Internet Explorer 8 і більше ранніх версіях.

Метод `substring()` схожий на метод `slice()`. Різниця між ними в тому, що метод `substring()` не може приймати в якості параметрів від’ємні значення.

```
var str = "Apple, Banana, Kiwi";
var res = str.substring(7, 13);
```

В результаті в змінної `res` буде рядок `"Banana"`.

Якщо другий параметр не вказано, то метод `substring()` виріже весь підрядок, починаючи з заданої позиції.

Метод `substr()` аналогічний методу `slice()`. Різниця між ними в тому, що другим параметром у методі `substr()` задається довжина підрядка, який витягується.

```
var str = "Apple, Banana, Kiwi";
var res = str.substr (7, 6);
```

В результаті в змінної `res` буде рядок `"Banana"`.

Якщо в першому параметрі задано від'ємне значення, то позиція буде відраховуватися з кінця рядка.

Другий параметр не може бути від'ємним, так як він визначає довжину підрядка.

Якщо другий параметр не вказано, то метод `substr()` виріже весь підрядок, починаючи з заданої позиції.

Заміна вмісту рядка

Метод `replace()` шукає в рядку задану підрядок і замінює її іншим значенням:

```
str = "Відвідайте Microsoft!";
var n = str.replace ("Microsoft", "MSiter");
```

В результаті в змінної n буде рядок `"Відвідайте MSiter!"`.

Метод `replace()` не змінює вихідний рядок. Він повертає новий рядок.

За замовчуванням, метод `replace ()` замінює перший підрядок, який співпадає:

```
str = "Відвідайте Microsoft і Microsoft!";
var n = str.replace ("Microsoft", "MSiter");
```

В результаті в змінної `n` буде рядок `"Відвідайте MSiter і Microsoft!"`.

За замовчуванням, метод `replace()` регістрозалежний. Тому запис підрядка `MICROSOFT` (великими літерами) в наступному прикладі не спрацює:

```
str = "Відвідайте Microsoft!";
var n = str.replace ("MICROSOFT", "MSiter");
```

Щоб замінити підрядок незалежно від регістру літер, використовуйте регулярний вираз з прапором `/i`:

```
str = "Відвідайте Microsoft!";
var n = str.replace(/MICROSOFT/i, "MSiter");
```

Регулярні вирази записуються без лапок.

Щоб замінити всі знайдені збіги, використовуйте регулярний вираз з прапором /g:

```
str = "Відвідайте Microsoft i Microsoft!";
var n = str.replace(/Microsoft/g, "MSiter");
```

Зміна регістра букв

Метод `toUpperCase()` дозволяє перетворити рядок у верхній регістр:

```
var text1 = "Hello World!";
var text2 = text1.toUpperCase();
```

Метод `toLowerCase()` дозволяє перетворити рядок в нижній регістр:

```
var text1 = "Hello World!";
var text2 = text1.toLowerCase();
```

Об’єднання рядків

Метод `concat()` об'єднує дві і більше рядків:

```
var text1 = "Hello";
var text2 = "World";
var text3 = text1.concat(" ", text2);
```

Метод `concat()` може використовуватися замість оператора +. Наступний приклад робить те ж саме, що і попередній:

```
var text = "Hello" + " " + "World!";
var text = "Hello".concat(" ", "World!");
```

Слід пам'ятати, що всі методи рядків повертають новий рядок. Вони ніяк не змінюють вихідну рядок. Якщо говорити формально, рядки мають імунітет. Вони не можуть змінюватися, їх можна тільки замінити.

Витягування символу з рядка

Існує 2 безпечних методу для вилучення символу з рядка:

```
charAt (позиція)
charCodeAt (позиція)
```

Метод `charAt()` повертає символ рядка, розташований в заданій позиції:

```
var str = "HELLO WORLD";
str.charAt(0); // поверне H
```

Метод `charCodeAt()` повертає код символу рядка, розташованого в заданій позиції:

```
var str = "HELLO WORLD";
str.charCodeAt (0); // поверне 72
```

Перетворення рядка в масив

За допомогою методу `split()` рядок можна перетворити в масив:

```
var txt = "a, b, c, d, e"; // Рядок
txt.split ( ","); // Поділ по комам
txt.split ( ""); // Поділ по прогалин
txt.split ( "|"); // Поділ по вертикальної межі
```

Якщо роздільник не вказано, то буде повернуто масив, який складається з одного елемента – початкового рядка, розташованого за індексом [0].

Якщо в якості роздільника вказана порожній рядок " ", то буде повернуто масив, в якому рядок розділено посимвольний:

```
var txt = "Hello"; // Рядок
txt.split(" "); // Поділ по символам
```

Дата (Об'єкт Date)

JavaScript об'єкт Date дозволяє працювати з датами. За замовчуванням, JavaScript буде використовувати часову зону браузера і відображати дату у вигляді повної текстової рядки: "Wed Oct 24 2018 10:48:26 GMT + 0300".

Створення об'єктів Date

Об'єкт Date створюється за допомогою конструктора `new Date ()`.

Існує 4 способи створення нового об'єкта дати:

- `new Date ();`
- `new Date (рік, місяць, день, години, хвилини, секунди, мілісекунди);`
- `new Date (мілісекунди);`
- `new Date (рядок дати).`

Конструктор `new Date()` створює новий об'єкт дати з поточними датою і часом:

```
var d = new Date();
```

Увага! Об'єкт Date статичний. Час на комп'ютері продовжує йти, але об'єкт дати не змінюється.

Конструктор `new Date(рік, місяць, ...)` створює новий об'єкт дати з заданими датою і часом:

1) 7 числових параметрів визначають рік, місяць, день, години, хвилини, секунди, мілісекунди (саме в такому порядку):

```
var d = new Date(2018, 11, 24, 10, 33, 30, 0);
```

Увага! В JavaScript нумерація місяців йде від 0 до 11. Січень – 0. Грудень – 11.

2) 6 числових параметрів визначають рік, місяць, день, години, хвилини, секунди:

```
var d = new Date(2018, 11, 24, 10, 33, 30);
```

3) 5 числових параметрів визначають рік, місяць, день, години, хвилини:

```
var d = new Date(2018, 11, 24, 10, 33);
```

4) 4 числових параметрів визначають рік, місяць, день, години:

```
var d = new Date(2018, 11, 24, 10);
```

5) 3 числових параметра визначають рік, місяць, день:

```
var d = new Date(2018, 11, 24);
```

6) 2 числових параметра визначають рік, місяць:

```
var d = new Date(2018, 11);
```

Увага! Не можна опускати параметр місяці. Якщо задається один параметр, то він буде інтерпретуватися як мілісекунди.

Якщо рік вказано у вигляді однієї або двох цифр, то він інтерпретується як 19 xx:

```
var d1 = new Date(99, 11, 24);

var d2 = new Date(9, 11, 24);
```

Конструктор `new Date(рядок дати)` створює новий об'єкт дати за спеціальним текстовим рядком:

```
var d = new Date("October 13, 2014 11:13:00");
```

Конструктор `new Date(мілісекунди)` створює новий об'єкт дати як нульова дата плюс кількість заданих мілісекунд. JavaScript зберігає дати у вигляді кількості мілісекунд, що пройшли з 1 січня 1970, 00:00:00 UTC (універсальні координати часу).

Нульова дата – це: 1 січня 1970 00:00:00 UTC.

Зараз у нас 1540367306081 мілісекунд з 1 січня 1970.

В добі (24 години) 86 400 000 мілісекунд.

1 січня 1970 плюс 100 000 000 000 мілісекунд дасть 3 березня 1973:

```
var d = new Date (100000000000);
```

1 січня 1970 мінус 100 000 000 000 мілісекунд дасть 31 жовтня 1966:

```
var d = new Date (-100 млрд);
```

Формати дати

В JavaScript існує 3 основних типи формату введення дати:

Тип	**Приклад**
Запис по ISO	"2015-03-25" (Міжнародний стандарт)
Короткий запис	"03/25/2015"
Довгий запис	"Mar 25 2015" або "25 Mar 2015"

ISO 8601 – це міжнародний стандарт представлення дати і часу. Формат ISO відповідає строгому стандарту JavaScript. Інші формати визначені не так добре, тому можуть відрізнятися в залежності від браузера.

Виведення дати в JavaScript

Незалежно від використаного формату введення, JavaScript за замовчуванням виводить дату в форматі повної текстової рядки:

```
Wed Mar 25 2015 3:00:00 GMT + 0300
```

JavaScript дати в форматі ISO

1) Синтаксис стандарту ISO 8601 (РРРР-ММ-ДД, де РРРР – повний рік, ММ – номер місяця, ДД – день) також є віддається перевага форматом дати в JavaScript.

Приклад (Повна дата):

```
var d = new Date("2015-03-25");
```

Увага! Дата обчислюється щодо часової зони користувача. Так, в залежності від місця розташування користувача результат вищенаведеного прикладу буде коливатися між 24 і 25 березнем.

2) Дата за стандартом ISO також може записуватися і без визначення дня (РРРР-ММ):

```
var d = new Date ( "2015-03");
```

В цьому випадку в залежності від часового поясу користувача дата буде коливатися між 28 лютого і 1 березня.

3) Крім цього, дата за стандартом ISO також може записуватися без визначення місяця і дня (РРРР):

```
var d = new Date("2015");
```

В цьому випадку в залежності від часового поясу користувача дата буде коливатися між 31 грудня 2014 та 1 січня 2015.

4)Дата за стандартом ISO може записуватися з додаванням годин, хвилин і секунд (РРРР-ММ-ДДТЧЧ: ММ: ССZ):

```
var d = new Date("2015-03-25T12: 00: 00Z");
```

Дата і час поділяються заголовної латинською літерою Т.

Час по UTC визначається заголовної латинською літерою Z.

Якщо ви хочете вказати час відносно UTC, то приберіть букву Z і додайте зрушення в форматі + ГГ: ХХ або -ЧЧ: ММ:

```
var d = new Date("2015-03-25T12:00:00-06:30");
```

UTC (Універсальні координати часу – анг. Universal Time Coordinated) те ж саме, що GMT (GMT – анг. Greenwich Mean Time).

Увага! Якщо в рядку дати-часу й вказати великі латинські літери T і Z, то результат буде різним у різних браузерах.

Часові зони

При установці дати без вказівки часової зони JavaScript буде використовувати тимчасову зону браузера. При отриманні дати без вказівки часової зони результат буде перетворений в часову зону браузера.

Іншими словами: Якщо дата / час створюються по GMT (час по Грінвічу), то дата / час будуть перетворені в київську годину, якщо браузер користувача працює в Києві.

Короткий запис дати

Короткий запис дати має наступну форму "ММ / ДД / РРРР", де РРРР – повний рік, ММ – номер місяця, ДД – день.

Приклад:

```
var d = new Date ( "03/25/2015");
```

Увага! У деяких браузерах вказівку місяці і дні без нуля на початку може привести до помилки:

```
var d = new Date("2015-3-25");
```

Поведінка браузера при форматі запису "РРРР / ММ / ДД" не визначене.

Деякі браузери спробують вгадати формат. Деякі повернуть значення NaN (not-a-number, що не число).

```
var d = new Date("2015/03/25");
```

Поведінка браузера при форматі запису "ДД-ММ-РРРР" також не визначено. Деякі браузери спробують вгадати формат. Деякі повернуть значення NaN.

```
var d = new Date("25-03-2015");
```

Довгий запис дати

Довгий запис дати має наступний синтаксис "МММ ДД РРРР":

```
var d = new Date("Mar 25 2015»);
```

Місяць і день можуть бути в будь-якому порядку:

```
var d = new Date("25 Mar 2015»);
```

Місяць може записуватися або повністю (January), або скорочено (Jan), коми ігноруються. Імена регістронезалежні:

```
var d1 = new Date("January 25 2015»);
var d2 = new Date("Jan 25 2015»);
var d = new Date("JANUARY, 25, 2015»).
```

Методи об'єкта Date

Коли об'єкт Date створений, ви можете оперувати їм за допомогою його методів. Методи об'єкта Date дозволяють отримувати і встановлювати рік, місяць, день, годину, хвилини, секунди і мілісекунди в об'єкті дати, використовуючи як локальний час, так і UTC або GMT.

Відображення дати

За замовчуванням, JavaScript буде відображати дату в форматі повної текстової рядки: Wed Mar 25 2015 3:00:00 GMT + 0300.

Коли ви виводите об'єкт дати в HTML, він автоматично перетворюється в рядок за допомогою методу `toString()`.

Приклад:

```
d = new Date();
document.getElementById("demo").innerHTML = d;
```

Теж саме:

```
d = new Date();
document.getElementById("demo").innerHTML = d.toString();
```

Метод `toUTCString()` перетворює дату в рядок UTC (стандарт відображення дати).

```
var d = new Date();
document.getElementById("demo").innerHTML = d.toUTCString();
```

Метод `toDateString()` перетворює дату в більш читабельний формат:

```
var d = new Date();
document.getElementById("demo").innerHTML = d.toDateString();
```

Введення дати за допомогою парсинга

Коректно скласти рядок дати можна за допомогою методу `Date.parse()` перетворити в мілісекунди. Метод `Date.parse ()` повертає кількість мілісекунд, що пройшли з 1 січня 1970 до заданої дати:

```
var msec = Date.parse ( "March 21, 2012");
document.getElementById("demo").innerHTML = msec;
```

Надалі ці мілісекунди можна перетворити в об'єкт дати:

```
var msec = Date.parse("March 21, 2012");
var d = new Date(msec);
document.getElementById("demo").innerHTML = d;
```

Наступні методи можна використовувати для отримання інформації з об'єкта дати:

Метод	**Опис**
`getFullYear()`	Отримати рік в форматі чотирьох цифр (рррр)
`getMonth()`	Отримати номер місяця (0-11)
`getDate()`	Отримати число місяця (1-31)
`getHours()`	Отримати годину (0-23)
`getMinutes()`	Отримати хвилини (0-59)
`getSeconds()`	Отримати секунди (0-59)
`getMilliseconds()`	Отримати мілісекунди (0-999)
`getTime()`	Отримати час (кількість мілісекунд, що пройшли з 1 січня 1970)
`getDay()`	Отримати номер дня тижня (0-6)

Метод `getTime()` повертає кількість мілісекунд, що пройшли з 1 січня 1970: Додати

```
var d = new Date();
document.getElementById("demo").innerHTML = d.getTime();
```

Метод `getFullYear()` повертає рік в форматі чотирьох цифр:

```
var d = new Date();
document.getElementById("demo").innerHTML = d.getFullYear();
```

Метод `getMonth()` повертає номер місяця (0-11):

```
var d = new Date();
document.getElementById("demo").innerHTML = d.getMonth();
```

В JavaScript перший місяць (січень) має номер 0, таким чином у останнього, грудня, буде номер 11.

Щоб поверталося назва місяця, можна використовувати масив з іменами місяців і метод `getMonth()`:

```
var d = new Date();
var months = ["Январь", "Февраль", "Март", "Апрель", "Май", "Червень",
  "Липень", "Август", "Вересень", "Жовтень", «Листопад», "Грудень"];
document.getElementById("demo").innerHTML = months[d.getMonth()];
```

Метод `getDate()` повертає число місяця (1-31):

```
var d = new Date();
document.getElementById("demo").innerHTML = d.getDate();
```

Метод `getHours()` повертає годину (0-23):

```
var d = new Date();
document.getElementById("demo").innerHTML = d.getHours();
```

Метод `getMinutes()` повертає хвилини (0-59):

```
var d = new Date();
document.getElementById("demo").innerHTML = d.getMinutes();
```

Метод `getSeconds()` повертає секунди (0-59):

```
var d = new Date();
document.getElementById("demo").innerHTML = d.getSeconds();
```

Метод `getMilliseconds()` повертає мілісекунди (0-999):

```
var d = new Date();
document.getElementById("demo").innerHTML = d.getMilliseconds();
```

Метод `getDay()` повертає номер дня тижня (0-6):

```
var d = new Date();
document.getElementById("demo").innerHTML = d.getDay();
```

В JavaScript першим днем тижня (0) вважається "Неділя", навіть якщо в країні, де знаходиться користувач, першим днем тижня вважається "Понеділок". Щоб поверталося назва дня тижня, можна використовувати масив з днями тижня і метод `getDay()`:

```
var d = new Date();
var days = [ "Воскресенье", "Понедельник", "Вівторок", "Середа",
   "Четвер", "П'ятниця", "Субота"];
document.getElementById("demo").innerHTML = days[d.getDay()];
```

Методи дати по UTC

Крім звичайних методів є також аналогічні методи, але які повертають дані по UTC (Universal Time Zone):

Метод	**Опис**
`getUTCDate()`	Те ж, що `getDate()`
`getUTCDay()`	Те ж, що `getDay()`
`getUTCFullYear()`	Те ж, що `getFullYear()`
`getUTCHours()`	Те ж, що `getHours()`
`getUTCMilliseconds()`	Те ж, що `getMilliseconds()`
`getUTCMinutes()`	Те ж, що `getMinutes()`
`getUTCMonth()`	Те ж, що `getMonth()`
`getUTCSeconds()`	Те ж, що `getSeconds()`

Методи для зміни даних об'єкта Date

Крім отримання даних в об'єкті `Date` також є методи, які дозволяють змінити дату й час (рік, місяць, день, годину, хвилини, секунди, мілісекунди).

Метод	**Опис**
`setDate()`	Встановлює день (1-31)
`setFullYear()`	Встановлює рік (також можна встановити місяць і день)
`setHours()`	Встановлює годину (0-23)
`setMilliseconds()`	Встановлює мілісекунди (0-999)
`setMinutes()`	Встановлює хвилини (0-59)
`setMonth()`	Встановлює місяць (0-11)
`setSeconds()`	Встановлює секунди (0-59)
`setTime()`	Встановлює час (кількість мілісекунд, що пройшли з 1 січня 1970)

Метод `setFullYear()` встановлює в об'єкті дати рік. У наступному прикладі встановлюється 2020 рік:

```
var d = new Date();
d.setFullYear(2020);
document.getElementById("demo").innerHTML = d;
```

При бажанні за допомогою методу `setFullYear()` також можна встановити місяць і день:

```
var d = new Date();
d.setFullYear(2020 року, 11, 3);
document.getElementById("demo").innerHTML = d;
```

Метод `setMonth()` встановлює в об'єкті дати місяць (0-11):

```
var d = new Date();
d.setMonth(11);
document.getElementById("demo").innerHTML = d;
```

Метод `setDate()` встановлює в об'єкті дати день (1-31):

```
var d = new Date();
d.setDate(20);
document.getElementById("demo").innerHTML = d;
```

Крім цього, метод `setDate ()` може використовуватися для додавання днів до дати:

```
var d = new Date();
d.setDate(d.getDate() + 50);
document.getElementById("demo").innerHTML = d;
```

Якщо при додаванні днів відбувається зсув місяців або року, то ці зміни в об'єкті `Date` відбуваються автоматично.

Метод `setHours()` встановлює в об'єкті дати годину (0-23):

```
var d = new Date();
d.setHours(22);
document.getElementById("demo").innerHTML = d;
```

Метод `setMinutes()` встановлює в об'єкті дати хвилини (0-59):

```
var d = new Date();
d.setMinutes(30);
document.getElementById("demo").innerHTML = d;
```

Метод `setSeconds()` встановлює в об'єкті дати секунди (0-59):

```
var d = new Date();
d.setSeconds(30);
document.getElementById("demo").innerHTML = d;
```

Порівняння дат

В JavaScript дати легко порівнюються. У наступному прикладі поточна дата порівнюється з 14 січня 2100 року:

```
var today, someday, text;
today = new Date();
someday = new Date();
someday.setFullYear(2100, 0, 14);
if (someday> today) {
    text = "14 січня 2100 року ще не настав.";
} Else {
    text = "14 січня 2100 роки вже минуло.";
}
document.getElementById("demo").innerHTML = text;
```

Увага! JavaScript рахує місяці від 0 до 11. Січень – 0. Грудень – 11.

Питання для самоперевірки:

1. Пояснити поняття функції JavaScript. Навести приклади.
2. Пояснити механізм виклику функції та повернення результату.
3. Пояснити поняття об’єкти JavaScript. Навести приклади.
4. Які властивості та методи об’єкти JavaScript ви можете назвати? Наведіть приклади.
5. Дати визначення поняття об’єкти String. Навести приклади.
6. Назвати та надати опис основних методів об’єкту String. Навести приклади.
7. Надати коротку характеристику об’єкту Date. Навести приклад.
8. Які формати дати присутні в JavaScript?
9. Назвати та надати опис основних методів об’єкту Date. Навести приклади.

МАСИВИ JAVASCRIPT

Масиви (Об’єкт Array)

Масиви JavaScript використовуються для зберігання безлічі значень в одній змінної.

```
var cars = ["Saab", "Volvo", "BMW"];
```

Масив – це особлива змінна, яка може зберігати більше одного значення за раз. Якщо у вас є список деяких значень (наприклад, список марок автомобілів), то збереження їх в окремих змінних буде виглядати наступним чином:

```
var car1 = "Saab";
var car2 = "Volvo";
var car3 = "BMW";
```

Все начебто нормально, але що якщо вам потрібно пройтися в циклі по маркам автомобілів і знайти якусь конкретну марку? І при цьому у вас є не 3, а 300 автомобілів? У цьому випадку вам допоможе масив!

Масив дозволяє зберігати безліч значень під одним ім'ям, і ви можете отримати доступ до значення за його індексом.

Створення масиву

Найпростіший спосіб створити масив в JavaScript це визначити змінну – масив, присвоївши їй потрібні значення в вигляді константи – масиву:

```
var імя_масиву = [елемент1, елемент2, ...];
```

Приклад:

```
var cars = ["Saab", "Volvo", "BMW"];
```

Пропуски і новий рядок не мають значення. Декларація масиву може займати і кілька рядків:

```
var cars = [
   "Saab",
   "Volvo",
   "BMW"
];
```

Увага! Кома після останнього елемента (наприклад, "BMW",) в різних браузерах працює по різному. Наприклад, в IE 8 і більше ранніх версіях це призведе до помилки.

У наступному прикладі також створюється масив і присвоюються значення:

```
var cars = new Array("Saab", "Volvo", "BMW");
```

Обидва приклади роблять одне і те ж. Зазвичай використовувати конструкцію new Array() немає необхідності. Для простоти, читабельності і більшої швидкості виконання скрипта краще використовувати перший спосіб створення масиву (за допомогою константи – масиву).

Доступ до елементів масиву

Щоб отримати доступ до елементу масиву, необхідно звернутися до нього по його індексу.

У наступному виразі витягується значення першого елемента масиву `cars`:

```
var name = cars[0];
```

У наступному виразі змінюється перший елемент масиву `cars`:

```
cars [0] = "Opel";
```

Приклад:

```
var cars = ["Saab", "Volvo", "BMW"];
document.getElementById("demo").innerHTML = cars[0];
```

Зверніть увагу, що перший елемент масиву має індекс [0], другий [1] і т.д. Індексація масивів завжди починається з 0. Крім цього JavaScript допускає використання всього масиву, звернувшись до нього по його імені:

```
var cars = ["Saab", "Volvo", "BMW"];
document.getElementById("demo").innerHTML = cars;
```

Масиви – це особливий тип об'єктів. Оператор `typeof` для масивів повертає тип `"object"`. Однак JavaScript масиви краще описувати як масиви. Масиви для доступу до "елементів" використовують цифрові номера. У наступному прикладі person [0] повертає значення `"Іван"`:

```
var person = ["Іван", "Петров", 46];
```

Об'єкти для доступу до своїх "підзмінним" використовують їх імена. У наступному прикладі `person.firstName` повертає значення `"Іван"`:

```
var person = {firstName: "Іван", lastName: "Петров", age: 46};
```

Елементами масиву можуть бути об'єкти

Масиви – особливий вид об'єктів. Завдяки цьому, масиви можуть зберігати змінні різних типів. Крім примітивних значень в масивах можуть зберігатися об'єкти, функції і інші масиви:

```
myArray[0] = Date.now;
myArray[1] = myFunction;
myArray[2] = myCars;
```

Властивості і методи масивів

Реальна сила масивів JavaScript полягає в їх вбудованих властивості і методи:

```
var x = cars.length; // Властивість length повертає кількість елементів
var y = cars.sort(); // Метод sort () сортує масив
```

Властивість масиву `length` повертає довжину масиву (кількість його елементів).

```
var fruits = ["Банан", "Апельсин", "Яблуко", "Манго"];
fruits.length; // довжина масиву fruits дорівнює 4
```

Увага! Значення, що повертається властивістю length, завжди на одиницю більше, ніж найбільший індекс у масиві.

Звернення до першого елементу массиву. Приклад:

```
fruits = ["Banana", "Orange", "Apple", "Mango"];
var first = fruits[0];
```

Звернення до останнього елемента массиву. Приклад:

```
fruits = ["Banana", "Orange", "Apple", "Mango"];
var last = fruits[fruits.length – 1];
```

Обхід елементів массиву

Кращий спосіб обійти всі елементи масиву, це скористатися оператором циклу `for`:

```
var fruits, text, fLen, i;

fruits = ["Банан", "Апельсин", "Яблуко", "Манго"];
fLen = fruits.length;
text = "<ul>";
for (i = 0; i <fLen; i ++) {
  text + = "<li>" + fruits[i] + "</li>";
}
```

Також, можна скористатися функцією `Array.forEach ()`:

```
var fruits, text;
fruits = ["Banana", "Orange", "Apple", "Mango"];

text = "<ul>";
fruits.forEach(myFunction);
text + = "</ ul>";

function myFunction(value) {
 text + = "<li>" + value + "</li>";
}
```

Додавання елементів в масив

Найпростіший спосіб додати новий елемент в масив це скористатися методом `push`:

```
var fruits = ["Банан", "Апельсин", "Яблуко", "Манго"];
fruits.push("Лимон"); // додамо новий елемент в масив fruits
```

Також, новий елемент можна додати за допомогою властивості `length`:

```
var fruits = ["Банан", "Апельсин", "Яблуко", "Манго"];
fruits[fruits.length] = "Лимон"; // додамо новий елемент в масив fruits
```

УВАГА! Додавання елементів з великими індексами може створити в масиві "дірки" зі значенням undefined.

У наступному прикладі в масиві fruits будуть створені "дірки" за індексами [4] і [5]:

```
var fruits = ["Банан", "Апельсин", "Яблуко", "Манго"];
```

```
fruits [6] = "Лимон"; // додамо новий елемент в масив fruits
```

Асоціативні масиви

У багатьох мовах програмування існує особливий тип масивів, в яких в якості індексів використовуються імена. Такі масиви називаються асоціативними. JavaScript не підтримує асоціативні масиви. В JavaScript масиви завжди використовують цифрові індекси.

```
var person = [];
person[0] = "Іван";
person[1] = "Петров";
person[2] = 46;
var x = person.length; // person.length поверне 3
var y = person[0]; // person[0] поверне "Іван"
```

УВАГА!! Якщо ви як індекси вкажете імена, то JavaScript перевизначити такий масив в стандартний об'єкт. Після цього властивості і методи масиву повертатимуть невірний результат.

```
var person = [];
person["firstName"] = "Іван";
person["lastName"] = "Петров";
person["age"] = 46;
var x = person.length; // person.length поверне 0
var y = person[0]; // person[0] поверне undefined
```

Різниця між масивами і об'єктами

В JavaScript масиви для доступу до елементів використовують цифрові індекси. Об'єкти використовують індекси– імена. Таким чином, масиви – це особливий тип об'єктів з цифровою індексацією елементів.

Якщо вам потрібно, щоб імена елементів були рядками, то використовуйте об'єкти.

Якщо вам потрібно, щоб імена елементів були цифрами, то використовуйте масиви.

Уникайте конструкції `new Array()`. Немає ніякої необхідності для створення масиву використовувати вбудований JavaScript конструктор масивів `new Array()`. Замість цього використовуйте оператор `[]`.

У наступному прикладі два вирази створюють новий порожній масив з ім'ям `points`:

```
var points = new Array(); // Погано
var points = []; // Добре
```

Ключове слово `new` тільки ускладнює код. Також, воно може привести до несподіваних результатів:

```
var points = new Array(40, 100); // Створюється масив з двома елементами (40 і 100)
```

Але що якщо з декларації прибрати всього один елемент:

```
var points = new Array (40); // Створюється масив з 40 елементами типу undefined !!!!!
```

Зазвичай виникає питання: Як я дізнаюся, що змінна є масивом? Проблема полягає в тому, що в JavaScript для масивів оператор `typeof` повертає тип "object":

```
var fruits = ["Банан", "Апельсин", "Яблуко", "Манго"];
typeof fruits; // поверне object
```

Оператор `typeof` повертає тип "object", тому що в JavaScript масив це об'єкт. Як же тоді бути?

Рішення №1: Щоб вирішити цю проблему в ECMAScript 5 визначається новий метод `Array.isArray()`:

```
Array.isArray (fruits); // поверне true
```

Але тут виникає інша проблема: ECMAScript 5 Не підтримується в старих браузерах.

Рішення №2: Можна визначити таку власну функцію `isArray()`:

```
function isArray (x) {
  return x.constructor.toString (). indexOf ( "Array")> – 1;
}
```

Ця функція завжди повертає true, якщо в її параметрі переданий масив. Вірніше, вона повертає true, якщо в прототипі об'єкта є слово "Array".

Рішення №3: Оператор `instanceof` повертає true, якщо об'єкт був створений за допомогою заданого конструктора:

```
var fruits = ["Банан", "Апельсин", "Яблуко", "Манго"];
fruits instanceof Array // поверне true
```

Методи Об'єкта Array

Реальна сила масивів JavaScript полягає в їх вбудованих властивості і методи. Завдяки вбудованим методам об'єкта Array, з масивами JavaScript можна робити найрізноманітніші речі без особливих зусиль і ресурсозатрат.

Перетворення масиву в рядок

JavaScript метод `toString()` перетворює масив в рядок із значень елементів масиву, розділених комами.

```
var fruits = ["Банан", "Апельсин", "Яблуко", "Манго"];
document.getElementById("demo").innerHTML = fruits.toString();
```

В результаті буде виведений рядок `"Банан, Апельсин, Яблуко, Манго"`.

Метод `join()` також об'єднує всі елементи масиву в один рядок. Він діє як метод `toString()`, але при цьому дозволяє вказати роздільник:

```
var fruits = ["Банан", "Апельсин", "Яблуко", "Манго"];
document.getElementById("demo").innerHTML = fruits.join("*");
```

В результаті буде виведений рядок `"Банан*Апельсин*Яблуко*Манго"`.

Видалення і додавання елементів

Коли ви працюєте з масивами, вам часто доводиться видаляти і додавати елементи. За допомогою спеціальних методів це можна зробити легко і безпечно.

Метод `pop()` видаляє останній елемент масиву:

```
var fruits = ["Банан", "Апельсин", "Яблуко", "Манго"];
fruits.pop(); // видаляє останній елемент ("Манго") масиву fruits
```

При цьому метод `pop()` повертає значення видаленого елемента:

```
var fruits = [ "Банан", "Апельсин", "Яблуко", "Манго"];
var x = fruits.pop(); // в змінної x буде рядок "Манго"
```

Метод `push()` додає новий елемент в кінець масиву:

```
var fruits = ["Банан", "Апельсин", "Яблуко", "Манго"];
fruits.push("Ківі"); // додає новий елемент ("Ківі") в кінець масиву fruits
```

При цьому метод push() повертає нову довжину масиву:

```
var fruits = ["Банан", "Апельсин", "Яблуко", "Манго"];
var x = fruits.push("Ківі"); // в змінної x буде 5
```

Метод shift() видаляє перший елемент масиву і зрушує залишилися елементи до менших індексів:

```
var fruits = ["Банан", "Апельсин", "Яблуко", "Манго"];
fruits.shift(); // видаляє перший елемент "Банан" з масиву fruits
```

При цьому метод shift() повертає значення видаленого елемента:

```
var fruits = ["Банан", "Апельсин", "Яблуко", "Манго"];
var x = fruits.shift(); // в змінної x буде рядок "Банан"
```

Метод unshift() додає новий елемент в початок масиву і пересуває інші елементи:

```
var fruits = ["Банан", "Апельсин", "Яблуко", "Манго"];
fruits.unshift("Лимон"); // додає новий елемент "Лимон" в масив fruits
```

При цьому метод unshift() повертає нову довжину масиву:

```
var fruits = ["Банан", "Апельсин", "Яблуко", "Манго"];
var x = fruits.unshift("Лимон"); // в змінної x буде 5
```

Зміна значень елементів

Доступ до значень елементів масиву здійснюється за номером індексу елемента. Нумерація індексів починається з 0. Індекс першого елемента масиву [0], другого – [1], третього – [2] і т. Д.

```
var fruits = ["Банан", "Апельсин", "Яблуко", "Манго"];
fruits[0] = "Ківі"; // змінить значення першого елемента масиву fruits на "Ківі"
```

Властивість length дозволяє додавати нові елементи в кінець масиву:

```
var fruits = ["Банан", "Апельсин", "Яблуко", "Манго"];
fruits[fruits.length] = "Ківі"; // додає новий елемент в масив fruit
```

Стирання елементів

У зв'язку з тим, що масиви в JavaScript є об'єктами, існує ще один спосіб видалення елементів масиву – за допомогою оператора `delete`:

```
var fruits = ["Банан", "Апельсин", "Яблуко", "Манго"];
delete fruits[0]; // значення першого елемента масиву fruits зміниться на undefined
```

Увага! Використання оператора delete може залишити в масиві невраховуваних "діри" з значень undefined. Для видалення елементів масиву краще використовувати метод pop () або shift ().

Додавання групи елементів

Метод `splice()` використовується для додавання в масив групи нових елементів:

```
var fruits = ["Банан", "Апельсин", "Яблуко", "Манго"];
fruits.splice(2, 0, "Лимон", "Ківі");
```

Перший параметр (2) визначає позицію, куди вставляються нові елементи. Другий параметр (0) визначає, скільки елементів має бути видалено. Інші параметри (`"Лимон", "Ківі"`) визначають які вставляються елементи.

Метод `splice()` також можна використовувати для видалення елементів з масиву. Робить він це безпечно і чисто, не залишаючи в масиві "дірок":

```
var fruits = [ "Банан", "Апельсин", "Яблуко", "Манго"];
fruits.splice (0, 1); // видаляє перший елемент масиву fruits
```

Перший параметр (0) визначає позицію, куди вставляються нові елементи. Другий параметр (1) визначає, скільки елементів має бути видалено. Інші параметри не задані. Нові елементи не вставляються.

Злиття масивів

Метод `concat()` створює новий масив шляхом злиття (об'єднання) існуючих масивів.

Приклад злиття двох масивів:

```
var myGirls = ["Cecilie", "Lone"];
var myBoys = ["Emil", "Tobias", "Linus"];
var myChildren = myGirls.concat(myBoys); // Злиття масивів myGirls і myBoys
```

Метод `concat()` не змінює існуючі масиви. Він створює новий масив. Метод `concat()` в параметрах може приймати будь– яке число масивів.

Приклад злиття трьох масивів:

```
var arr1 = ["Cecilie", "Lone"];
var arr2 = ["Emil", "Tobias", "Linus"];
var arr3 = ["Robin", "Morgan"];
var myChildren = arr1.concat(arr2, arr3); // Масив arr1 об'єднується з arr2 і arr3
```

Як параметри метод `concat()` може приймати як змінні, так і значення.

Приклад злиття масиву зі значенням:

```
var arr1 = ["Cecilie", "Lone"];
var myChildren = arr1.concat(["Emil", "Tobias", "Linus"]);
```

Витяг частини масиву

Метод `slice()` дозволяє витягти (вирізати) частина масиву в новий масив.

У наступному прикладі вилучається частина масив, починаючи з елемента з індексом 1 ("Апельсин"):

```
var fruits = ["Банан", "Апельсин", "Лимон", "Яблуко", "Манго"];
var citrus = fruits.slice(1);
```

Метод `slice()` створює новий масив. У вихідному масиві ніякі елементи не видаляються.

У наступному прикладі вилучається частина масив, починаючи з елемента з індексом 3 ("Яблуко"):

```
var fruits = [ "Банан", "Апельсин", "Лимон", "Яблуко", "Манго"];
var citrus = fruits.slice (3);
```

Метод `slice()` приймає два параметри. Наприклад, `slice (1, 3)`. У цьому випадку метод `slice()` вибирає елементи, починаючи з елемента з індексом, зазначеним в першому параметрі, і закінчуючи (але не включаючи) елементом з індексом, зазначеним у другому параметрі. Якщо другий параметр не заданий, то витягуються всі елементи до кінця масиву.

Сортування масивів

Крім додавання і видалення елементів масиву, часто виникає необхідність впорядкувати їх в певному порядку. Для цих випадків також існує цілий набір спеціальних методів.

Сортування масиву

Метод `sort()` сортує масив у алфавітному порядку:

```
var fruits = [ "Банан", "Апельсин", "Яблуко", "Манго"];
fruits.sort(); // Сортує елементи масиву fruits
```

Реверс масиву

Метод `reverse()` змінює порядок елементів масиву на зворотний. Цей метод можна використовувати для сортування масиву в зворотному порядку:

```
var fruits = ["Банан", "Апельсин", "Яблуко", "Манго"];
fruits.sort(); // Сортує елементи масиву fruits
fruits.reverse(); // Розгортає елементи в зворотному порядку
```

Числове сортування

За замовчуванням, метод `sort()` сортує значення елементів масиву як рядки.

Це добре працює зі рядковими значеннями ("Апельсин" встане перед "Банан"). Однак, якщо сортувати числа як рядки, то "25" буде більше "100", тому що рядок "2" більше рядка "1".

Таким чином, метод `sort()` дасть невірний результат при сортуванні числових значень. Вирішити цю проблему можна за допомогою функції порівняння:

```
var points = [40, 100, 1, 5, 25, 10];
points.sort(function (a, b) {return a – b});
```

Цей же трюк можна використовувати для сортування числових значень в зворотному порядку:

```
var points = [40, 100, 1, 5, 25, 10];
points.sort(function (a, b) {return b – a});
```

Мета функції порівняння – визначити альтернативний порядок сортування. Функція порівняння в залежності від параметрів повинна повертати від'ємне значення, нуль або позитивне значення:

```
function(a, b) {
  return a– b
}
```

Коли метод `sort()` порівнює два значення, він передає ці значення в функцію порівняння і потім сортує їх відповідно до повернутих (негативним, нульовим, позитивним) значенням. Порівнюючи значення 40 і 100, метод `sort()` викликає функцію

порівняння `function (40,100)`. Функція обчислює вираз 40 – 100 і повертає – 60 (від'ємне значення). В результаті метод sort() визначає 40 як значення менше 100.

Пошук найбільшого / найменшого значення в масиві

У масиві немає вбудованих функцій для пошуку максимального або мінімального значення. Проте, після того як ви відсортували масив, ви можете отримати найбільше і найменше значення. Однак, сортування всього масиву тільки заради того, щоб знайти найбільше (або найменше) значення, є вкрай неефективним способом зробити це.

Для пошуку найбільшого числа в масиві можна скористатися методом `Math.max.apply`:

```
function myArrayMax (arr) {
   return Math.max.apply (null, arr);
}
```

`Math.max.apply([1, 2, 3])` еквівалентно `Math.max(1, 2, 3)`.

Для пошуку найменшого числа в масиві можна скористатися методом `Math.min.apply`:

```
function myArrayMin (arr) {
   return Math.min.apply (null, arr);
}
```

`Math.min.apply([1, 2, 3])` еквівалентно `Math.min(1, 2, 3)`.

Найшвидшим рішенням є використання "саморобної" функції.

У наступній функції в циклі проходимо по масиву, порівнюючи кожне значення, поки не буде знайдено найбільше.

Приклад пошук максимального значення:

```
function myArrayMax(arr) {
  var len = arr.length
  var max = - Infinity;
  while (len- - ) {
    if (arr [len] > max) {
      max = arr[len];
    }
  }
  return max;
}
```

У наступній функції в циклі проходимо по масиву, порівнюючи кожне значення, поки не буде знайдено найменше.

Приклад пошук мінімального значення:

```
function myArrayMin(arr) {
  var len = arr.length
  var min = Infinity;
  while (len– – ) {
    if (arr [len] < min) {
      mim = arr[len];
    }
  }
  return min;
}
```

Сортування масиву об'єктів

Часто масиви JavaScript містять об'єкти:

```
var cars = [
 {Type: "Volvo", year: 2016},
 {Type: "Saab", year: 2001},
 {Type: "BMW", year: 2010}
];
```

Навіть якщо в об'єктах є властивості з різними типами даних, метод `sort()` все одно може використовуватися для сортування такого масиву.

Рішення полягає в написанні функції порівняння для значень властивостей:

```
cars.sort (function (a, b) {return a.year – b.year});
```

Порівняння властивостей строкового типу буде трохи більш складним:

```
cars.sort (function (a, b) {
   var x = a.type.toLowerCase ();
   var y = b.type.toLowerCase ();
   if (x > y) {return 1;}
   return 0;
});
```

Методи перебирання масиву

Методи перебирання масиву взаємодіють з кожним елементом масиву.

Метод `forEach()` один раз викликає функцію зворотного виклику для кожного елемента масиву.

```
var txt = "";
var numbers = [45, 4, 9, 16, 25];
numbers.forEach (myFunction);

function myFunction (value, index, array) {
  txt = txt + value + "<br>";
}
```

Зверніть увагу, що функція зворотного виклику приймає 3 параметра:

- значення елемента (`value`);
- індекс елемента (`index`);
- сам масив (`array`).

У попередньому прикладі у функції зворотного виклику встановлюється параметр із значенням елемента `value`. Цей приклад можна переписати таким чином:

```
var txt = "";
var numbers = [45, 4, 9, 16, 25];
numbers.forEach (myFunction);

function myFunction (value) {
  txt = txt + value + "<br>";
}
```

Метод `Array.forEach()` підтримується всіма браузерами за винятком Internet Explorer 8 і більше ранніх версій.

Метод map() створює новий масив, попередньо виконавши функцію зворотного виклику з кожним елементом вихідного масиву. Метод `map()` не виконує функцію зворотного виклику, якщо у елементів масиву немає значень. Метод `map()` не змінює оригінальний масив.

У наступному прикладі створюється новий масив, значеннями елементів якого будуть значення елементів вихідного масиву помножені на 2:

```
var numbers1 = [45, 4, 9, 16, 25];
var numbers2 = numbers1.map(myFunction);

function myFunction(value, index, array) {
  return value * 2;
}
```

Функція зворотного виклику приймає 3 параметра:

- значення елемента (value);
- індекс елемента (index);
- сам масив (array).

Якщо у функції зворотного виклику використовується тільки перший параметр value, то параметри index і array можуть бути опущені:

```
var numbers1 = [45, 4, 9, 16, 25];
var numbers2 = numbers1.map(myFunction);

function myFunction(value) {
  return value * 2;
}
```

Метод Array.map() підтримується всіма браузерами за винятком Internet Explorer 8 і більше ранніх версій.

Метод filter() створює новий масив з елементами вихідного масиву, що пройшли задану перевірку.

У наступному прикладі створюється новий масив з елементів вихідного масиву значення, яких більше 18:

```
var numbers = [45, 4, 9, 16, 25];
var over18 = numbers.filter(myFunction);

function myFunction(value, index, array) {
 return value> 18;
}
```

Функція зворотного виклику приймає 3 параметра:

- значення елемента (value);
- індекс елемента (index);
- сам масив (array).

У попередньому прикладі у функції зворотного виклику не використовуються параметри index і array, тому їх можна опустити:

```
var numbers = [45, 4, 9, 16, 25];
var over18 = numbers.filter(myFunction);
```

```
function myFunction(value) {
 return value> 18;
}
```

Метод `Array.filter()` підтримується всіма браузерами за винятком Internet Explorer 8 і більше ранніх версій.

Метод `reduce()` / `reduceRight()` виконує функцію зворотного виклику з кожним елементом масиву для обчислення єдиного значення (відомості до єдиного значення). Метод `reduce()` обробляє елементи масиву зліва направо. Метод `reduceRight()` обробляє елементи масиву справа наліво. `Метод reduce()` / `reduceRight()` не впливає на вихідний масив.

У наступному прикладі обчислюється сума всіх чисел в масиві:

```
var numbers1 = [45, 4, 9, 16, 25];
var sum = numbers1.reduce(myFunction);

function myFunction(total, value, index, array) {
 return total + value;
}
```

Зверніть увагу, що функція зворотного виклику приймає 4 параметра:

- початкове / раніше повернене значення (`total`);
- значення елемента (`value`);
- індекс елемента (`index`);
- сам масив (`array`).

У попередньому прикладі у функції зворотного виклику не використовуються параметри index і array, тому його код можна переписати таким чином:

```
var numbers1 = [45, 4, 9, 16, 25];
var sum = numbers1.reduce(myFunction);

function myFunction(total, value) {
 return total + value;
}
```

Метод `reduce()` / `reduceRight()` може приймати початкове значення:

```
var numbers1 = [45, 4, 9, 16, 25];
var sum = numbers1.reduce(myFunction, 100);
function myFunction(total, value) {
 return total + value;
}
```

Метод `Array.reduce()` / `Array.reduceRight()` підтримується всіма браузерами за винятком Internet Explorer 8 і більше ранніх версій.

Метод `every()` перевіряє, виконують задану умову всі елементи масиву.

У наступному прикладі перевіряється, чи більше 18 значення всіх елементів масиву:

```
var numbers = [45, 4, 9, 16, 25];
var allOver18 = numbers.every(myFunction);

function myFunction(value, index, array) {
  return value> 18;
}
```

Функція зворотного виклику приймає 3 параметра:

- значення елемента (`value`);
- індекс елемента (`index`);
- сам масив (`array`).

Якщо функція зворотного виклику використовує тільки перший параметр (`value`), то інші параметри можна опустити:

```
var numbers = [45, 4, 9, 16, 25];
var allOver18 = numbers.every(myFunction);

function myFunction(value) {
 return value> 18;
}
```

Метод `Array.every()` підтримується всіма браузерами за винятком Internet Explorer 8 і більше ранніх версій.

Метод `some()` перевіряє, чи виконує заданий умова хоча б один елемент масиву.

У наступному прикладі перевіряється, чи є в масиві хоча б один елемент зі значенням більше 18:

```
var numbers = [45, 4, 9, 16, 25];
var someOver18 = numbers.some(myFunction);

function myFunction(value, index, array) {
  return value> 18;
}
```

Функція зворотного виклику приймає 3 параметра:

- значення елемента (value);
- індекс елемента (index);
- сам масив (array).

Метод Array.some() підтримується всіма браузерами за винятком Internet Explorer 8 і більше ранніх версій.

Метод indexOf() шукає в масиві елемент із заданим значенням і повертає його індекс.

Увага! Перший елемент матиме індекс 0, другий – 1 і т.д.

У наступному прикладі шукаємо елемент зі значенням "Apple":

```
var fruits = ["Apple", "Orange", "Apple", "Mango"];
var a = fruits.indexOf( "Apple");
```

Метод Array.indexOf () підтримується всіма браузерами за винятком Internet Explorer 8 і більше ранніх версій.

Синтаксис: масив.indexOf(елемент, початок)

- елемент – обов'язковий параметр, елемент для пошуку.
- початок – необов'язковий параметр, позиція для початку пошуку. Якщо вказано від'ємне значення, то пошук почнеться з позиції, відрахувавши з кінця масиву, і продовжиться до кінця масиву.

Якщо елемент не знайдений, то метод Array.indexOf () поверне – 1. Якщо в масиві кілька елементів із заданим значенням, то буде повернуто індекс першого знайденого елемента.

Метод Array.lastIndexOf() аналогічний методу Array.indexOf(), але він починає пошук з кінця масиву і веде його до початку масиву.

У наступному прикладі шукаємо елемент зі значенням "Apple":

```
var fruits = ["Apple", "Orange", "Apple", "Mango"];
var a = fruits.lastIndexOf( "Apple");
```

Метод Array.lastIndexOf() підтримується всіма браузерами за винятком Internet Explorer 8 і більше ранніх версій.

Синтаксис: масив.lastIndexOf(елемент, початок)

- елемент – обов'язковий параметр, елемент для пошуку.
- початок – необов'язковий параметр, позиція для початку пошуку. Якщо вказано від'ємне значення, то пошук почнеться з позиції, відрахувавши з кінця масиву, і продовжиться до початку масиву.

Метод find() повертає значення першого елемента масиву, що пройшов задану перевірку в функції зворотного виклику.

У наступному прикладі відбувається пошук (і повернення значення) першого елемента, значення якого більше 18:

```
var numbers = [4, 9, 16, 25, 29];
var first = numbers.find(myFunction);

function myFunction(value, index, array) {
 return value> 18;
}
```

Функція зворотного виклику приймає 3 параметра:

- значення елемента (`value`);
- індекс елемента (`index`);
- сам масив (`array`).

Метод `Array.find()` не підтримується в старих браузерах.

Метод `findIndex()` повертає індекс першого елемента масиву, що пройшов задану перевірку в функції зворотного виклику.

У наступному прикладі повертається індекс першого елемента, значення якого більше 18:

```
var numbers = [4, 9, 16, 25, 29];
var first = numbers.findIndex(myFunction);

function myFunction(value, index, array) {
  return value> 18;
}
```

Функція зворотного виклику приймає 3 параметра:

- значення елемента (`value`);
- індекс елемента (`index`);
- сам масив (`array`).

Метод `Array.findIndex()` не підтримується в старих браузерах.

Питання для самоперевірки:

1. Описати способи створення масиву у JavaScript.
2. Навести приклад присвоєння значення конкретному елементу.
3. Пояснити, що таке асоціативні масиви.
4. Пояснити, в чому полягає різниця між масивом і об’єктом
5. Назвати та надати опис основних методів об’єктуArray. Навести приклади.
6. Надати короткий опис способів сортування масивів. Навести приклади.
7. Надати короткий опис методів перебирання масиву. Навести приклади.

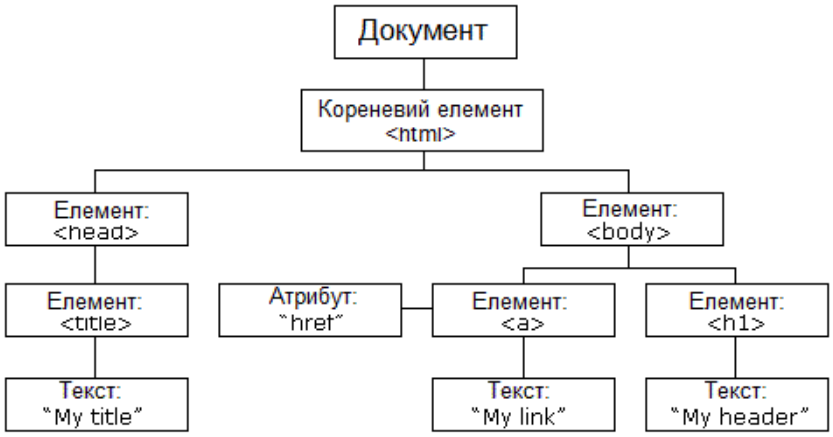
ОБ’ЄКТНА МОДЕЛЬ ДОКУМЕНТА

Завдяки HTML DOM програми на JavaScript можуть отримати доступ і змінювати всі елементи HTML документа.

HTML DOM (Document Object Model)

Після завантаження веб-сторінки браузер створює Об'єктну модель документа (анг. Document Object Model) цієї сторінки.

Модель HTML DOM створюється у вигляді дерева об'єктів:



Завдяки цій об'єктної моделі JavaScript отримує всю міць, необхідну для створення динамічного HTML:

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

DOM – це офіційний стандарт консорціуму W3C (World Wide Web Consortium). DOM визначає стандарт для доступу до документів:

"W3C об'єктна модель документа (DOM) (від англ. Document Object Model) – це незалежний від платформи і мови програмний інтерфейс, що дозволяє програмам і скриптам динамічно отримати доступ до вмісту документів, а також змінювати їх вміст, структуру і оформлення."

Стандарт W3C DOM розділений на 3 різні частини:

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

- HTML DOM – стандартна модель для HTML документів.

HTML DOM – це стандартна об'єктна модель і програмний інтерфейс для HTML документів. Він визначає:

- HTML елементи як об'єкти;
- Властивості всіх HTML елементів;
- Методи для доступу до всіх HTML елементам;
- Події для всіх HTML елементів.

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

Методи і властивості

HTML DOM методи – це дії, які ви можете виконувати (з елементами HTML).

HTML DOM властивості – це значення (елементів HTML), які ви можете встановлювати або змінювати.

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

В DOM все HTML елементи визначені як об'єкти.

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

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

Метод – це дія, що ви можете виконати (наприклад, додавання чи видалення елемента HTML).

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

```
<html>
<body>

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

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

</body>
</html>
```

У попередньому прикладі `getElementById` – це метод, а `innerHTML` – властивість.

Найчастіший спосіб отримати доступ до елементу HTML – це використовувати ідентифікатор `id` елемента.

У попередньому прикладі метод `getElementById` використовує `id = "demo"`, щоб знайти потрібний елемент.

Найпростіший спосіб прочитати вміст елемента – це скористатися властивістю `innerHTML`.

Властивість `innerHTML` корисна тим, що вона дозволяє прочитати або змінити вміст будь-якого елементу HTML, включаючи `<html>` і `<body>`.

Об'єкт документа

Об'єкт документа `document` є батьком всіх інших об'єктів на веб-сторінці. Цей об'єкт і представляє вашу веб-сторінку.

Якщо вам потрібно отримати доступ до якогось елементу на HTML сторінці, то перш вам потрібно звернутися до об'єкта document.

Нижче представлено кілька прикладів, як можна використовувати об'єкт документа для доступу і маніпулювання HTML.

Пошук елементів HTML

Метод	**Опис**
`document.getElementById (id)`	Пошук елемента за ідентифікатором `id`
`document.getElementsByTagName (ім'я)`	Пошук елемента по імені тега
`document.getElementsByClassName (ім'я)`	Пошук елемента по імені класу

Зміна елементів HTML

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

Додавання і видалення елементів

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

Додавання обробника події

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

Пошук об'єктів HTML

Властивість	Опис	DOM
`document.anchors`	Повертає всі елементи `<a>`, у яких є атрибут `name`	1
`document.applets`	Повертає всі елементи `<applet>` (Заборонено в HTML5)	1
`document.baseURI`	Повертає абсолютний базовий URI документа	3
`document.body`	Повертає елемент `<body>`	1
`document.cookie`	Повертає куки документа	1
`document.doctype`	Повертає `doctype` документа	3
`document.documentElement`	Повертає елемент `<html>`	3
`document.documentMode`	Повертає режим, використаний браузером	3
`document.documentURI`	Повертає URI документа	3
`document.domain`	Повертає доменне ім'я сервера документа	1
`document.embeds`	Повертає всі елементи `<embed>`	3
`document.forms`	Повертає всі елементи `<form>`	1
`document.head`	Повертає елемент `<head>`	3
`document.images`	Повертає всі елементи `<img>`	1
`document.implementation`	Повертає реалізацію DOM	3
`document.inputEncoding`	Повертає кодування документа (набір символів)	3
`document.lastModified`	Повертає дату і час поновлення документа	3
`document.links`	Повертає всі елементи `<area>` і `<a>`, у яких є атрибут `href`	1
`document.readyState`	Повертає статус завантаження документа	3
`document.referrer`	Повертає URL-адресу документа, який завантажував поточний документ	1
`document.scripts`	Повертає всі елементи `<script>`	3
`document.strictErrorChecking`	Повертає, чи виконується перевірка помилок чи ні	3
`document.title`	Повертає елемент `<title>`	1
`document.URL`	Повертає повний URL документа	1

Перший HTML DOM рівень 1 (1998) визначав 11 HTML об'єктів, наборів об'єктів і властивостей. Всі вони все ще актуальні в HTML5. Пізніше, в HTML DOM рівень 3 були додані нові об'єкти, набори і властивості.

Пошук елементів

Пошук HTML елементів

Часто в JavaScript необхідно проводити певні маніпуляції з HTML елементами. Щоб це зробити, спочатку потрібно знайти потрібний об'єкт. Знайти HTML елемент можна декількома способами:

- за ідентифікатором `id`;
- по імені тега;
- по імені класу;
- по селекторам CSS;
- за розділами об'єктів HTML.

Пошук HTML елемента за ідентифікатором

Найпростіший спосіб знайти HTML елемент в DOM – це використовувати його ідентифікатор `id`.

У наступному прикладі ми шукаємо елемент з `id = "intro"`:

```
var myElement = document.getElementById ( "intro");
```

Якщо елемент буде знайдений, то він буде повернутий у вигляді об'єкта (в змінну `myElement`).

Якщо елемент не буде знайдений, то в змінна `myElement` буде містити значення `null`.

Пошук HTML елемента по імені тега

У наступному прикладі ми шукаємо всі елементи `<p>`:

```
var x = document.getElementsByTagName("p");
```

У наступному прикладі спочатку відбувається пошук елемента з `id = "main"`, а потім всіх елементів `<p>` всередині `"main"`:

```
var x = document.getElementById ( "main");
var y = x.getElementsByTagName ( "p");
```

Пошук HTML елемента по імені класу

Якщо потрібно знайти всі HTML елементи з одним і тим же ім'ям класу, то використовують метод `getElementsByClassName ()`.

У наступному прикладі повертається список усіх елементів з атрибутом `class = "intro"`:

```
var x = document.getElementsByClassName ( "intro");
```

Увага! Пошук елементів по імені класу не працює в Internet Explorer 8 і більше ранніх версіях.

Пошук HTML елемента по CSS селекторів

Якщо потрібно знайти всі HTML елементи, які підходять по заданому CSS селектору (id, імена класів, типи, атрибути, значення атрибутів і т.п.), використовується метод `querySelectorAll ()`.

У наступному прикладі повертається список усіх елементів `<p>` з атрибутом `class = "intro"`:

```
var x = document.querySelectorAll("p.intro");
```

Увага! Метод querySelectorAll () не працює в Internet Explorer 8 і більше ранніх версіях.

Пошук HTML елемента за розділами HTML об'єктів

У наступному прикладі проводиться пошук елемента форми з атрибутом `id = "frm1"` в наборі об'єктів `forms`, і відображаються всі значення елементів:

```
var x = document.forms["frm1"];
var text = "";
var i;
for (i = 0; i <x.length; i ++) {
  text + = x.elements[i].value + "<br>";
}
document.getElementById("demo").innerHTML = text;
```

Також доступні наступні HTML об'єкти (і набори об'єктів): `document.anchors; document.body; document.documentElement; document.embeds; document.forms; document.head; document.images; document.links; document.scripts; document.title`.

Зміна HTML

HTML DOM дозволяє JavaScript змінювати вміст HTML елементів.

Зміна потоку виведення HTML

JavaScript може створювати динамічний HTML контент.

В JavaScript можна використовувати метод `document.write ()` для прямого запису в потік виводу HTML:

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

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

</body>
</html>
```

Увага! Ніколи не використовуйте метод document.write () після повного завантаження документа. Він перезапише вміст всього документа.

Зміна вмісту HTML елемента

Найпростіший спосіб змінити вміст HTML елемента – це скористатися властивістю innerHTML.

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

У наступному прикладі змінюється вміст елемента `<p>`:

```
<html>
<body>

<p id = "p1"> Привіт всім! </ P>

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

</body>
</html>
```

Наведений в прикладі HTML документ містить елемент `<p>` з атрибутом `id = "p1"`. Ми використовуємо HTML DOM, щоб отримати доступ до елементу з `id = "p1"`. JavaScript змінює вміст (`innerHTML`) елемента на рядок "Новий текст!"

У наступному прикладі змінюється вміст елемента `<h1>`:

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

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

<script>
var element = document.getElementById ("id01");
```

```
element.innerHTML = "Новий заголовок";
</script>

</body>
</html>
```

Наведений в прикладі HTML документ містить елемент `<h1>` з атрибутом `id = "id01"`. Ми використовуємо HTML DOM, щоб отримати доступ до елементу з `id = "id01"`. JavaScript змінює вміст (`innerHTML`) елемента на рядок `"Новий заголовок"`.

Зміна значення атрибута

HTML DOM дозволяє змінювати значення атрибутів HTML елементів.

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

У наступному прикладі змінюється значення атрибута `src` елемента `<img>`:

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

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

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

</body>
</html>
```

Наведений в прикладі HTML документ містить елемент `<img>` з атрибутом `id = "myImage"`. Ми використовуємо HTML DOM, щоб отримати доступ до елементу з `id = "myImage"`. JavaScript змінює значення атрибута `src` цього елемента з `"smiley.gif"` на `"landscape.jpg"`

DOM – зміна CSS

HTML DOM дозволяє JavaScript змінювати стиль HTML елементів.

Синтаксис: `document.getElementById(id).style.властивість = новий стиль`

У наступному прикладі змінюється стиль елемента `<p>`:

```
<html>
<body>

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

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

</body>
</html>
```

Навігація по вузлах

HTML DOM використовуючи відносини між вузлами дозволяє переміщатися по дереву вузлів.

Відповідно до стандарту HTML DOM, все в HTML документі є вузлом:

- Весь документ – вузол документа.
- Кожен HTML елемент – вузол елемента.
- Текст всередині HTML елементів – вузол тексту.
- Кожен HTML атрибут – вузол атрибута (заборонений).
- Всі коментарі – вузли коментарів.
- Вузли HTML DOM.

З HTML DOM можна отримати доступ до всіх вузлів за допомогою JavaScript. Можна створювати нові вузли, а існуючі можна змінювати або видаляти.

Відносини між вузлами

Вузли в дереві DOM мають ієрархічні відносини між один одним. Щоб описати ці відносини зазвичай використовують такі поняття як батько (предок), дитина (нащадок) і родич (споріднений елемент).

У дереві вузлів самий верхній вузол називається кореневим вузлом.

У кожного вузла, за винятком кореневого, завжди є один і тільки один предок або батько.

У будь-якого вузла може бути скільки завгодно нащадків.

Споріднені або однорівневі вузли – це вузли, у яких один і той же предок (батько).

```
<html>
 <head>
   <title> Посібник по ТММГВ </title>
 </head>
 <body>
   <h1> Розділ 1 </h1>
   <p> Що таке JavaScript </p>
 </body>
</html>
```

З HTML коду, представленого вище, ми можемо зрозуміти наступне:

<html> – кореневий вузол.

<html> не має батька.

<html> – батько вузлів <head> і <body>.

<head> – перший нащадок вузла <html>.

<body> – останній нащадок вузла <html>.

Також ми бачимо:

<head> має одного нащадка – <title>.

<title> має одного нащадка (текстовий вузол) – "Посібник по ТММГВ".

<body> має двох нащадків – <h1> і <p>.

<h1> має одного нащадка – "Розділ 1".

<p> має одного нащадка – "Що таке JavaScript".

<h1> і <p> – родинні (одного рівня) вузли.

Переміщення між вузлами

Щоб переміщатися між вузлами, використовуються наступні JavaScript властивості вузлів: parentNode; childNodes[номер_узла]; firstChild; lastChild; nextSibling; previousSibling.

Вузли-нащадки і значення вузлів

Звичайна помилка при роботі з DOM – це очікування, що вузол елемента містить текст.

Приклад:

```
<title id = "demo"> Лабораторний практикум </title>
```

У наведеному прикладі вузол елемента <title> не містить текст.

Він містить текстовий вузол із значенням "Посібник по ТММГВ".

Отримати доступ до значення текстового вузла можна за допомогою властивості вузла innerHTML:

```
var myTitle = document.getElementById ("demo").innerHTML;
```

Звернення до властивості innerHTML має той же ефект як звернення до nodeValue першого нащадка:

```
var myTitle = document.getElementById ( "demo"). firstChild.nodeValue;
```

Отримати доступ до першого нащадку також можна наступним чином:

```
var myTitle = document.getElementById("demo").childNodes[0].nodeValue;
```

У всіх трьох наступних прикладах витягується текст елемента <h1> і копіюється в елемент <p>.

Приклад 1

```
<html>
<body>

<h1 id = "id01"> Моя перша сторінка </h1>
<p id = "id02"> </p>

<script>
document.getElementById("id02").innerHTML = document.getElementById("id01").innerHTML;
</script>

</body>
</html>
```

Приклад 2

```
<html>
<body>

<h1 id = "id01"> Моя перша сторінка </h1>
<p id = "id02"> </p>

<script>
document.getElementById("id02").innerHTML =
document.getElementById("id01").firstChild.nodeValue;
</script>

</body>
</html>
```

Приклад 3

```
<html>
<body>

<h1 id = "id01"> Моя перша сторінка </h1>
<p id = "id02"> </p>
```

```
<script>
document.getElementById("id02").innerHTML =
document.getElementById("id01").childNodes[0] .nodeValue;
</script>

</body>
</html>
```

Кореневі вузли DOM

Існують два спеціальних властивості, які дозволяють отримати доступ до всього документа:

`document.body` – тіло документа

`document.documentElement` – весь документ

Приклад 1

```
<html>
<body>

<p> Привіт всім! </p>
<div>
<p> DOM дуже корисний! </p>
<p> Цей приклад демонструє властивість <b> document.body </b>. </p>
</div>

<script>
alert (document.body.innerHTML);
</script>

</body>
</html>
```

Приклад 2

```
<html>
<body>

<p> Привіт всім! </p>
<div>
<p> DOM дуже корисний! </p>
<p> Цей приклад демонструє властивість <b> document.documentElement </b>. </p>
</div>
```

```
<script>
alert (document.documentElement.innerHTML);
</script>

</body>
</html>
```

Властивість nodeName

Властивість `nodeName` визначає ім'я вузла.

- `nodeName` – властивість тільки для читання
- `nodeName вузла елемента` – це те ж саме, що і ім'я тега
- `nodeName вузла атрибуту` – це ім'я атрибута
- `nodeName текстового вузла` – це завжди #text
- `nodeName вузла документа` – завжди #document

Приклад:

```
<h1 id = "id01"> Моя перша сторінка </h1>
<p id = "id02"> </p>

<script>
document.getElementById("id02").innerHTML = document.getElementById("id01").nodeName;
</script>
```

Увага! Властивість nodeName завжди містить ім'я тега HTML елемента у верхньому регістрі.

Властивість nodeValue

Властивість `nodeValue` визначає значення вузла.

- `nodeValue` для вузлів елементів має значення null
- `nodeValue` для текстових уз
- `nodeValue` для текстових вузлів містить сам текст
- `nodeValue` для вузлів атрибутів містить значення атрибута

Властивість nodeType

Властивість `nodeType` має статус "тільки для читання". Воно повертає тип вузла.

Приклад

```
<h1 id = "id01"> Моя перша сторінка </h1>
<p id = "id02"> </p>
```

```
<script>
document.getElementById("id02").innerHTML = document.getElementById("id01").nodeType;
</script>
```

Найбільш важливі властивості nodeType

Вузол	Тип	Приклад
ELEMENT_NODE	1	<h1 class = "heading"> MSiter </ h1>
TEXT_NODE	3	MSiter
COMMENT_NODE	8	<! – Це коментар ->
DOCUMENT_NODE	9	Сам HTML документ (батько <html>)
DOCUMENT_TYPE_NODE	10	<! Doctype html>

Робота з елементами (вузлами)

Створення нових HTML елементів (вузлів)

Щоб додати новий елемент в дерево HTML DOM, спочатку потрібно його створити, а потім приєднати його до існуючого елементу.

Приклад:

```
<div id = "div1">
<p id = "p1"> Це параграф. </p>
<p id = "p2"> Це інший параграф. </p>
</div>

<script>
var para = document.createElement ("p");
var node = document.createTextNode ("Це новий параграф.");
para.appendChild (node);

var element = document.getElementById ("div1");
element.appendChild (para);
</script>
```

Спочатку створюється новий елемент (вузол елемента) <p>:

```
var para = document.createElement ("p");
```

Щоб додати текст в елемент <p>, спочатку потрібно створити текстовий вузол.

```
var node = document.createTextNode ("Це новий параграф.");
```

Потім необхідно приєднати текстовий вузол до елементу <p>:

```
para.appendChild (node);
```

І, нарешті, потрібно приєднати новий елемент до існуючого. Знаходимо необхідний існуючий елемент:

```
var element = document.getElementById ("div1");
```

І приєднуємо до нього новий елемент:

```
element.appendChild (para);
```

Створення нових HTML елементів – insertBefore ()

Метод appendChild () в попередньому прикладі приєднує новий елемент, як останній нащадок.

Якщо вам потрібен протилежний результат, то ви можете скористатися методом insertBefore ():

```
<div id = "div1">
<p id = "p1"> Це параграф. </p>
<p id = "p2"> Це інший параграф. </p>
</div>

<script>
var para = document.createElement ("p");
var node = document.createTextNode ("Це новий параграф.");
para.appendChild (node);

var element = document.getElementById ("div1");
var child = document.getElementById ("p1");
element.insertBefore (para, child);
</script>
```

Видалення існуючих HTML елементів

Щоб видалити HTML елемент, ви повинні знати його батьківський елемент:

```
<div id = "div1">
<p id = "p1"> Це параграф. </p>
<p id = "p2"> Це інший параграф. </p>
</div>

<script>
var parent = document.getElementById ("div1");
var child = document.getElementById ("p1");
parent.removeChild (child);
</script>
```

Увага! У специфікації DOM 4 реалізований метод node.remove (). Однак через слабку підтримки браузерами, вам не слід його використовувати.

Цей HTML документ містить елемент `<div>` з двома вузлами нащадками (два елементи `<p>`):

```
<div id = "div1">
<p id = "p1"> Це параграф. </p>
<p id = "p2"> Це інший параграф. </p>
</div>
```

Знайдемо елемент з `id = "div1"`:

```
var parent = document.getElementById("div1");
```

Знайдемо елемент `<p>` з `id = "p1"`:

```
var child = document.getElementById ("p1");
```

Видалимо нащадка у батька:

```
parent.removeChild (child);
```

Звичайно, було б добре мати можливість просто видалити елемент, без посилання на його батька. Але, на жаль, DOM так влаштована, що їй потрібно знати як сам елемент, який потрібно видалити, так і його батька.

Таким чином, маємо таку послідовність дій: знаходимо елемент, який потрібно видалити, і використовуємо його властивість `parentNode`, щоб визначити його батька:

```
var child = document.getElementById ("p1");
child.parentNode.removeChild (child);
```

Заміна HTML елементи

Щоб замінити який-небудь елемент в HTML DOM, використовується метод `replaceChild ()`:

```
<div id = "div1">
<p id = "p1"> Це параграф. </p>
<p id = "p2"> Це інший параграф. </p>
</div>

<script>
var para = document.createElement ("p");
var node = document.createTextNode ("Зовсім новий параграф.");
para.appendChild (node);

 var parent = document.getElementById ("div1");
 var child = document.getElementById ("p1");
 parent.replaceChild (para, child);
</script>
```

Набори елементів

Метод `getElementsByTagName ()` повертає об'єкт HTMLCollection.

Об'єкт HTMLCollection – це схожий на масив список (набір або колекція) HTML елементів.

Наступний код відбирає в документі всі елементи `<p>`:

```
var x = document.getElementsByTagName ("p");
```

Доступ до елементів набору можна отримати за номером індексу. Так, щоб отримати доступ до другого елементу `<p>` в наборі, потрібно написати:

```
y = x [1];
```

Увага! Індексація елементів в наборі починається з 0.

Властивість `length` визначає кількість елементів в наборі HTMLCollection:

```
var myCollection = document.getElementsByTagName ("p");
document.getElementById ("demo").innerHTML = myCollection.length;
```

Пояснення прикладу: створимо набір всіх елементів <p>, виведемо на екран довжину цього набору.

Властивість length буває корисно, коли необхідно в циклі обійти всі елементи набору.

У наступному прикладі змінюється фоновий колір усіх елементів <p>:

```
var myCollection = document.getElementsByTagName ("p");
var i;
for (i = 0; i <myCollection.length; i ++) {
myCollection[i].style.backgroundColor = "red";
}
```

Увага! HTMLCollection – це НЕ масив! HTMLCollection може виглядати як масив, але насправді ним не є. Ви можете в циклі проходити по списку елементів і посилатися на елементи по їх номерами (як в масиві), але ви не можете використовувати методи масиву valueOf (), pop (), push () або join () з HTMLCollection.

Списки вузлів

Об'єкт NodeList – це список (набір) вузлів, витягнутих з документа. Об'єкт NodeList – це майже те ж саме, що об'єкт HTMLCollection.

Деякі (старі) браузери повертають об'єкт NodeList замість HTMLCollection при роботі таких методів як, наприклад, getElementsByClassName ().

У всіх браузерах властивість childNodes повертає об'єкт NodeList.

У більшості браузерів метод querySelectorAll () повертає об'єкт NodeList.

У наступному прикладі відбираються в документі все вузли <p>:

```
var myNodeList = document.querySelectorAll ( "p");
```

Доступ до елементів в об'єкті NodeList можна отримати за номером індексу. Так, щоб отримати доступ до другого вузла <p>, потрібно написати:

```
y = myNodeList [1];
```

Властивість length визначає кількість вузлів в наборі NodeList:

```
var myNodelist = document.querySelectorAll ("p");
document.getElementById ("demo").innerHTML = myNodelist.length;
```

Властивість `length` буває корисно, коли необхідно в циклі обійти всі вузли в списку NodeList.

У наступному прикладі змінюється фоновий колір усіх елементів `<p>` в списку вузлів:

```
var myNodelist = document.querySelectorAll ("p");
var i;
for (i = 0; i <myNodelist.length; i ++) {
myNodelist [i] .style.backgroundColor = "red";
}
```

Увага! NodeList – це НЕ масив! Він може виглядати як масив, але насправді ним не є. Ви можете в циклі проходити по списку вузлів і посилатися на вузли як в масиві, але ви не можете використовувати методи масиву `valueOf (), pop (), push ()` або `join ()` зі списком вузлів.

Різниця між HTMLCollection і NodeList

HTMLCollection – це набір HTML елементів.

NodeList – це набір вузлів документа.

NodeList і HTMLCollection – багато в чому одне й те саме. І об'єкт HTMLCollection, і об'єкт NodeList – схожі на масив списки (набори) об'єктів.

У обох є властивість length, яке визначає кількість об'єктів в списку (наборі).

Обидва, як в масиві, надають індекс (0, 1, 2, 3, 4, ...) для доступу до кожного об'єкту списку.

Доступ до об'єктів списку HTMLCollection можна отримати за їхніми іменами, id або номеру індексу.

Доступ до об'єктів списку NodeList можна отримати тільки по їх номеру індексу.

Тільки об'єкт NodeList може містити вузли атрибутів і текстові вузли.

Питання для самоперевірки:

1. Пояснити поняття HTML DOM. Навести приклади діаграми DOM уявного гіпервидання.
2. Назвати та надати характеристику методів JavaScript для вибору елементу (тегу) веб-сторінки.
3. Назвати та пояснити варіанти оголошення (створення) функцій користувача JavaScript.
4. Пояснити, як «з нуля» додати новий html-елемент (тег) до сторінки гіпервидання за допомогою JavaScript.
5. Пояснити, як можна видалити існуючий html-елемент (тег) у гіпервиданні за допомогою JavaScript.
6. Пояснити, як можна замінити існуючий html-елемент (тег) іншим у гіпервиданні за допомогою JavaScript.
7. Пояснити, як можна замінити стиль існуючого html-елемента (тега) у гіпервиданні за допомогою JavaScript.

ПОДІЇ DOM

Реагування на події

Код JavaScript може виконуватися при виникненні будь-яких подій, на кшталт того, коли користувач натискає мишкою на HTML елемент. Так, наприклад, щоб виконати якийсь код при натисканні користувачем на елемент, необхідно додати код JavaScript в атрибут події HTML елемента:

```
onclick = код JavaScript
```

Приклади HTML подій:

- коли користувач клацає мишкою
- коли веб-сторінка повністю завантажилася
- коли зображення було завантажене
- коли курсор мишки наводиться на елемент
- коли поле введення змінено
- коли HTML форма відправлена
- коли користувач натискає на клавішу клавіатури

У наступному прикладі вміст елемента `<h1>` змінюється, коли користувач клацає на нього мишкою:

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

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

</body>
</html>
```

У наступному прикладі JavaScript функція викликається з обробника події:

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

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

<script>
```

```
function changeText (id) {
    id.innerHTML = "Ой!";
}
</script>

</body>
</html>
```

HTML атрибути подій

Щоб призначити HTML елементу обробник події, потрібно використовувати відповідний атрибут події.

У наступному прикладі призначається обробник події `onclick` для елемента кнопки. При натисканні користувачем на кнопку, виконується функція `displayDate`:

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

Призначення обробника події за допомогою HTML DOM

HTML DOM дозволяє призначити HTML елементу обробник події за допомогою JavaScript.

У наступному прикладі в якості обробника події onclick елемента з `id = "myBtn"` призначається функція `displayDate`. При натисканні користувачем на кнопку, функція виконується:

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

Події onload і onunload

Події `onload` і `onunload` спрацьовують, коли користувач заходить на веб-сторінку або залишає її.

Подія `onload` може використовуватися, наприклад, для перевірки типу і версії користувальницького браузера і завантаження відповідного варіанту веб-сторінки, ґрунтуючись на цій інформації.

Ще події `onload` і `onunload` можуть використовуватися для роботи з файлами кукі.

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

Подія onchange

Часто подія `onchange` використовується в поєднанні з функціями перевірки полів введення.

Нижче показаний приклад, як можна використовувати подія onchange. Функція upperCase () буде викликатися, коли користувач змінює вміст поля введення:

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

Події onmouseover і onmouseout

Події onmouseover і onmouseout можуть використовуватися для виклику певних функцій тоді, коли користувач наводить курсор миші на HTML елемент або виводить його з елементу. –

Події onmousedown, onmouseup і onclick

Події onmousedown, onmouseup і onclick – це все група подій, пов'язаних з натисканням кнопки мишки. Коли натискається кнопка миші, спочатку виникає подія onmousedown, потім, коли кнопка миші відпускається, виникає подія onmouseup, і, нарешті, коли всі ці події закінчилися, виникає подія onclick.

Метод addEventListener

Додамо обробник події, який спрацьовує при натисканні користувачем на кнопку:

```
document.getElementById ("myBtn").addEventListener ("click", displayDate);
```

Метод addEventListener () приєднує обробник події до певного елементу. При цьому новий обробник події не переписує вже існуючі обробники подій.

Таким чином, ви можете додавати скільки завгодно обробників подій до одного елементу. При цьому це можуть бути обробники подій одного типу, наприклад, дві події натискання мишкою.

Ви можете додавати обробники подій до будь-якого об'єкта DOM, а не тільки до HTML елементам, наприклад, до об'єкту вікна.

Метод addEventListener () дозволяє легко контролювати те, як обробник реагує на, так зване, "спливання" події.

Коли використовується метод addEventListener (), JavaScript відділяється від розмітки HTML, що покращує читаність скрипта і дозволяє додавати обробники подій навіть тоді, коли ви не можете контролювати розмітку HTML.

Щоб видалити обробник події, потрібно скористатися методом removeEventListener().

```
елемент.addEventListener (подія, функція, useCapture);
```

- перший параметр – тип події (наприклад, "click" або "mousedown").
- другий параметр – функція, яка буде викликатися при виникненні події.

- третій параметр – логічне значення (true / false), що визначає чи слід відправити подія далі ("спливання") або потрібно закрити цю подію. Цей параметр необов'язковий.

Зверніть увагу, що в імені події не використовується префікс "on" – "click" замість "onclick".

У наступному прикладі при натисканні користувачем на елемент з'являється вікно з повідомленням "Hello World!":

```
елемент.addEventListener ("click", function () {alert ( "Hello World!");});
```

Також, можна задати і зовнішню "іменовану" функцію:

```
елемент.addEventListener ("click", myFunction);

function myFunction () {
   alert ( "Hello World!");
}
```

Метод addEventListener () дозволяє додавати кілька обробників подій до одного і того ж елементом не переписуючи вже існуючі обробники подій:

```
елемент.addEventListener ("click", myFunction);
елемент.addEventListener ("click", mySecondFunction);
```

Також, можна додавати обробники подій різних типів:

```
елемент.addEventListener ("mouseover", myFunction);
елемент.addEventListener ("click", mySecondFunction);
елемент.addEventListener ("mouseout", myThirdFunction);
```

Додавання обробника подій до об'єкту window

Метод addEventListener () дозволяє додавати обробники подій до будь-якого об'єкта HTML DOM – HTML елементам, HTML документу, об'єкту вікна (об'єкт window) та іншим об'єктам, що підтримує події як об'єкт xmlHttpRequest.

У наступному прикладі додається оброблювач події, який спрацьовує, коли користувач змінює розмір вікна браузера:

```
window.addEventListener("resize", function () {
   document.getElementById("demo").innerHTML = "якийсь текст";
});
```

Передача параметрів

Якщо необхідно передати параметри, то використовуйте "анонімну" функцію, яка викликає спеціалізовану функцію з параметрами:

```
елемент.addEventListener ("click", function () {myFunction (p1, p2);});
```

Спливання або перехоплення події

В HTML DOM існує два способи поширення події – спливання і перехоплення.

Поширення події – це послідовність обробки події HTML елементами. Якщо у вас є елемент `<p>`, вкладений в елемент `<div>`, і користувач мишкою натискає на елемент `<p>`, то який елемент повинен обробити подія "`click`" першим.

При спливанні першим обробляє подія самий вкладений елемент, потім його батько і т.д .: таким чином спочатку обробляти подія "`click`" буде елемент `<p>`, а потім елемент `<div>`.

При перехопленні все відбувається навпаки – спочатку подія обробляє самий зовнішній елемент, в нашому випадку `<div>`, а потім вкладений, тобто елемент `<p>`.

Метод `addEventListener ()` дозволяє задавати тип поширення події. Це можна зробити за допомогою параметра "`useCapture`":

```
addEventListener (подія, функція, useCapture);
```

За замовчуванням цей параметр має значення false, що задає спливання події. Якщо задати йому значення true, то буде використовуватися перехоплення.

```
document.getElementById ("myP").addEventListener ("click", myFunction, true);
document.getElementById ("myDiv").addEventListener ("click", myFunction, true);
```

Метод removeEventListener ()

Метод `removeEventListener ()` видаляє обробник події, підключений методом `addEventListener ()`:

```
елемент.removeEventListener ("mousemove", myFunction);
```

Об'єктна модель браузера

Об'єктна модель браузера (BOM від англ. Browser Object Model) дозволяє JavaScript "спілкуватися" з браузером.

Не існує будь-яких офіційних стандартів для Об'єктній моделі браузера (BOM). Так як сучасні браузери реалізують (майже) одні й ті ж методи і властивості для JavaScript інтерактивно, їх часто відносять до методів і властивостей BOM.

Об'єкт window

Об'єкт `window` підтримується всіма браузерами. Він представляє вікно браузера. Всі глобальні JavaScript об'єкти, функції і змінні автоматично стають членами об'єкта `window`.

Глобальні змінні є властивостями об'єкта `window`.

Глобальні функції є методами об'єкта `window`.

Навіть об'єкт `document` (в HTML DOM) є властивістю об'єкта `window`:

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

те ж саме що:

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

Розмір вікна

Щоб визначити розмір вікна браузера, можна використовувати дві властивості. Обидва властивості повертають розмір в пікселях:

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

Вікно браузера (область перегляду) не включає панель інструментів і смугу прокрутки.

Інші методи об'єкта `window`:

```
window.open () – відкриває нове вікно
window.close () – закриває поточне вікно
window.moveTo () – пересуває поточне вікно
window.resizeTo () – змінює розмір поточного вікна
```

Об'єкт Screen

Об'єкт `window.screen` містить інформацію про екран користувача. Об'єкт window.screen можна використовувати без префікса `window`.

Властивості: `screen.width; screen.height; screen.availWidth; screen.availHeight; screen.colorDepth; screen.pixelDepth.`

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

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

Властивість screen.height повертає висоту екрану користувача в пікселях:

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

Властивість screen.availWidth повертає ширину екрана користувача в пікселях мінус інтерфейс на зразок таскбара Windows.

У наступному прикладі виводиться доступна ширина екрану:

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

Властивість screen.availHeight повертає висоту екрану користувача в пікселях мінус інтерфейс на зразок таскбара Windows.

У наступному прикладі виводиться доступна висота екрану:

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

Властивість screen.colorDepth повертає число біт, що використовуються для відображення кольору.

У наступному прикладі виводиться глибина кольору екрану в бітах:

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

Властивість screen.pixelDepth повертає глибину пікселя екрану в бітах:

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

Об'єкт Location

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

Об'єкт window.location може записуватися без префікса window.

Властивість window.location.href повертає URL поточної сторінки.

```
document.getElementById ( "demo").innerHTML =
"МІСТО" + window.location.href;
```

Властивість window.location.hostname повертає ім'я інтернет хоста (поточної сторінки).

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

Властивість window.location.pathname повертає шлях поточної сторінки.

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

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

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

Властивість window.location.port повертає номер порту інтернет хоста (поточної сторінки).

```
document.getElementById ("demo").innerHTML =
"Номер порту:" + window.location.port;
```

Метод window.location.assign () завантажує новий документ:

```
<html>
<head>
<script>
function newDoc () {
   window.location.assign ("https://msiter.ru")
}
</script>
</head>
<body>

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

</body>
</html>
```

Об'єкт History

Об'єкт `window.history` містить історію відвіданих браузером сторінок. Об'єкт `window.history` може записуватися без префікса `window`.

Щоб забезпечити приватність користувачів, є деякі обмеження того, як JavaScript може отримати доступ до цього об'єкта.

Метод `history.back ()` завантажує попередній URL в списку відвіданих сторінок. По суті цей метод діє так само, як кнопка браузера "Назад".

У наступному прикладі на сторінці створюється своя кнопка повернення назад:

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

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

</body>
</html>
```

Метод `history.forward ()` завантажує наступний URL в списку відвіданих сторінок. По суті цей метод діє так само, як кнопка браузера "Вперед".

У наступному прикладі на сторінці створюється своя кнопка переходу вперед:

```
<html>
<head>
<script>
function goForward () {
   window.history.forward ()
 }
</script>
</head>
<body>
<input type = "button" value = "Вперед" onclick = "goForward ()">
</body>
</html>
```

Об'єкт Navigator

Об'єкт `window.navigator` містить інформацію про браузер відвідувача сторінки. Об'єкт `window.navigator` може записуватися без префікса `window`.

Деякі приклади: `navigator.appName; navigator.appCodeName; navigator.platform.`

Властивість `appName` повертає ім'я браузера, як додатку:

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

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

Властивість `appCodeName` повертає кодове ім'я браузера:

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

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

Властивість `product` повертає ім'я движка браузера:

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

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

Властивість `appVersion` повертає інформацію про версії браузера:

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

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

Властивість `userAgent` повертає заголовок призначеного для користувача агента, посланого браузером сервера:

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

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

Властивість platform повертає платформу браузера (операційну систему):

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

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

Властивість `language` повертає мову браузера:

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

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

Властивість `onLine` повертає true, якщо браузер підключений до мережі Інтернет:

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

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

Метод `javaEnabled ()` повертає true, якщо обробка Java включена:

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

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

УВАГА!!!

Інформація з об'єкта navigator може вводити в оману. Її не слід використовувати для визначення версії браузера, так як:

- різні браузери можуть використовувати один і той же ім'я;
- дані об'єкта `navigator` можуть змінюватися власником браузера;
- деякі браузери дають невірні ідентифікатори про себе, щоб обходити перевірку сайтів;
- браузери не можуть повідомляти про нові операційних системах, які вийшли після самих браузерів.

Спливаючі вікна повідомлень

В JavaScript є три види спливаючих вікон: з попередженням, з підтвердженням і з пропозицією.

Вікно з попередженням

Вікно з попередженням часто використовується, якщо ви хочете довести якусь інформацію до користувача. Коли з'являється вікно з попередженням, то, щоб продовжити, користувач повинен натиснути кнопку "OK".

```
Синтаксис: window.alert ("якийсь текст");
```

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

Приклад:

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

Вікно з підтвердженням

Вікно з підтвердженням часто використовується, якщо необхідно, щоб користувач прийняв або відкинув щось. Коли з'являється вікно з підтвердженням, то, щоб продовжити, користувач повинен натиснути або на кнопку "OK", або на кнопку "Скасувати". Якщо користувач натискає на кнопку "OK", то вікно повертає значення true. Якщо ж користувач натискає на кнопку "Скасувати", то вікно повертає значення false.

```
Синтаксис: window.confirm ( "якийсь текст");
```

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

Приклад:

```
if (confirm ( "Натисніть кнопку!")) {
    txt = "Ви натиснули OK!";
```

```
} else {
    txt = "Ви натиснули Скасування!";
}
```

Вікно з пропозицією введення

Вікно з пропозицією введення часто використовується, коли необхідно, щоб користувач перед входом на сторінку ввів якесь значення. Коли з'являється вікно з пропозицією введення, то, щоб продовжити, користувач повинен ввести якесь значення і натиснути на кнопку "OK" або "Скасувати". Якщо користувач натискає кнопку "OK", то вікно повертає введене значення. Якщо ж користувач натискає кнопку "Скасувати", то вікно повертає значення null.

Синтаксис: `window.prompt ("якийсь текст", "текст за умовчанням");`

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

Приклад:

```
var person = prompt ("Введіть ваше ім'я", "Гаррі Поттер");
if (person == null || person == "") {
    txt = "Користувач скасував введення.";
} else {
    txt = "Привіт" + person + "! Як справи?";
}
```

Переведення рядка

Щоб у спливаючому вікні вивести повідомлення на декількох рядках, потрібно використовувати спеціальний код `"\n"` – зворотна коса риска з символом n.

Приклад:

```
alert ("Привіт! \nЯк справи?");
```

Події за таймером

Об'єкт `window` дозволяє виконувати код через заздалегідь визначений період часу або тимчасові інтервали. Ці тимчасові інтервали називаються подіями за таймером.

Два ключових методи, які дозволяють це реалізувати:

- `setTimeout` (функція, мілісекунди) – виконує функцію по закінченню заданого числа мілісекунд.
- `setInterval` (функція, мілісекунди) – аналогічний методу `setTimeout ()`, але при цьому виклик функції постійно повторюється.

Метод setTimeout ()

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

Перший параметр – це функція, яка викликається.

Другий параметр задає час в мілісекундах, яке повинно пройти до виклику функції.

Метод `windsetTimeout ()` може записуватися без префікса `window`.

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

```
<button onclick = "setTimeout (myFunction, 3000)"> Try it </button>
<script>
function myFunction () {
   alert ('Привіт');
}
</script>
```

Метод clearTimeout ()

Метод `clearTimeout ()` зупиняє виконання функції, викликаної в методі `setTimeout()`.

Синтаксис: `window.clearTimeout (timeoutVariable);`

Може записуватися без префікса `window`.

Метод `clearTimeout ()` використовує змінну, що повертається методом `setTimeout()`:

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

Якщо функція, що спрацьовує за таймером, ще не була викликана, то її виконання можна зупинити викликавши метод `clearTimeout ()`.

Наступний приклад аналогічний попередньому, тільки додано кнопку `"Зупинити"`:

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

Метод setInterval ()

Метод `setInterval ()` регулярно повторює виклик функції через заданий інтервал часу. Може записуватися без префікса `window`.

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

Перший параметр – це функція, яка викликається.

Другий параметр задає час в мілісекундах, яке повинно пройти між викликами функції.

У наступному прикладі викликається функція myTimer, яка виводить поточний час, кожну секунду (як в цифрових годинниках):

```
var myVar = setInterval (myTimer 1000);
function myTimer () {
   var d = new Date ();
   document.getElementById ("demo").innerHTML = d.toLocaleTimeString ();
}
```

Увага! В 1 секунді – 1000 мілісекунд.

Метод clearInterval ()

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

Синтаксис: window.clearInterval (timerVariable);

Метод clearInterval () використовує змінну, що повертається методом setInterval ():

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

Наступний приклад аналогічний попередньому, тільки додано кнопку "Зупинити час":

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

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

<script>
var myVar = setInterval (myTimer 1000);
function myTimer () {
    var d = new Date ();
    document.getElementById ("demo").innerHTML = d.toLocaleTimeString ();
}
</ script>
```

Питання для самоперевірки:

1. Пояснити поняття події DOM. Навести приклади.
2. Пояснити поняття об’єктна модель браузера, властивості об'єкта window.
3. Надати коротку характеристику основним методам об'єкта window.
4. Надати опис спливаючих вікон повідомлень, синтаксис їх створення. Навести приклади.
5. Пояснити методи обробки подій за таймером. Навести приклади.

БІБЛІОТЕКА JQUERY

Переваги застосування

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

Бібліотека jQuery створена і випущена Джоном Резігом в січні 2006 року. Вона поширює застосування CSS-селекторів для узгодження з вмістом моделі DOM. Крім інших функціональних можливостей, вона надає засоби для маніпулювання моделлю DOM, обробки Ajax запитів і подій, а також анімації.

В даний час ця бібліотека є найпоширенішою серед усіх бібліотек JavaScript.

Крім jQuery існують і інші бібліотеки для JavaScript'а:

- Prototype (prototypejs.org). Являє собою прообраз всіх сучасних бібліотек JavaScript. Створена і випущена Семом Стівесоном в 2005 році. Включає в себе функціональні можливості DOM, Ajax і обробки подій, а також різні прийоми програмування.
- Yahoo UI (developer.yahoo.com/yui). Будучи результатом власної розробки інтегрованого середовища в компанії Yahoo, оприлюднена в лютому 2006 року. Включає в себе функціональні можливості DOM, Ajax, обробки подій і анімації, а також цілий ряд попередньо побудованих віджетів. (Міні додатків типу календаря, сітки, меню гармошки і т.п.)
- base2 (code.google.com/p/base2). Створена Діном Едвардсом і випущена в березні 2007 року для підтримки функціональних можливостей DOM і обробки подій. Претендує на популярність своїми спробами реалізувати різні специфікації стандарту W3C в універсальному, крос-браузерному вигляді.

Всі згадані вище бібліотеки грамотно побудовані, щоб всебічно охоплювати ті проблемні області, для яких вони розроблені.

Але, повернемося до jQuery. Розглянемо на прикладі переваги його застосування. Почнемо відразу з цікавого, хоч і непрактично прикладу. Необхідно розфарбувати таблицю в такому стилі:

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

Передбачається, що у нас є CSS-файл, в якому визначено клас «odd». Все, що нам потрібно зробити, – це наділити непарні рядки в таблиці ці класом. Для початку – рішення на чистому Javascript:

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

Тепер – jQuery:

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

Як видно з цього прикладу, бібліотека jQuery дозволяє знаходити прості і витончені рішення для, здавалося б, складних завдань. Нижче наводиться мінімальний html-документ, до якого підключена бібліотека jQuery:

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

Тут в рядках 7 і 8 здійснюється підключення бібліотеки jQuery. У нашому випадку вона зберігається в тому ж каталозі, що і html-документи. Центральними в скрипті є рядки 10-14, сенс яких полягає в тому, щоб забезпечити виклик безіменній функції (рядки 11-13) після повного завершення завантаження і підготовки до роботи html-документа. Це стандартний прийом роботи з бібліотекою, тому що тільки після завершення завантаження документа всі його елементи стають доступними для звернення до них з скрипта.

Принцип роботи jQuery

Основною відмінністю jQuery від інших бібліотек є те, що можна застосовувати якісь дії не тільки до окремого елементу, але і до колекції в цілому. Тобто досить виділити потрібну колекцію елементів і застосувати до неї необхідні дії.

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

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

З введенням CSS в веб-технології для відділення уявлення від вмісту знадобився спосіб, що дозволяє посилатися на групи елементів сторінки з зовнішніх таблиць стилів. В результаті був розроблений метод, заснований на використанні селекторів. Бібліотека jQuery використовує ті ж самі селектори.

Синтаксис відбору групи елементів простий:

```
$ (selector) або jQuery (selector)
```

Функція `$()` (псевдонім функції jQuery `()`) повертає спеціальний об'єкт JS, який містить масив елементів DOM, відповідних вказаним селектору. У цього об'єкта багато зручних зумовлених методів, здатних впливати на групу елементів.

На мові програмування такого роду конструкція називається обгорткою (wrapper), тому що вона обгортали відібрані елементи додатковою функціональністю. Ми будемо використовувати термін обгортка jQuery, або обгорнутий набір, посилаючись на групи елементів, управляти якими дозволяють методи, певні в jQuery.

Припустимо, нам потрібно реалізувати поступове зникнення всіх елементів `<div>`. jQuery дозволяє зробити це так:

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

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

```
$ ("div").FadeOut ().AddClass ("removed");
```

Такий ланцюжок (chain) команд jQuery можна продовжувати до нескінченності. В інтернеті можна без зусиль відшукати приклади ланцюжків jQuery, що складаються з десятків команд.

Незважаючи на те, що група відібраних елементів представлена досить складним об'єктом JS, в разі необхідності ми можемо звертатися до неї як до звичайного масиву елементів. В результаті такі дві інструкції дають ідентичні результати:

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

Оскільки тут використовували селектор по атрибуту ID, йому буде відповідати один елемент. У першому випадку використовується метод бібліотеки jQuery – html (), який заміщає вміст елемента DOM деякої HTML-розміткою. У другому випадку за допомогою jQuery витягується масив елементів, вибирається перший елемент масиву з індексом 0 і потім його вміст заміщається за допомогою самого простого способу JS.

Той же результат можна отримати за допомогою селектора, який може відібрати декілька елементів:

```
$ ("div").html ("let's add some text!");
```

або

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

Бібліотека jQuery підтримує різні типи селекторів. Ось кілька прикладів:

Селектор	**Опис**
`$ ("p: even");`	відбирає все парні елементи <p>
`$ ("tr: nth-child (1)");`	відбирає перші рядки в усіх таблицях
`$ ("body> div");`	відбирає елементи `<div>`, що є прямими нащадками елемента `<body>`
`$ ("a [href $ = pdf]");`	відбирає посилання на файли PDF
`$ ("body> div: has (a)");`	відбирає елементи `<div>`, які є прямими нащадками елемента `<body>` і містять посилання

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

Визначення функції $ ()

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

Розберемося з кожним детальніше.

$ (html)

Дозволяє створити html-елементи «на льоту» з «чистого» HTML. Наприклад, можна створити елемент `div`, що містить параграф з текстом «`Привіт!`» і додати його до елемента з `id = "body"` таким чином:

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

Або ще коротше:

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

Елемент до відпрацювання скрипта:

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

Елемент після відпрацювання скрипта:

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

$ (elems)

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

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

$ (expr [, context])

Це найбільш часто використовувана форма функції $. Перший, обов'язковий, параметр – це вираз, який дозволить jQuery знайти елемент на сторінці. Другий, необов'язковий, параметр вказує, де шукати цей елемент (за замовчуванням jQuery буде шукати по всій сторінці).

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

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

Наш документ:

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

Результат:

```
[<p> два </p>]
```

Знайдемо всі радіокнопки в першій формі на сторінці:

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

Ще приклад: виклик методу $("div") повертає колекцію всіх елементів div документа.

```
 <! DOCTYPE HTML PUBLIC "- // W3C // DTD HTML 4.01 Transitional // EN">
<html>
   <head>
<title> приписують клас divc до всіх елементів div документа </title>
   <! – Підключаємо бібліотеку jQuery версії 1.11.1 ->
   <script type = "text / javascript" src = "jquery-1.11.1.js"> </script>
   <script type = "text / javascript">
      $ (
         function () {
            $("div").AddClass("divc");
         }
      )
   </script>
<style type = "text / css">
.divc {
   border: double 3 px navy;
   width: 100 px;
   height: 100 px;
```

```
   background-color: # ffffa0;
   text-align: center;
   font-size: 5em;
   font-weight: bolder;
}
</style>
</head>
   <body bgcolor = "# cacaff">
   <h1> приписують клас divc до всіх елементів div документа</h1>
      <div> 1 </div>
      <div> 2 </div>
      <div> 3 </div>
   </body>
</html>
```

Тут в тілі html-документа визначили три елементи `div`, що містять тільки їх номери. Вибірка колекції всіх елементів div документа здійснюється за допомогою виклику `$("div")`, метод `addClass("divc")`, який застосовується до колекції, припише елементів div css-клас `divc`. Результат цього зображений на рисунку.

Можна модифікувати результат, додатково змінивши колір фону і тексту елементів div, робиться це за допомогою виклику методу css ()

```
$("div").addClass("divc").css("backgroundcolor", "red").css ("color", "white");
```

Тут до елементів `div` послідовно застосовується клас `divc`, після чого колір фону змінюється на червоний, а колір переднього плану на білий. Такі ланцюжки викликів методів мають високу виразністю, лаконічні і широко застосовуються при роботі з бібліотекою.

Для завдання значення властивості css використовувався виклик однойменного методу з двома параметрами `css ("імя_властивість", "значення_властивість ")`.

Якщо викликати метод з одним параметром, то він поверне значення властивості, наприклад, результатом виклику

```
alert ("Колір фону:" + $ ("div").css("background-color"));
```

Таким чином, основним завданням при роботі з бібліотекою є виділення необхідного набору елементів документа і виконання з ним необхідних дій.

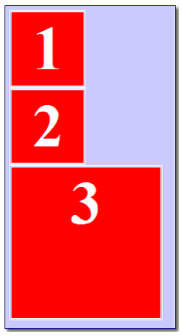
Засоби відбору колекцій елементів в jQuery

У бібліотеку вбудовані надзвичайно потужні засоби відбору елементів. Почнемо ми з того, як вибрати єдиний унікальний елемент. Для того щоб вибрати його, необхідно унікально помітити, задавши в елементі значення атрибута `id`. Трохи змінимо попередній приклад, задавши ідентифікатор останнього елемента `div3` і змінивши його розміри.

```
<! DOCTYPE HTML PUBLIC "- // W3C // DTD HTML 4.01 Transitional // EN">
<html>
   <head>
   <title> Вибираємо унікальний елемент і міняємо його властивості </title>
     <! – Підключаємо бібліотеку jQuery версії 1.3.2 ->
     <script type = "text / javascript" src = "jquery-1.3.2.js"> </script>
     <script type = "text / javascript">
        $(
           function () {
             $("div").addClass("divc").css("background-color", "red").css("color",
"white");
             $("# div3").css("width", "200px").css("height", "200px");
           }
        )
     </script>
   <style type = "text / css">
    .divc {
       border: double 3 px navy;
       width: 100 px;
       height: 100 px;
       background-color: # ffffa0;
       text-align: center;
       font-size: 5em;
       font-weight: bolder;
    }
   </style>
   </head>
   <body bgcolor = "# cacaff">
      <h1> Вибираємо унікальний елемент і міняємо його властивості </h1>
      <div> 1 </div>
```

```
      <div> 2 </div>
      <div id = "div3"> 3 </div>
   </body>
</html>
```

Тут два рядки змінені в порівнянні з попереднім прикладом: ми задали ідентифікатор для одного з елементів `div` і двічі застосували до нього метод `css ()` для зміни його ширини і висоти.



Тепер перейдемо до більш складним прикладів, відповідаючи на питання, як нам відшукати в документі складну конструкцію, спробуємо змінити властивості не просто елементів `div`, а тільки тих, які вкладені в `div`, тобто конструкції виду:

```
<div> <div> div в div </div> </div>
```

і змінимо для нього колір фону. Зробити це можна за допомогою конструкції:

```
$ ("div> div").css ("background-color", "lime");
```

Іноді буває потрібно і доцільно змінити, наприклад, всі посилання на сторінці, зробивши так, щоб при переході по посиланню вміст відкривалося не в тому ж, а новому вікні. Зробити це можна, додавши в тег `<a>` атрибут `target = "_blank"`.

Описати завдання виявилося важче, ніж її вирішити, рішення здійснюється в один рядок.

```
$("a").attr ("target", "_blank")
```

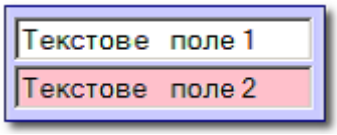
`$("a")` відбирає все гіперпосилання в документі, а метод `attr ()`, додає необхідний атрибут.

Тепер давайте спробуємо дістатися до елементів через атрибути тегів, змінивши у всіх елементів, доступних тільки для читання, фоновий колір на рожевий. Проробимо ці дії на прикладі двох текстових полів:

```
<form method = "get" action = "#">
    <input type = "text" value = "Текстове поле 1">
    <br> <input type = "text" value = "Текстове поле 2" readonly = "true">
</form>
```

Для перефарбовування необхідно ввести в обробник завершення завантаження документа рядок (виділено сірим):

```
<script type = "text / javascript">
   $(
      function () {
        $("input [readonly]").css ("background-color", "pink");
      }
   )
</script>
```



Тепер наведемо класичний приклад перефарбування рядків html-таблиці через одну. Цей приклад містять практично всі керівництва по бібліотеці.

```
01 <html>
02 <head>
03 <meta http-equiv = "content-type" content = "text / html;
charset = windows-1251 ">
04 <title> Різнокольорові рядки таблиці </title>
05 <! – Підключаємо бібліотеку jQuery версії 1.11.1 ->
06 <script type = "text / javascript" src = "jquery-1.11.1.js"> </script>
07 <script type = "text / javascript">
08 $(
09 function () {
10 $("tr: nth-child (odd)").AddClass ("odd");
11 $("tr: nth-child (even)").AddClass ("even");
12 $("tr: nth-child (1)").
css("background-color", "# 0000ff").css("color", "white").css("fontweight",
"700").css(" font-family "," sans-serif ");
13 }
14 )
15 </script>
16 <style type = "text / css">
```

```
17 .odd {background-color: # ffffa0;}
18 .even {background-color: # a0ffa0;}
19 </style>
20 </head>
21 <body bgcolor = "# cacaff">
22 <h1 align = "center"> Різнокольорові рядки таблиці </h1>
23 <table align = "center" cellspacing = "0" cellpadding = "4" border = "1">
24 <tr> <th colspan = "2"> Заголовок таблиці </th> </tr> <tr> <td> Рядок: 2, стовпець:
1 </td> <td> Рядок: 2, стовпець: 2 </td> </tr>
25 <tr> <td> Рядок: 3, стовпець: 1 </td> <td> Рядок: 3, стовпець: 2 </td> </tr>
26 <tr> <td> Рядок: 4, стовпець: 1 </td> <td> Рядок: 4, стовпець: 2 </td> </tr>
27 <tr> <td> Рядок: 5, стовпець: 1 </td> <td> Рядок: 5, стовпець: 2 </td> </tr>
28 </table>
29 </body>
30 </html>
```

Тут в рядках 24-29 описана html-таблиця, яка містить рядок заголовка і ще п'ять рядків.

Створено css-класи для відображення непарних (`odd`) і парних (`even`) рядків таблиці (рядки вихідного тексту 16-19).

Вибірка рядків таблиці здійснюється за допомогою селектора `tr: nth-child ()`, що означає `n`-ий нащадок серед елементів `tr` (батьківським елементом є `table`). Як параметр вказується номер нащадка. Також використовували `odd` для виділення непарних рядків, `even` – парних, 1 – для першого рядка (див. рядки 10 – 12 лістингу).

Бібліотека володіє надзвичайно потужними засобами для відбору колекцій елементів документа, тут ми познайомилися тільки з малою їх частиною.

У бібліотеці jQuery реалізований механізм пошуку елементів, який використовує CSS і Xpath. Тобто, для знаходження необхідного елемента ви можете скористатися як механізмом селектор CSS, так і запитами по документу в стилі Xpath. Крім використання селекторів CSS, в jQuery можна використовувати XPath.

XPath (XML Path Language) є мовою для звернення до частин XML-документа. XHTML, є підмножиною XML, тому за допомогою XPath можна звертатися до частин HTML-документа. Дана властивість можна використовувати в jQuery, так як ця бібліотека підтримує досить велика підмножина XPath і об'єднує його з деякими селекторами CSS для створення гнучкого механізму пошуку елементів на сторінці.

Наприклад:

Знайдемо всі елементи `p`, які містять посилання

```
$ ("p [a]");
```

Знайдемо всі елементи `a`, які містять атрибут `rel`, рівний "`nofollow`"

```
$ ("//a [@ rel = 'nofollow']");
```

Знайдемо всі видимі елементи `li` всередині елемента `ul`

```
$ ("ul / li: visible"); // XPath + селектор
// або
$ ("ul> li: visible"); // CSS + селектор
```

Методи для маніпуляції елементами документа і їх властивостями

Query пропонує розробнику велика кількість методів для маніпуляції елементами документа і їх властивостями. Коротко розглянемо деякі з цих методів, а також деякі особливості роботи з ними.

Метод	**Опис**
`append(content)/prepend(content)`	Додати переданий елемент або вираз в кінець / в початок обраного елемента.
`appendTo(expr)/prependTo(expr)`	Додати вибраний елемент в кінець / в початок переданого елемента.
`attr(name)`	Отримати значення атрибута.
`attr(params)`	Встановити значення атрибутів. Атрибути передаються у вигляді `{ключ1: значення1 [ключ2: значення2 [...]]}`
`attr(name, value)`	Встановити значення одного атрибута.
`css(name)/css(params)/css(name, value)`	Отримати / встановити значення окремих параметрів CSS. Аналогічно функції `attr ()`.
`text()/text(val)`	Отримати / задати текст елемента. У разі введення тексту спеціальні символи HTML замінюються символьними примітивами (entities, наприклад, знак ">" буде замінений >)
`empty()`	Видалити всі піделементи поточного елемента.

У наступних прикладах ми будемо працювати з наступним HTML-кодом:

```
<div id = "my-div">
  <a href="http://google.com/" id="my-link"> Посилання </a>
</div>
```

Приклад 1:

```
$ ("# my-div").html (); // поверне <a href="#" id="my-link"> Посилання </a>
```

Приклад 2:

```
$ ("# my-link").attr ("href"); // поверне http://google.com/
```

Приклад 3:

```
$ ("# my-div").append ("<strong> Жирний текст </ strong>");
// або
$ ("<strong> Жирний текст </strong>").AppendTo ($ ("# my-div"));
```

HTML стане таким:

```
<div id = "my-div">
  <a href="http://google.com/" id="my-link"> Посилання </a>
  <strong> Жирний текст </strong>
</div>
```

Приклад 4:

```
$ ("# my-div").empty ();
```

HTML стане таким:

```
<div id = "# my-div"> </div>
```

Приклад 5:

```
$ ("# my-div").text ("<a> Якір </a>");
```

HTML стане таким:

```
<div id = "# my-div"> <a> Якір </a> "</div>
```

Приклад 6:

```
$ ("# my-div").css (
  {
    backgroundColor: "# F00",
    color: "# 00F"
  }
);
```

HTML стане таким:

```
<div id = "my-div" style = "background-color: # F00; color: # 00F">
  <a> Якір </a>;
</div>
```

В останньому прикладі зверніть увагу, що для складені властивостей CSS начебто `background-color`, `font-weight` і інших використовуються їх еквіваленти з JavaScript (`backgroundColor`, `fontWeight` і т.п.).

Обробка подій

Принцип обробки подій в бібліотеці простий: для відібраних елементів викликається метод, що зв'язує з подією функцію, що викликається при порушенні події. Наприклад, для обробки натискання кнопки з ідентифікатором clickMe досить визначити наступний обробник:

```
<script type = "text / javascript">
$ (
   function () {
      $ ("#clickMe").click (function ()
         {
            alert ("Натиснули!")
         }
      );
}
)
</script>
```

Тут `click ()` – обробник події натискання кнопки.

При всій своїй простоті такий підхід дозволяє легко і просто управляти елементами документа. Нижче наведемо приклад, що дозволяє клацанням миші на гіперпосиланнях перемішати елемент `div` з ідентифікатором `el` і змінювати його розміри. Сам елемент описаний таким чином:

```
<style type = "text / css">
   #el {
      width: 40px; height: 40px; background-color: yellow;
      border: solid thin red;
      position: absolute; left: 200 px; top: 400px;
      }
</style>
<div id = "el">
</div>
```

Відзначимо, що елемент позиціонується абсолютно, для нього задані колір фону, розміри, положення і рамка.

Для зміни положення і розмірів елемента використовується функція `change ()`

```
01 function change (prop, incr)
02 {
03 var val = $("# el").css (prop);
04 val = parseInt (val);
05 val + = incr;
06 $("# el"). css (prop, val + "px");
07}
```

Функції віддаються два параметри: `prop` – ім'я змінюваного властивості і `incr` – приріст значення змінюваного властивості. Складність полягає в тому, що положення і положення і розміри елемента кодуються з одиницями зміни, наприклад, 200 px і тому являють собою рядки. У рядку 3 в змінну `val` записується значення властивості, воно, як говорилося вище, є строкове значення і має розмірність. У рядку 4 за допомогою функції parseInt () рядок перетвориться в число, після чого це число змінюється на значення приросту (рядок 5). Нарешті, в рядку шість, задається нове значення властивості. Зауважте, дописали його розмірність.

Гіперпосилання для зміни властивостей елемента зверстані в таблицю:

```
<table>
   <tr>
      <td> <a id="right" href="#"> Вправо 20 </a> </td>
      <Td> <a id="left" href="#"> Ліворуч 20 </a> </td>
   </tr>
   <tr>
      <td> <a id="up" href="#"> Вгору 20 </a> </td>
      <td> <a id="down" href="#"> Вниз 20 </a> </td>
   </tr>
   <tr>
      <td> <a id="plus" href="#"> Збільшити </a> </td>
      <td> <a id="minus" href="#"> Зменшити </a> </td>
   </tr>
</table>
```

Вони необхідні для переміщення елемента вправо, вліво, вгору, вниз на 20 пікселів, а також збільшення ширину і висоти на те ж значення.

Обробка клацань миші на гіперпосиланнях здійснюється безіменній функцією

```
$ (
   function () {
      $ ("# right"). click (function ()
         {change ( "left", 20); }
      );
      $ ("# left"). click (function ()
         {change ( "left", -20);}
      );
      $ ("# up"). click (function ()
         {change ("top", -20);}
      );
      $ ("# down").click (function ()
         {change ("top", 20);}
      );
      $ ("# plus").click (function ()
         {change ("width", 20); change ("height", 20);}
      );
      $ ("# minus").click (function ()
         {change ("width", -20); change ("height", -20);}
      );
   }
)
```

Як і раніше тут описані обробники події клацань миші на гіперпосиланнях, в яких за допомогою виклику функції `change ()` здійснюються маніпуляції над елементом документа.

Наведемо ще один приклад, легко дозволяє управляти видимістю елементів. Видимість задається властивістю css `display`. Якщо задати `display: none`, то елемент стає невидимим, якщо `display: inline`, то елемент відображається в потоці виведення, `display: block` – для відображення створюється контейнер і він відображається так, як якщо б опинився всередині елемента `div`.

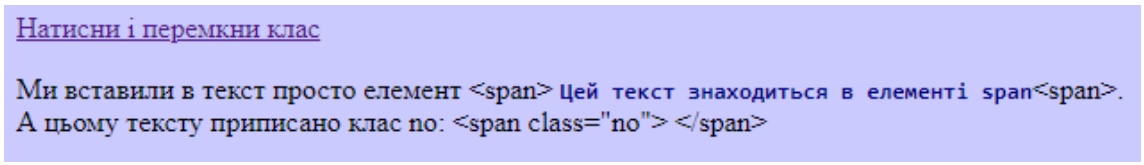
Створимо наступний фрагмент html:

```
<p> Ми вставили в текст просто елемент &lt;span&gt;
<span>Цей текст знаходиться в елементі span</span>&lt;span&gt.
<br>А цьому тексту приписано клас no: &lt;span class="no"&gt;
<span class="no">Цей текст знаходиться в елементі span, йому початково приписано клас no</span>&lt;/span&gt;
</p>
```

Використовувана в прикладі розмітка css має вигляд:

```
<style type = "text / css">
    span {font-weight: 700; color: navy; font-family: monospace; display: inline;}
    .no {display: none};
</ style>
```

Зауважимо, що другий елемент span не відображається.



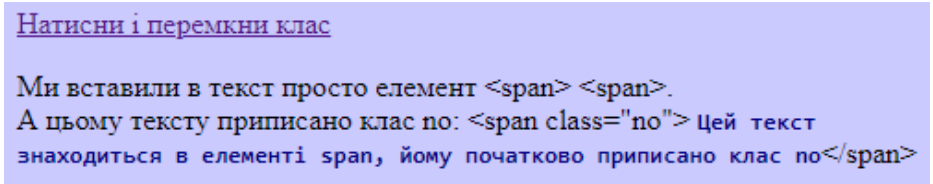
Можна налаштовувати видимість за допомогою методу `toggleClass (ім'я_класу)`, який перемикає приписування елементу класу з ім'ям ім'я_класу.

Скрипт, який здійснює перемикання має вигляд:

```
<script type = "text / javascript">
    $ (
       function () {
          $ ("# toggle").click (function ()
             {
                $ ("span").toggleClass ("no");
             }
          );
       }
    )
```

Для перемикання стилю створена гіперпосилання

```
<br> <a href="#" id="toggle"> Натисни і перемкни клас </a>
```



За допомогою даного прийому легко здійснювати управління елементами, які необхідно відкрити / закрити, наприклад, змісту сайтів.

Анімація

Ключовою функцією, на якій базуються всі інші, є функція `animate`:

```
animate (params, speed, easing, callback);
```

де `params` – властивості, які беруть участь в анімації у вигляді пар {ключ: значення}. Наприклад: `{height: "show"}` або `{opacity: 50, width: 100, height: 200}`. Тут:

`speed` – швидкість в мілісекундах.

`easing` – уповільнення анімації (сповільнюється чи до кінця, `"easein"`, або, навпаки, прискорюється, `"easeout"`. Додаткові типи доступні в модулях розширення).

`callback` – функція, яка буде викликана після завершення анімації.

Приклад.

Нехай у нас є елемент `div` з яким-небудь текстом. Ми хочемо приховати цей текст, замінити новим, і показати оновлений текст.

```
$ ("# my-div")
  .animate ({height: "hide"}, 300)
  .text ("Новий текст")
  .animate ({height: "show"}, 300);
```

Увага! Метод animate маніпулює тільки тими атрибутами, для яких можна виставити числове значення (такі як height, weight, opacity, top і т.п.).

Метод animate є основою більшості, якщо не всіх, ефектів jQuery і плагінів. Наприклад, jQuery пропонує наступні методи для показу і приховування елементів:

Метод	**Опис**
`show ([speed [, callback]])`	показати елемент
`hide ([speed [, callback]])`	приховати елемент
`fadeIn (speed [, callback])`	показати елемент шляхом зміни його прозорості
`fadeOut (speed [, callback])`	приховати елемент шляхом зміни його прозорості
`slideDown (speed, callback)`	показати елемент, спустивши його зверху
`slideUp (speed, callback)`	показати елемент, піднявши його знизу

Де `speed` – швидкість в мілісекундах або одне з `"slow"` (600 мілісекунд) або `"fast"` (200 мілісекунд);

`callback` – функція, яка буде викликана після виконання анімації.

Розглянемо, наприклад, реалізацію функції show:

```
show: function (speed, callback) {
  // знаходимо в переданої нам колекції приховані елементи
  var hidden = this.filter (":hidden");
  // якщо нам передана швидкість, то викликаємо метод animate
  if (speed)
  {
    hidden.animate (
      {
        height: "show",
        width: "show",
        opacity: "show"
      },
      speed,
      callback
    )
  }
  else
  {
    // інакше просто проходимо по колекції елементів і показуємо їх,
    // змінюючи стиль display з none на block
    hidden.each (
      function ()
      {
        this.style.display = this.oldblock? this.oldblock: "";
        if (jQuery.css (this, "display") == "none")
          this.style.display = "block";
      }
    );
  }
  return this;
}
```

Необхідно також зауважити, що всі ефекти бібліотеки jQuery застосовуються до елементів не відразу, а по черзі. Тобто, припустимо, що ми написали такий код:

```
for (i = 0; i <10; i ++)
{
  $ ( "# my-div").animate ({height: "show"}, 300);
  $ ( "# my-div").animate ({height: "hide"}, 300);
}
```

В результаті ми отримаємо не безладне моргання, поки двадцять ефектів борються за право показати / приховати елемент «my-div», а плавний показ, потім приховування, потім

знову показ – і так 20 разів. Зверніть увагу, що черга ефектів складається поелементно. Тобто ефекти, застосовані до різних елементів одночасно, будуть виконуватися одночасно.

У бібліотеку вбудовані засоби анімації, з деякими з них ми познайомимося в цьому розділі. Для цього створимо п'ять однакових елементів, які помістимо в таблицю

```
<table align = "center" cellpadding = "2">
   <tr>
      <td> <div id = "a1"> </ div> </td>
      <td> <div id = "a2"> </ div> </td>
      <td> <div id = "a3"> </ div> </td>
      <td> <div id = "a4"> </ div> </td>
      <td> <div id = "a5"> </ div> </td>
   </tr>
   <tr align = "center">
   <td colspan = "5"> <a id="anim" href="#"> Запустити </a> </td>
   </tr>
</table>
```

Нижче елементів розмістимо гіперпосилання з ідентифікатором `anim`, клацання на якій буде управляти анімацією. Природно, задамо розмальовку елементів за допомогою css:

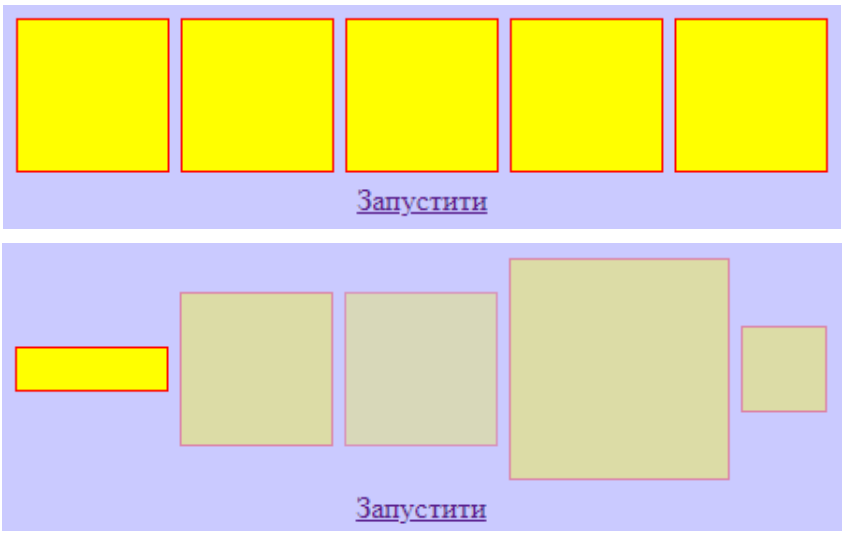
```
<style type = "text / css">
   div {
      width: 80px; height: 80px; background-color: yellow;
      border: solid thin red;
   }
</style>
```

Сама анімація запускається після клацання на гіперпосиланні з написом «Запустити».

```
01 <script type = "text / javascript">
$ (
function () {
$ ("# anim").click (function ()
{
$ ("# a1").slideUp ("slow").slideDown ("slow");
$ ("# a2").fadeTo ('slow', 0.1).fadeTo ('slow', 1.0);
$ ("# a3").fadeOut ('slow'). fadeIn ('slow');
$ ("# a4").animate ({opacity: "0.1", height: "+ = 50", width: "+ = 50"}, "slow").
animate ({opacity: "1.0", height: "- = 50", width: "- = 50"}, "slow");
$ ("# a5").animate ({opacity: "0.1", height: "- = 50", width: "- = 50"}, "slow").
animate ({opacity: "1.0", height: "+ = 50", width: "+ = 50"}, "slow")
```

```
}
);
}
</script>
```

Тут використовуються різні приклади відкриття / закриття елемента, `slideUp ()` / `slideDown ()`, зникнення / появи документа (метод `fadeTo ()`). Цьому методу першим параметром передається тривалість анімаційного ефекту, а другим – кінцева величина непрозорості.



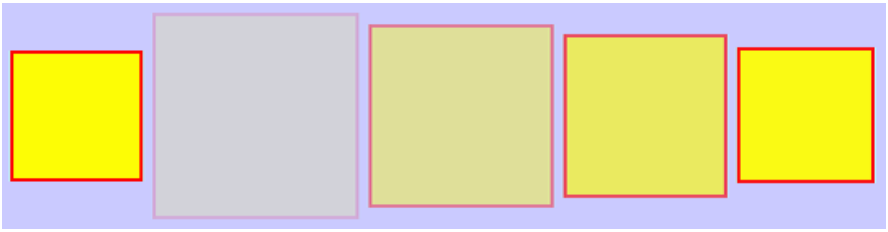
Більш загальним методом `animate ()` передається об'єкт, в якому можна кінцеві задавати css-властивості елемента, включаючи непрозорість, розміри. Другим параметром є тривалість ефекту.

Можна також анімувати наїзд миші на елемент документа, використовуючи для цього метод `hover ()`. Це дозволяє послідовно запускати анімацію, переміщаючи курсор на елементами. Як і раніше використовуємо п'ять елементів `div`. Але в даному випадку трохи перевизначити скрипт, який відповідає за анімацію.

```
01 <script type = "text / javascript">
02 $ (
03 function () {
04 $ ("div").hover (function ()
05 {
06 $ (this).animate ({opacity: "0.1", height: "+ = 50",
07 width: "+ = 50"}, "slow").
08 animate ({opacity: "1.0", height: "- = 50",
09 width: "- = 50"}, "slow")
10}
11);
```

```
12}
13)
14 </script>
```

Даний скрипт при наїзді миші на елемент запускає збільшення розмірів елемента з одночасним зменшенням непрозорості до 0.1, після чого елемент відновлює свої розміри і непрозорість. Якщо провести курсором миші над елементами, метод `hover ()` запуститься в різний час, породжуючи цікавий ефект.



Зверніть увагу на те, яким чином застосовується анімація до об'єкта (рядок 6 лістингу) – методу `$ ()` передається `this`. Це забезпечує коректне застосування анімації при проходженні курсора миші над різними елементами.

Питання для самоперевірки:

1. Надати характеристику бібліотеки jQuery.
2. Пояснити принцип роботи jQuery.
3. Описати методи для маніпуляції елементами документа і їх властивостями за допомогою jQuery.
4. Пояснити принцип обробки подій за допомогою jQuery. Навести приклади.
5. Надати характеристику функції animate.
6. Надати опис методів анімації за допомогою jQuery. Навести приклади.
7. Пояснити, як за допомогою jQuery можна вибрати елемент (тег) веб-сторінки. Надати короткий опис можливих випадків: вибір усіх однакових елементів, вибір окремого елемента, вибір елемента під час спрацювання події, вибір елемента, який знаходиться всередині іншого елемента.
8. Пояснити, як створити новий html-елемент (тег) гіпервидання методами jQuery.

ФРЕЙМВОРК BOOTSTRAP

Створення веб-сторінки за допомогою Bootstrap 4

Bootstrap – безкоштовний front-end фреймворк для більш швидкої і легкої веб-розробки.

Bootstrap включає в себе шаблони дизайнів CSS для форм, кнопок, таблиць, навігації, зображень та інших елементів, а також додаткові JavaScript плагіни.

Bootstrap дає вам можливість легко створювати гнучкий дизайн, який автоматично добре виглядає на багатьох пристроях: від мобільного телефону до широкоекранних робочих столів.

Підключення Bootstrap

Є два варіанти використання Bootstrap: з віддаленого або локального сервера. Якщо ви не хочете зберігати Bootstrap файли на локальний сервер, то потрібно включити в тег `head` наступний код:

```
<!-- Latest compiled and minified CSS -->
<link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">

<!-- jQuery library -->
<script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>

<!-- Latest compiled JavaScript -->
<script
src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>
```

Завантажити файли для сервера можна за посиланням https://getbootstrap.com/ (англійською).

Додаємо в документ HTML5 doctype

Bootstrap використовує HTML-елементи і властивості CSS, для яких потрібно HTML5.

Завжди вказуйте `doctype` для HTML5 на початку сторінки, а також атрибут `lang` і правильний метод кодування символів:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
  </head>
</html>
```

Забезпечуємо правильну візуалізацію для мобільних пристроїв

Bootstrap 4 так само призначений і для мобільних пристроїв. Стилі мобільних пристроїв є частиною базової платформи.

Щоб забезпечити правильну візуалізацію і масштабування торкання, додайте наступний тег `<meta>` всередині тега `<head>`:

```
<meta name = "viewport" content = "width = device-width, initial-scale = 1">
```

Параметр `width = device-width` задає ширину сторінки, що відповідає ширині екрану пристрою (яка залежить від пристрою).

Параметр `initial-scale = 1` задає початковий рівень масштабування, коли сторінка перший раз завантажується браузером.

Вибираємо контейнер

Bootstrap вимагає наявності елемента для обгортання всього вмісту сайту.

На вибір є два класи контейнерів:

- Клас `.container` являє собою чутливий контейнер з фіксованою шириною
- Клас `.container-fluid` є контейнер, що розгортається на всю ширину вікна перегляду.

У наступному прикладі показаний код каркаса Bootstrap сторінки з адаптивним контейнером фіксованої ширини

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Bootstrap Example</title>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">
  <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.0/jquery.min.js"></script>
  <script
src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>
</head>
```

```
<body>

<div class="container">
  <h1>My First Bootstrap Page</h1>
  <p>This is some text.</p>
</div>

</body>
</html>
```

Даний приклад показує сторінку з контейнером не фіксованої ширини.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Bootstrap Example</title>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">
  <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>
  <script
src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>
</head>
<body>

<div class="container-fluid">
  <h1>My First Bootstrap Page</h1>
  <p>This part is inside a .container-fluid class.</p>
  <p>The .container-fluid class provides a full width container, spanning the entire
width of the viewport.</p>
</div>

</body>
</html>
```

Bootstrap сітки

Bootstrap має чутливу, "мобільну" систему сітки, яка масштабується до 12 колонок відповідно до розміру пристрою або до розміру оглядового вікна. Вона включає предвизначені класи для легкого налаштування шаблонів, а також потужні mixins для генерації більш семантичних шаблонів.

Система сітки використовується для створення сторінок шаблону через серію рядків та колонок.

В системі сіток всього є 5 класів:

- `.col-` (додаткові малі пристрої – ширина екрану менше 567 пкс);
- `.col-sm-` (малі пристрої – ширина екрану рівна або більше 576 пкс);
- `.col-md-` (середні пристрої – ширина екрану рівна або більше 768 пкс);
- `.col-lg-` (великі пристрої – ширина екрану рівна або більше 992 пкс);
- `.col-xl-` (XLarge пристрої – ширина екрану рівна або більше 1200 пкс).

Ці класи можуть комбінуватися.

Базова структура в HTML:

```
<!-- Control the column width, and how they should appear on different devices -->
<div class="row">
  <div class="col-*-*"></div>
  <div class="col-*-*"></div>
</div>
<div class="row">
  <div class="col-*-*"></div>
  <div class="col-*-*"></div>
  <div class="col-*-*"></div>
</div>

<!-- Or let Bootstrap automatically handle the layout -->
<div class="row">
  <div class="col"></div>
  <div class="col"></div>
  <div class="col"></div>
</div>
```

Спершу створюємо рядок використовуючи `<div class = "row">`. Далі додаємо бажану кількість стовпців, використовуючи відповідний клас сіток. Перша зірка (*) представляє чутливість, друга – кількість стовпців. Всього їх повинно бути не більше 12.

span 1	span 1	span 1	span 1	span 1	span 1	span 1	span 1	span 1	span 1	span 1	span 1
span 4			span 4			span 4					
span 4			span 8								
span 6						span 6					
span 12											

Як працює система сітки Bootstrap:

- Для правильного вирівнювання та відступів, рядки повинні розміщуватись в межах `.container` (щоб мати фіксовану ширину) або `.container-fluid` (щоб мати повну ширину).
- Використовуйте `.row` для створення рядків.
- Класи, що починаються на `col-` використовуйте для створення колонок.
- Вміст повинен розміщуватись в межах колонок, та лише рядки можуть безпосередньо обгортати колонки.
- Використовуючи, наприклад `.row` та `.col-xs-4`, можна швидко створити шаблон сітки. LESS mixins також можна використовувати для більш раціонального створення шаблонів.
- Колонки мають внутрішні відступи (між їх рамками та вмістом) завдяки властивості `padding`. Ці внутрішні відступи є причиною того, чому рядки мають від'ємне значення `margin-left` та `margin-right`.
- Якщо проглянути приклади сітки нижче, то можна побачити, що вона ширша, за вміст, що не знаходиться в ній. Причина цього – згадане в попередньому пункті від'ємне значення в рядках. Також завдяки цьому, вміст в межах колонок сітки вишиковується в одну лінію з вмістом, що не знаходиться в сітці.
- Колонка сітки створюється через зазначення одного номера із загальної доступної "суми дванадцять" (тобто – загальна сума номерів колонок в одному рядку не повинна перевищувати 12). Наприклад, щоб створити три однакові колонки, які розтягнуться на всю ширину рядка, у кожній із них необхідно встановити клас `.col-xs-4`.
- Якщо в одному рядку вміщено більше, ніж 12 колонок, кожну групу додаткових колонок буде перенесено, як єдине ціле, на новий рядок.
- Класи сітки застосовуються до пристроїв, з шириною екрану більшою або рівною контрольній точці, яка встановлена для даного класу; і в той же час, класи сітки, призначені для пристроїв з вужчим екраном, ігноруються. Таким чином, застосування будь-якого класа `.col-md-` до певного елемента, буде впливати на його стилі не лише на пристроях середнього розміру, але також і на пристроях з широким екраном, якщо клас `.col-lg-` не представлено.

Приклади для застосування цих принципів.

Три одинакові колонки на всіх пристроях і ширині єкрану:

.col	.col	.col

```
<div class="row">
  <div class="col">.col</div>
  <div class="col">.col</div>
```

```
  <div class="col">.col</div>
</div>
```

Чутливі колонки:

Показано створення чотирьох колонок з однаковою шириною починаючи від планшетів і з масштабуванням до великих робочих столів. На мобільних телефонах або екранах менших 576 пкс шириною, колонки автоматично вкладаються одна на одну.

На широкому екрані:

.col-sm-3	.col-sm-3	.col-sm-3	.col-sm-3

На мобільному пристрої:

.col-sm-3
.col-sm-3
.col-sm-3
.col-sm-3

```
<div class="row">
  <div class="col-sm-3">.col-sm-3</div>
  <div class="col-sm-3">.col-sm-3</div>
  <div class="col-sm-3">.col-sm-3</div>
  <div class="col-sm-3">.col-sm-3</div>
</div>
```

Дві не одинакові чутливі колонки:

Показано створення двох не однакових колонок, починаючи від планшетів і з масштабуванням до великих робочих столів.

На широкому екрані:

.col-sm-4	.col-sm-8

На мобільному пристрої:

.col-sm-4
.col-sm-8

```
<div class="row">
  <div class="col-sm-4">.col-sm-4</div>
  <div class="col-sm-8">.col-sm-8</div>
</div>
```

Застосовують наступні медіа запити в LESS файлах для створення ключових контрольних точок у системі сітки.

```
/* Дуже вузькі пристрої (телефони, вужчі ніж 768px) */
/* Немає медіа запиту оскільки це встановлено початково в Bootstrap */

/* Невеликі пристрої (планшети, 768px та ширші) */
@media (min-width: @screen-sm-min) { ... }

/* Пристрої середнього розміру (настільні монітори, 992px та ширші) */
@media (min-width: @screen-md-min) { ... }

/* Великі пристрої (великі настільні монітори, 1200px та ширші) */
@media (min-width: @screen-lg-min) { ... }
```

Іноді додають в ці медіа запити `max-width`, щоб обмежити певні CSS, призначені для вужчого переліку пристроїв.

```
@media (max-width: @screen-xs-max) { ... }
@media (min-width: @screen-sm-min) and (max-width: @screen-sm-max) { ... }
@media (min-width: @screen-md-min) and (max-width: @screen-md-max) { ... }
@media (min-width: @screen-lg-min) { ... }
```

Призначення певних аспектів системи сітки Bootstrap для різних пристроїв.

Параметр	Малі пристрої (<576 px)	Малі пристрої (≥576 px)	Середні пристрої (≥768 px)	Великі пристрої (≥992 px)	XLarge пристрої (≥1200 px)
Поведінка сітки	Горизонтальна завжди	Складена спочатку, але горизонтальна, коли ширша за контрольні точки			
Ширина контейнера	Немає (автоматична)	540 px	720 px	960 px	1140 px
Префікс класа	.col-	`.col-sm-`	`.col-md-`	`.col-lg-`	.col-xl-
# колонок	12				

Ширина проміжку	30 px (15 px з кожної сторони колонки)
Можуть вкладатись	Так
Можуть зміщуватись	Так
Порядок колонок	Так

Більше прикладів формування сітки можна знайти за посиланнями:

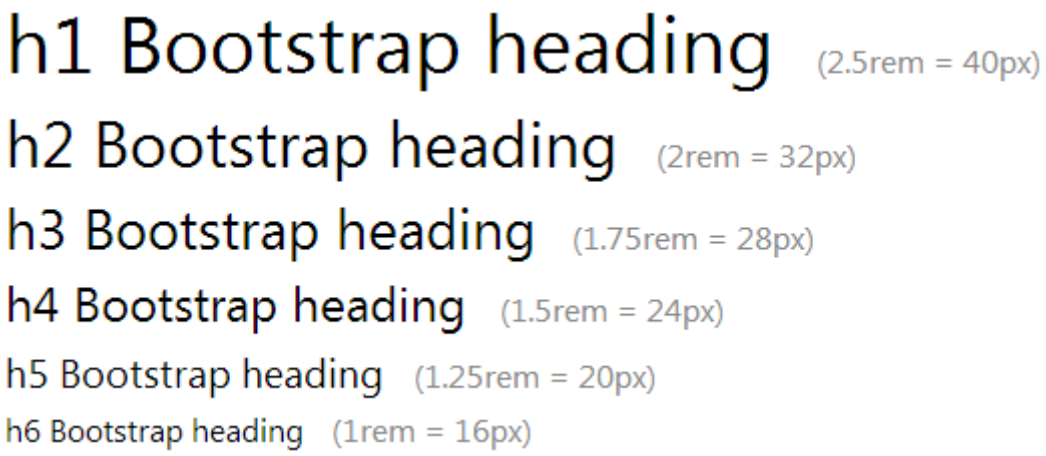
https://html5css.ru/bootstrap4/bootstrap_grid_system.php

https://twbs.docs.org.ua/css/#overview

Типографія

Заголовки

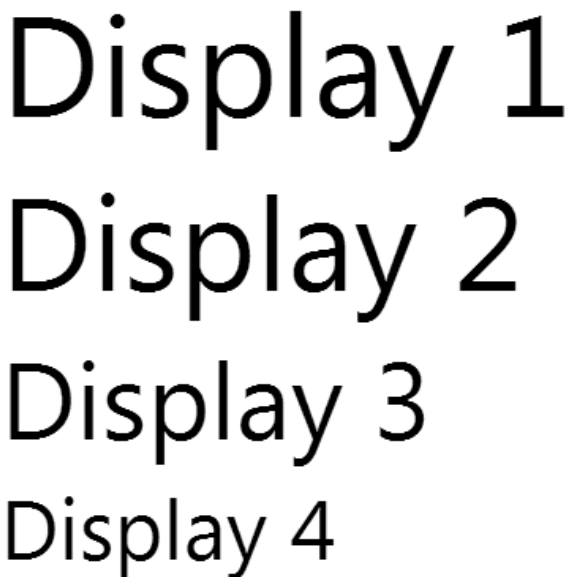
Доступні всі заголовки HTML – від `<h1>` до `<h6>`. Також доступні всі класи від `.h1` до `.h6`, для того випадку, коли ви хочете щоб шрифт був як для заголовка, але відображався в лінію (був вбудованим).



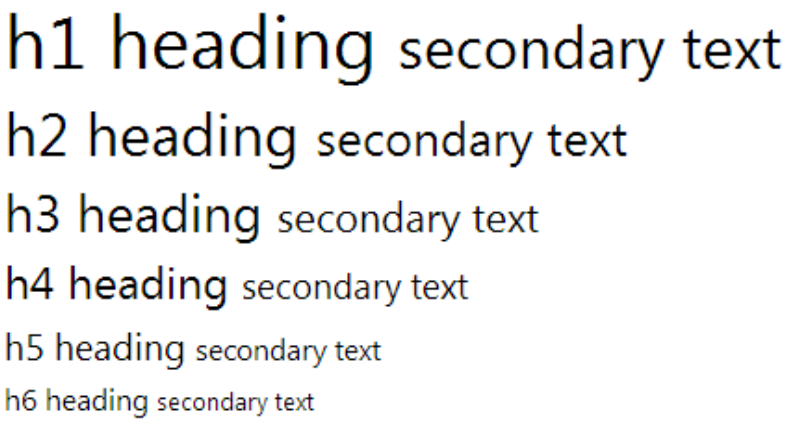

```
<h1>h1. Заголовок Bootstrap</h1>
<h2>h2. Заголовок Bootstrap</h2>
<h3>h3. Заголовок Bootstrap</h3>
<h4>h4. Заголовок Bootstrap</h4>
<h5>h5. Заголовок Bootstrap</h5>
<h6>h6. Заголовок Bootstrap</h6>
```

Для відображення заголовка шрифтом більшого розміру можна скористатися класами: `.display-1, .display-2, .display-3, .display-4.`

```
<h1 class="display-1">Display 1</h1>
<h1 class="display-2">Display 2</h1>
<h1 class="display-3">Display 3</h1>
<h1 class="display-4">Display 4</h1>
```



Створити світліший додатковий текст в будь-якому заголовку можна за допомогою тега `<small>` або класса `.small`.




```
<h1>h1. Заголовок Bootstrap <small>Додатковий текст</small></h1>
<h2>h2. Заголовок Bootstrap <small>Додатковий текст</small></h2>
<h3>h3. Заголовок Bootstrap <small>Додатковий текст</small></h3>
<h4>h4. Заголовок Bootstrap <small>Додатковий текст</small></h4>
<h5>h5. Заголовок Bootstrap <small>Додатковий текст</small></h5>
<h6>h6. Заголовок Bootstrap <small>Додатковий текст</small></h6>
```

Тіло документа

В Bootstrap глобально встановлено `font-size` початково 14 px, `line-height` 1.428. Це застосовується до `<body>` документа та всіх його параграфів. Окрім цього, параграф `<p>` формують відступаючи знизу на висоту рівну половині величини від `line-height` (10 px початково).

Створити провідний параграф можна через додавання класа `.lead`.

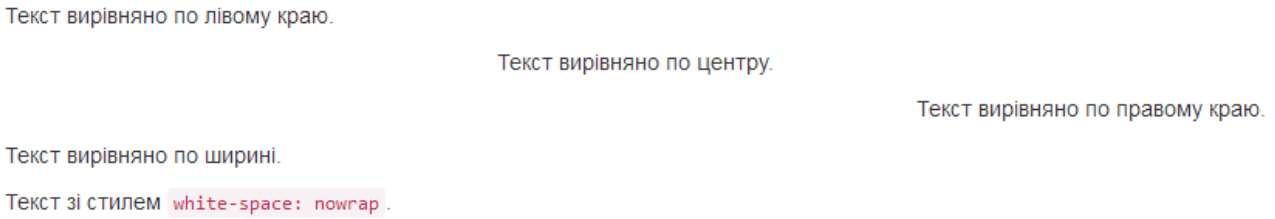
```
<p class="lead">...</p>
```

Вбудовані текстові елементи

Тег	Опис
`<mark>`	Для підсвічування текста, взятого з іншого контекста
`<del>`	Щоб показати блок текста, який позначено як видалений
`<s>`	Щоб показати блок текста, який позначено як неактуальний
`<ins>`	Щоб показати блок текста, який позначено як додатковий
`<u>`	Щоб підкреслити текст
`<small>`	Щоб зменшити акцентування рядка або блока текста (встановлює розмір текста у 85% від батьківського стилю)
`<strong>`	Щоб акцентувати увагу на блоці текста через жирний шрифт
`<em>`	Щоб зробити блок з курсивним текстом

Класи для вирівнювання

Легко вирівняти текст компонентів за допомогою класів вирівнювання.




```
<p class="text-left">Текст вирівняно по лівому краю.</p>
<p class="text-center">Текст вирівняно по центру.</p>
<p class="text-right">Текст вирівняно по правому краю.</p>
<p class="text-justify">Текст вирівняно по ширині.</p>
<p class="text-nowrap">Текст зі стилем <code>white-space: nowrap</code>.</p>
```

Класи перетворень

Зробіть перетворення текста в компонентах з класами зміни регістра літер.

текст з малої літери.

ТЕКСТ З ВЕЛИКОЇ ЛІТЕРИ.

Текст, В Якому Першу Літеру Кожного Слова Змінено На Велику.

```
<p class="text-lowercase">Текст з малої літери.</p>
<p class="text-uppercase">Текст з великої літери.</p>
<p class="text-capitalize">Текст, в якому першу літеру кожного слова змінено на велику.</p>
```

Абревіатури

Впроваджено стилізацію HTML-елементів `<abbr>` для абревіатур та акронімів, щоб показати розширену їх версію при наведенні. Абревіатури з атрибутом `title` мають внизу легке точкове підкреслення, а курсор приймає вигляд знаку питання при наведенні, надаючи додатковий контекст для зависання (`hover`).

Цитати

Цитати потрібні для цитування блоку вмісту з іншого ресурсу в тілі вашого документа.

Обгорніть за допомогою `<blockquote>` будь-який HTML-код, котрий має бути цитатою. Рекомендуємо починати цитату із `<p>`.

> Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer posuere erat a ante.

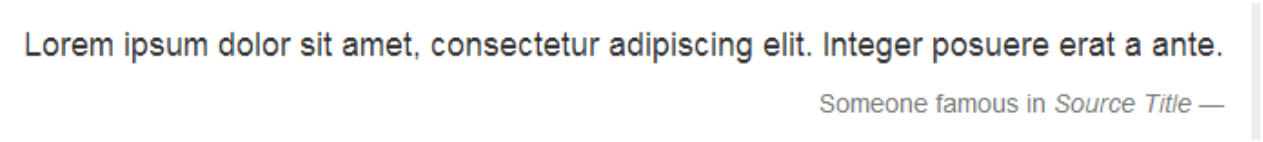
```
<blockquote>
  <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer posuere erat a ante.</p>
</blockquote>
```

Додайте `<footer>` для ідентифікації ресурсу. Обгорніть назву ресурсу в `<cite>`.

> Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer posuere erat a ante.
>
> — Someone famous in *Source Title*

```
<blockquote>
  <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer posuere erat a ante.</p>
  <footer>Someone famous in <cite title="Source Title">Source Title</cite></footer>
</blockquote>
```

Використовуйте `.blockquote-reverse` для вирівнювання цитати по правому краю.




```
<blockquote class="blockquote-reverse">
  ...
</blockquote>
```

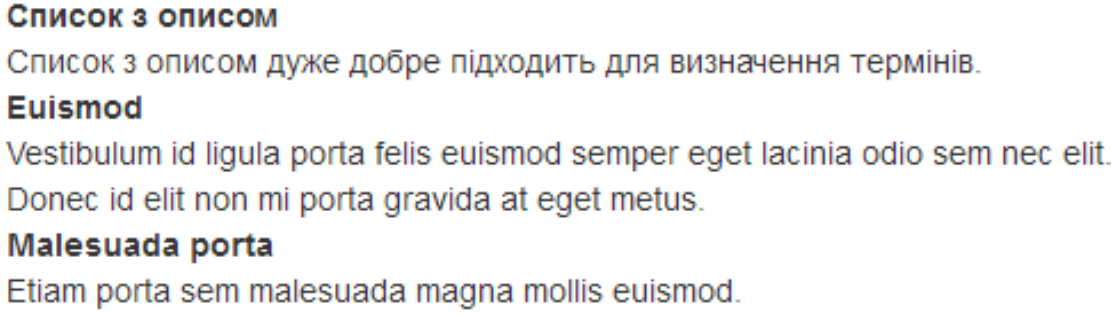
Списки

Без нумерації (список пунктів, коли порядок сортування немає явного значення) `<ul> <li>...</li> </ul>`

Із нумерацією (список пунктів, коли порядок сортування має явне значення) `<ol> <li>...</li> </ol>`

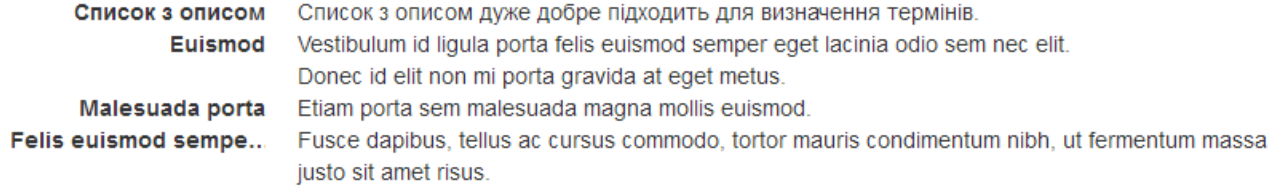
Опис

Список термінів та їх відповідних описів.




```
<dl>
  <dt>...</dt>
  <dd>...</dd>
</dl>
```

Горизонтальний опис – розташуйте терміни та їх описи пліч-о-пліч у лінію.



```
<dl class="dl-horizontal">
<dt>...</dt>
<dd>...</dd>
</dl>
```

Горизонтальний список опису буде автоматично звужуватись, якщо в ньому є надто довгі терміни, котрі не вміщаються в ліву колонку із `text-overflow`. У вузькому оглядовому вікні даний список прийме стандартний вигляд для `<dl>`.

Контекстні кольори і фони

Bootstrap також має деякі контекстні класи, які можуть використовуватися для надання «значення через кольори».

Класи для кольорів тексту

`.text-muted`, `.text-primary`, `.text-success`, `.text-info`, `.text-warning`, `.text-danger`, `.text-secondary`, `.text-white`, `.text-dark`, `.text-body` (колір тіла зазамовчуванням/найчастішечорний) і `.text-light`.

Цей текст вимкнено.

Цей текст важливий.

Цей текст свідчить про успіх.

Цей текст представляє деяку інформацію.

Цей текст представляє попередження.

Цей текст представляє небезпеку.

Вторинний текст.

Цей текст темно-сірий.

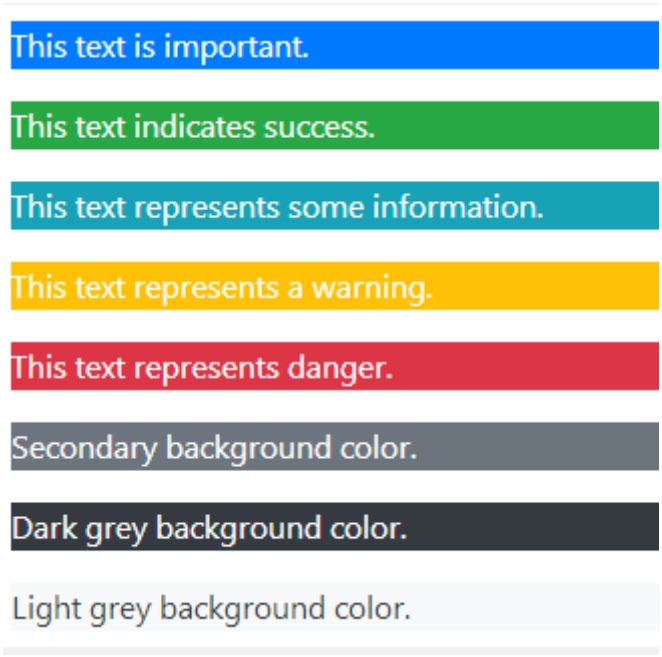
За замовчуванням колір тіла (часто чорний).

```
<p class="text-muted">Цей текст вимкнено.</p>
<p class="text-primary">Цей текст важливий.</p>
<p class="text-success">Цей текст свідчить про успіх.</p>
<p class="text-info">Цей текст представляє деяку інформацію.</p>
<p class="text-warning">Цей текст представляє попередження.</p>
<p class="text-danger">Цей текст представляє небезпеку.</p>
<p class="text-secondary">Вторинний текст.</p>
```

```
<p class="text-dark">Цей текст темно-сірий.</p>
<p class="text-body">За замовчуванням колір тіла (часто чорний).</p>
<p class="text-light">Цей текст світло-сірий (на білому фоні).</p>
<p class="text-white">Цей текст є білим (на білому фоні).</p>
```

Класи для кольорів фону

`.bg-primary`, `.bg-success`, `.bg-info`, `.bg-warning`, `.bg-danger`, `.bg-secondary`, `.bg-dark` і `.bg-light`.




```
<p class="bg-primary text-white">This text is important.</p>
<p class="bg-success text-white">This text indicates success.</p>
<p class="bg-info text-white">This text represents some information.</p>
<p class="bg-warning text-white">This text represents a warning.</p>
<p class="bg-danger text-white">This text represents danger.</p>
<p class="bg-secondary text-white">Secondary background color.</p>
<p class="bg-dark text-white">Dark grey background color.</p>
<p class="bg-light text-dark">Light grey background color.</p>
```

Таблиці

Для базового стилю — невеликих відступів та лише горизонтальних роздільних ліній — додайте базовий клас `.table` до будь-яких таблиць `<table>`.

```
<table class="table"> ... </table>
```

Використовуйте `.table-striped` щоб додати "смуги-зебри" до будь-яких рядків таблиці, котрі містять `<tbody>`.

```
<table class="table table-striped"> ... </table>
```

Додайте `.table-bordered` для розмежування всіх сторін таблиці, а також її комірок.

```
<table class="table table-bordered"> ... </table>
```

Додайте `.table-hover` щоб задіяти оголошення `hover` при наведенні на рядки таблиці в межах `<tbody>`.

```
<table class="table table-hover"> ... </table>
```

Додайте `.table-dark` щоб додати чорний фон в таблицю.

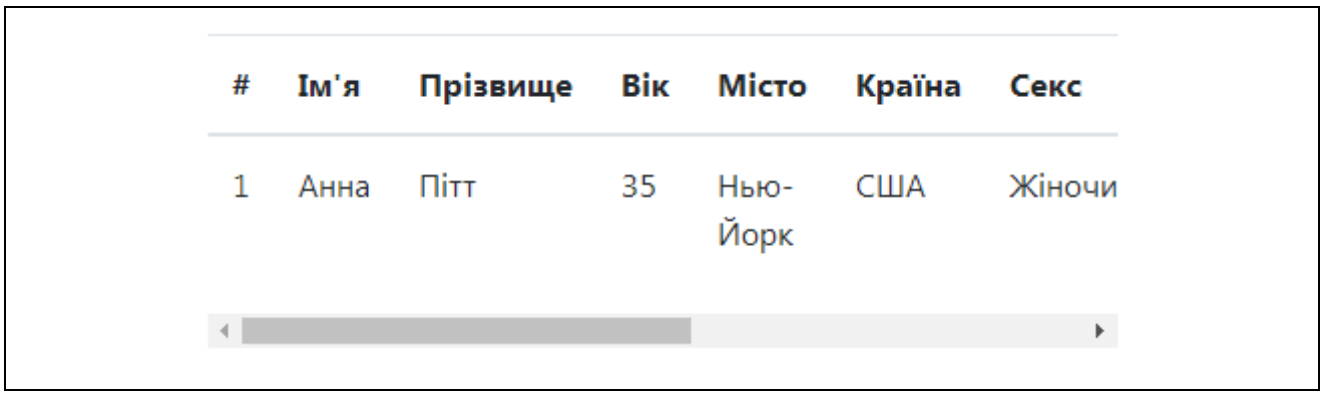
```
<table class="table table-dark"> ... </table>
```

Створюйте чутливі таблиці, обгортаючи будь-яку таблицю з класом `.table` в `.table-responsive` щоб створити в них горизонтальну лінію прокрутки для малих пристроїв (для 992 px). При перегляді на чомусь ширшому за 992 px, ви не побачите ніякої різниці між ними.

Можна також вказати яка таблиця повинна отримати полосу прокрутки в залежності від ширини екрану.

Клас	**Ширина екрана**
`.table-responsive-sm`	< 576 px
`.table-responsive-md`	< 768 px
`.table-responsive-lg`	< 992 px
`.table-responsive-xl`	< 1200 px

Приклад



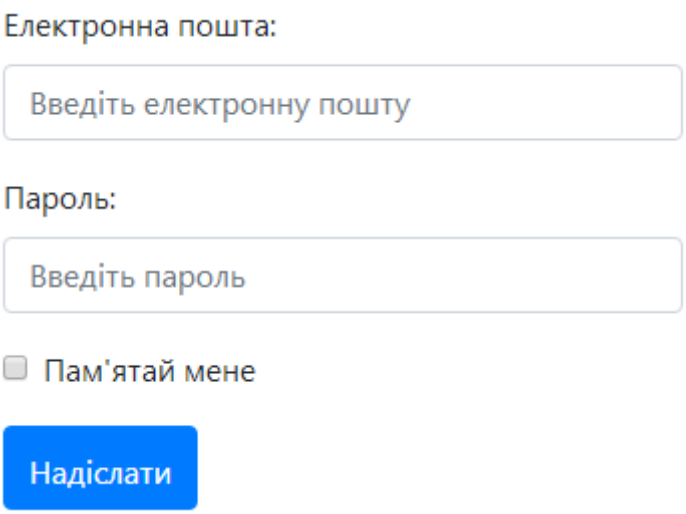
#	Ім'я	Прізвище	Вік	Місто	Країна	Секс
1	Анна	Пітт	35	Нью-Йорк	США	Жіночи

```
<div class="table-responsive-sm">
  <table class="table">
    ...
  </table>
</div>
```

Форми

Базовий приклад

Окремі елементи управління автоматично отримують певні глобальні налаштування. Всі текстові елементи `<input>`, `<textarea>`, та `<select>` з класом `.form-control` початково мають `width`: 100 %;. Обгорніть надписи та елементи керування в `.form-group` для оптимізації простору між ними.

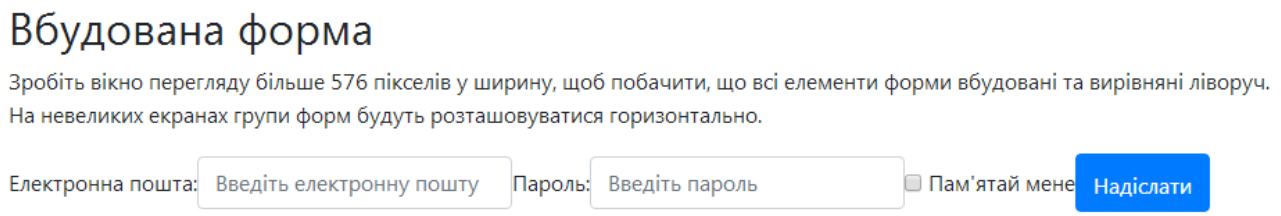



```
<form action="/action_page.php">
    <div class="form-group">
      <label for="email">Електронна пошта:</label>
      <input type="email" class="form-control" id="email" placeholder="Введіть
електронну пошту" name="email">
    </div>
```

```
    <div class="form-group">
      <label for="pwd">Пароль:</label>
      <input type="password" class="form-control" id="pwd" placeholder="Введіть пароль"
name="pswd">
    </div>
    <div class="form-group form-check">
      <label class="form-check-label">
        <input class="form-check-input" type="checkbox" name="remember"> Пам’ятай мене
      </label>
    </div>
    <button type="submit" class="btn btn-primary">Надіслати</button>
  </form>
```

Вбудована форма

Додайте `.form-inline` до `<form>` для вирівнювання по лівому краю та вставки блоку елементів управління форми. Це стосується лише форми в оглядовому вікні, з шириною щонайменше 576 px.



Горизонтальна форма

Використовуйте предвизначені класи в Bootstrap для вирівнювання ярликів та елементів управління в горизонтальному макеті форми, додаючи до цих форм клас `.form-horizontal`. Діючи таким чином, `.form-group` поводитиметься як рядки сітки, тобто відпадає потреба в `.row`.

Форми введення

Підтримуються найбільш поширені елементи управління форми та текстові поля введення. Також включено підтримку всіх типів поля HTML5: `text`, `password`, `datetime`, `datetime-local`, `date`, `month`, `time`, `week`, `number`, `email`, `url`, `search`, `tel`, та `color`.

Необхідно оголошувати тип поля. Форми введення матимуть весь набір стилів, лише якщо вони мають правильно оголошений `type`.

Textarea

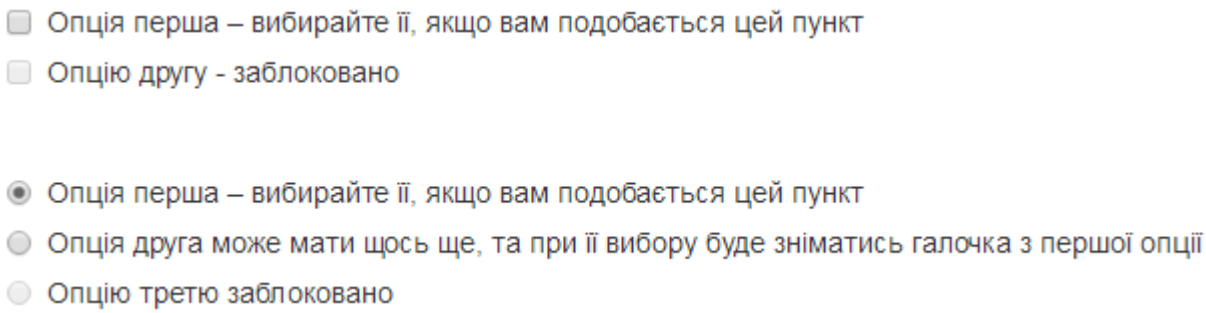
Textarea – це елемент управління форми, котрий підтримує багаторядкові лінії тексту. Змініть атрибут `rows` якщо необхідно.

```
<textarea class="form-control" rows="3"></textarea>
```

Галочки та перемикачі

Галочки призначені для позначення вибору одного або декількох елементів, у той час як перемикачі – можуть мати лише один вибраний елемент із декількох запропонованих.

Галочки чи перемикачі з атрибутом `disabled` будуть мати стиль заблокованого елемента. Щоб елементи `<label>` для галочок або перемикачів також відображали курсор зі значком "не-дозволено" при наведені на такий елемент, додайте клас `.disabled` до `.radio`, `.radio-inline`, `.checkbox`, `.checkbox-inline`, чи `<fieldset>`.



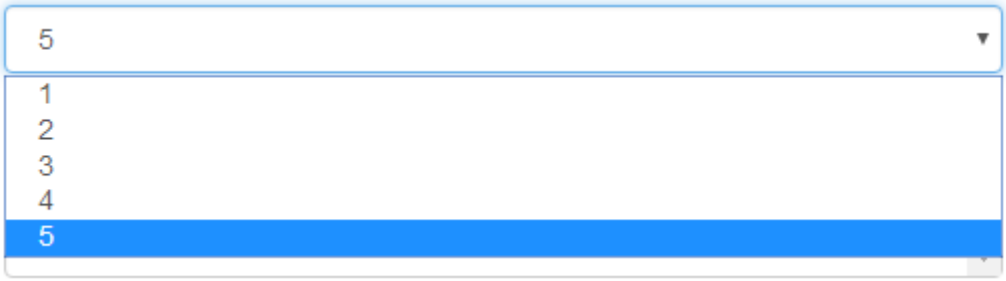

```
<div class="checkbox">
  <label>
    <input type="checkbox" value="">
    Опція перша – вибирайте її, якщо вам подобається цей пункт
  </label>
</div>
<div class="checkbox disabled">
  <label>
    <input type="checkbox" value="" disabled>
    Опцію другу – заблоковано
  </label>
</div>
<br>
<div class="radio">
  <label>
    <input type="radio" name="optionsRadios" id="optionsRadios1" value="option1"
checked>
    Опція перша – вибирайте її, якщо вам подобається цей пункт
  </label>
</div>
<div class="radio">
  <label>
    <input type="radio" name="optionsRadios" id="optionsRadios2" value="option2">
    Опція друга може мати щось ще, та при її вибору буде зніматись галочка з першої
опції
```

```
  </label>
</div>
<div class="radio disabled">
  <label>
    <input type="radio" name="optionsRadios" id="optionsRadios3" value="option3"
disabled>
    Опцію третю заблоковано
  </label>
</div>
```

Використовуйте клас `.checkbox-inline` або `.radio-inline` для серії полів галочок чи перемикачів, котрі відображаються в одну лінію.

Select

Використовуйте початкові опції, або додайте `multiple`, щоб мати можливість вибрати декілька пунктів.




```
<select class="form-control">
  <option>1</option>
  <option>2</option>
  <option>3</option>
  <option>4</option>
  <option>5</option>
</select>
```


```
<select multiple class="form-control">
  <option>1</option>
  <option>2</option>
  <option>3</option>
```

```
  <option>4</option>
  <option>5</option>
</select>
```

Заблоковані форми input

Додайте атрибут `disabled` до форми input для заборони введення даних користувачами.

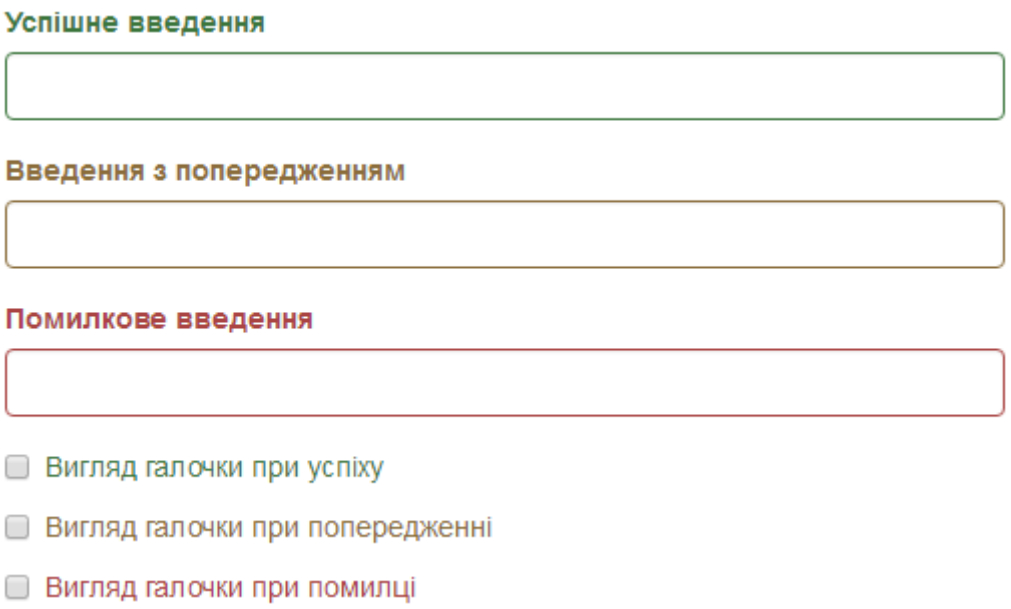
Додайте атрибут `disabled` до `<fieldset>` щоб заблокувати одразу всі елементи форми в середині `<fieldset>`.

Стан елемента input "тільки для читання"

Додайте логічний атрибут `readonly` в елемент `input`, щоб заблокувати введення у форму, та стилізувати елемент input як заблокований.

Стан пригодності

Bootstrap має стилі пригодності (`validation`) для елементів управління форми, із такими станами: успішна дія, попередження, помилка. Щоб ці стилі задіяти додайте `.has-warning`, `.has-error`, або `.has-success` до їх батьківських елементів. Будь-які `.control-label`, `.form-control`, та `.help-block` в середині цих елементів будуть відображати стани пригодності.




```
<div class="form-group has-success">
  <label class="control-label" for="inputSuccess1">Успішне введення</label>
  <input type="text" class="form-control" id="inputSuccess1">
</div>
<div class="form-group has-warning">
  <label class="control-label" for="inputWarning1">Введення з попередженням</label>
  <input type="text" class="form-control" id="inputWarning1">
```

```
</div>
<div class="form-group has-error">
  <label class="control-label" for="inputError1">Помилкове введення</label>
  <input type="text" class="form-control" id="inputError1">
</div>
<div class="has-success">
  <div class="checkbox">
    <label>
      <input type="checkbox" id="checkboxSuccess" value="option1">
      Вигляд галочки при успіху
    </label>
  </div>
</div>
<div class="has-warning">
  <div class="checkbox">
    <label>
      <input type="checkbox" id="checkboxWarning" value="option1">
      Вигляд галочки при попередженні
    </label>
  </div>
</div>
<div class="has-error">
  <div class="checkbox">
    <label>
      <input type="checkbox" id="checkboxError" value="option1">
      Вигляд галочки при помилці
    </label>
  </div>
</div>
```

Ви також можете за бажанням додати зправа значки відгуків за допомогою `.has-feedback`.

Значки відгуків працюють тільки з текстовими елементами `<input class="form-control">`.

Управління розміром

Встановлюйте висоту використовуючи такі класи як `.input-lg`, та встановлюйте ширину використовуючи класи колонок сітки, такі як `.col-lg-*`.

Кнопки

Для швидкого створення стилю кнопки є декілька спеціальних класів.


```
<button type="button" class="btn">Basic</button>
<button type="button" class="btn btn-primary">Primary</button>
<button type="button" class="btn btn-secondary">Secondary</button>
<button type="button" class="btn btn-success">Success</button>
<button type="button" class="btn btn-info">Info</button>
<button type="button" class="btn btn-warning">Warning</button>
<button type="button" class="btn btn-danger">Danger</button>
<button type="button" class="btn btn-dark">Dark</button>
<button type="button" class="btn btn-light">Light</button>
<button type="button" class="btn btn-link">Link</button>
```


```
<button type="button" class="btn btn-outline-primary">Primary</button>
<button type="button" class="btn btn-outline-secondary">Secondary</button>
<button type="button" class="btn btn-outline-success">Success</button>
<button type="button" class="btn btn-outline-info">Info</button>
<button type="button" class="btn btn-outline-warning">Warning</button>
<button type="button" class="btn btn-outline-danger">Danger</button>
<button type="button" class="btn btn-outline-dark">Dark</button>
<button type="button" class="btn btn-outline-light text-dark">Light</button>
```

Додавайте `.btn-lg`, `.btn-sm` щоб змінити їх розміри.

Створюйте кнопки для блочного розміщення, котрі заповнюють всю ширину батьківського елемента, за допомогою класу `.btn-block`.

Активний стан

Коли кнопка буде активною, вона матиме вигляд натиснутої (із затемненим фоном, темним обідком, та внутрішньою тінню). Для елемента <button>, це робиться за допомогою :active. Для елемента <a>, це робиться через .active. Однак, ви можете використовувати `.active` в `<button>` (та включити атрибут `aria-pressed="true"`) якщо вам потрібно повторити активний стан програмно.

Заблокований стан

Надайте кнопкам заблокованого вигляду через `opacity`. Додайте атрибут `disabled` до кнопок `<button>`.

Зображення

Закруглені кути

`.rounded` клас додає закруглені кути до зображення:

```
<Img src = "cinqueterre.jpg" class = "rounded" alt = "Cinque Terre">
```

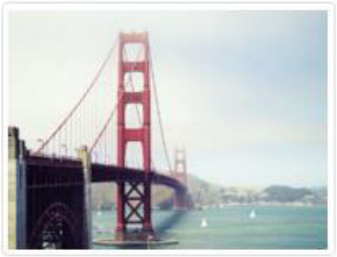
Коло

`.rounded-circle` клас формує зображення в коло:

```
<Img src = "cinqueterre.jpg" class = "rounded-circle" alt = "Cinque Terre">
```

Мініатюра

`.img-thumbnail` клас формує зображення на мініатюру (обрамлення):



```
<Img src = "cinqueterre.jpg" class = "img-thumbnail" alt = "Cinque Terre">
```

Вирівнювання зображень

`Float` зображення вправо з `.float-right` класом або вліво з `.float-left`:

```
<Img src = "paris.jpg" class = "float-left">
<Img src = "paris.jpg" class = "float-right">
```

Центроване зображення

Центрувати зображення можна додавши службові класи `.mx-auto` (Margin: Auto) і `.d-block (Display: Block)` до зображення:

```
<Img src = "paris.jpg" class = "mx-auto d-block">
```

Адаптивні зображення

Зображення бувають всіх розмірів, як і екрани. Адаптивні зображення автоматично коригуються відповідно до розміру екрана.

Адаптивне зображення створюється шляхом додавання `.img-fluid` класу до `<img>` тегу. Після цього зображення буде добре масштабуватися до батьківського елемента.

```
<img class="img-fluid" src="img_chania.jpg" alt="Chania">
```

З іншими можливостями Bootstrap можна ознайомитись за посиланням: https://html5css.ru/howto/howto_css_icon_bar.php

Питання для самоперевірки:

1. Пояснити, для чого використовується Bootstrap. Яка різниця між css-класами Bootstrap .container та .container-fluid.
2. Написати та надати надати опис css-класів для створення колонкового дизайну.
3. Пояснити поняття сітка Bootstrap. Навести приклад таблиці групування колонок (12 колонок, 3 одинакові колонки, одна менша та одна більша за шириною колонки, 2 одинакові колонки, 1 колонка), вписавши у кожну комірку цифру, що відповідає кількості колонок.
4. Навести деякі типографічні css-класи для текстових елементів у Bootstrap із коротким описом. Навести приклади.
5. Пояснити, яким чином зображення стають адаптованими у Bootstrap та навести приклади.
6. Надати короткий опис css-класів Bootstrap: .navbar, .navbar-expand {-sm| -md| -lg| -xl}, .navbar-brand, .navbarnav, .navbar-toggler, .navbar-tex, .navbar-collapse.
7. Надати коротку характеристику css-класів Bootstrap: .dropdown, .dropdown-menu, .dropdown-item, .dropdown-divider.
8. Надати опис css-класів Bootstrap: .carousel, .slide, .carousel-indicators, .carousel-inner, .carousel-control.

ДЖЕРЕЛА

1. Браун Е. Изучаем. Руководство по созданию современных веб-сайтов / Е. Браун. – М.: Диалектика, 2020. – 368 с.

2. Васильєв А.Н. Програмирование на JavaScript в примерах и задачах / А.Н. Васильев. – М.: Диалектика, 2016. – 720 с.

3. Никсон Р. Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5 / Р. Никсон. – С.-Пб.: Питер, 2015. – 688 с.

4. Прохоренок Н. HTML, JavaScript, PHP и MySQL. Джентльменский набор Web-мастера / Н. Прохоренок. – Санкт-Петербург: БХВ-Петербург, 2011. – 890 с.

5. Никсон Р. Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5 / Робин Никсон. – С.-Пб.: Питер, 2015. – 304 С.

6. Фленеган Д. JavaScript. Карманный справочник / Д. Флэнеган. - С.-Пб.: Питер, 2020. – 320 С.

7. JavaScript Tutorial [Електронний ресурс]. – URL: https://www.w3schools.com/js/default.asp

8. jQuery Tutorial [Електронний ресурс]. – URL: https://www.w3schools.com/jquery/default.asp

9. Bootstrap 4 Tutorial [Електронний ресурс]. – URL: https://www.w3schools.com/bootstrap4/default.asp