М.О. Слабінога

ВСТУП В СУЧАСНІ ВЕБ-ТЕХНОЛОГІЇ

ЛАБОРАТОРНИЙ ПРАКТИКУМ

Міністерство освіти і науки України

Івано-Франківський національний технічний університет нафти і газу

Кафедра комп'ютерних систем і мереж

М. О. Слабінога

ВСТУП В СУЧАСНІ ВЕБ-ТЕХНОЛОГІЇ

ЛАБОРАТОРНИЙ ПРАКТИКУМ

Івано-Франківськ

2019

УДК 004.514

С 47

Рецензент:

Я.І. Заячук кандидат технічних наук, доцент кафедри комп'ютерних систем і мереж Івано-Франківського національного технічного університету нафти і газу

Рекомендовано методичною радою університету
(протокол № від р.)

Слабінога М.О.

С 47. Вступ в сучасні веб-технології: лабораторний практикум / М.О. Слабінога – Івано-Франківськ: ІФНТУНГ, 2019. – 42 с.

МВ 02070855- -2019

Лабораторний практикум з дисципліни "Вступ в сучасні веб-технологі" розроблений відповідно до робочої програми навчальної дисципліни та робочого навчального плану.

Призначено для підготовки бакалаврів за спеціальністю 123 - "Комп'ютерна інженерія". Лабораторний практикум може бути використаний студентами очної та заочної форм навчання.

УДК 004.514
С 47

Рецензент:

Заячук Я.І. кандидат технічних наук, доцент кафедри комп'ютерних систем і мереж Івано-Франківського національного технічного університету нафти і газу

Рекомендовано методичною радою університету
(протокол № від)

Слабінога М.О.
С 47. Вступ в сучасні веб-технології: лабораторний практикум / М.О. Слабінога – Івано-Франківськ: ІФНТУНГ, 2019. – 42 с.
МВ 02070855- -2019

Лабораторний практикум з дисципліни "Вступ в сучасні веб-технологі" розроблений відповідно до робочої програми навчальної дисципліни та робочого навчального плану.

Призначено для підготовки бакалаврів за спеціальністю 123 - "Комп'ютерна інженерія". Лабораторний практикум може бути використаний студентами очної та заочної форм навчання.

УДК 004.514

МВ 02070855- -2019

Завідувач кафедри
комп'ютерних систем і мереж М.І. Горбійчук

Узгоджено:
Член експертно-рецензійної
комісії університету В.В. Бандура

Нормоконтролер Г. Я. Томашівська

Провідний бібліотекар НТБ Г. М. Мацюк

ЗМІСТ

ЗАГАЛЬНІ ВКАЗІВКИ ДО ВИКОНАННЯ ЛАБОРАТОРНИХ РОБІТ

Метою навчальної дисципліни є засвоєння необхідних знань з основ веб-технологій, а також формування твердих практичних навичок щодо розробки якісного програмного забезпечення мовою Javascript.

Для досягнення мети поставлені такі основні завдання:

- отримання теоретичних знань з основ веб-технологій, веб-дизайну та веб-програмування мовою Javascript;
- отримання практичних навичок з розробки програмного забезпечення, що виконується браузером на стороні клієнта.

У результаті вивчення навчальної дисципліни студент повинен:

знати:

- правила побудови документів HTML та включення до них скриптів мовою Javascript;
- синтаксичні структури мови Javascript та її можливості для розробки клієнтської частини веб-додатків;
- основні елементи об’єктної моделі браузера;
- концепцію функціонального та прототипного підходу до розробки ПЗ мовою Javascript.

вміти:

- керувати структурою HTML-сторінки методами Javascript та супутніми бібліотеками

- отримувати та опрацьовувати дані, що надаються серверною частиною програмного забезпечення з допомогою AJAX-запитів
- застосовувати комплексний підхід до розробки програмного забезпечення мовою Javascript.

Вимоги до програмного забезпечення ПК для виконання лабораторних робіт:

- Браузер (Mozilla Firefox, Opera, Google Chrome);
- Текстовий редактор (Sublime, Visual Studio Code, Atom, Notepad++);
- Git.
- Доступ до мережі Internet.

Програмний код всіх робіт повинен бути розміщений на репозиторії Github або Bitbucket.

ЛАБОРАТОРНА РОБОТА № 1

Основи мови Javascript

Мета роботи: ознайомитися з базовими конструкціями мови Javascript.

Теоретичні відомості

JavaScript (JS) — динамічна, об'єктно-орієнтована прототипна мова програмування. Реалізація стандарту ECMAScript. Найчастіше використовується для створення сценаріїв веб-сторінок, що надає можливість на стороні клієнта (пристрої кінцевого користувача) взаємодіяти з користувачем, керувати браузером, асинхронно обмінюватися даними з сервером, змінювати структуру та зовнішній вигляд веб-сторінки.

JavaScript класифікують як прототипну (підмножина об'єктно-орієнтованої), скриптову мову програмування з динамічною типізацією. Окрім прототипної, JavaScript також частково підтримує інші парадигми програмування (імперативну та частково функціональну) і деякі відповідні архітектурні властивості, зокрема: динамічна та слабка типізація, автоматичне керування пам'яттю, прототипне наслідування, функції як об'єкти першого класу.

Мова JavaScript використовується для:

- написання сценаріїв веб-сторінок для надання їм інтерактивності;
- створення односторінкових веб-застосунків (ReactJS, AngularJS, Vue.js);
- програмування на стороні сервера (Node.js);
- стаціонарних застосунків (Electron, NW.js);
- мобільних застосунків (React Native, Cordova);

- сценаріїв в прикладному ПЗ (наприклад, в програмах зі складу Adobe Creative Suite чи Apache JMeter);
- всередині PDF-документів тощо.

Незважаючи на схожість назв, мови Java та JavaScript є двома різними мовами, що мають відмінну семантику, хоча й мають схожі риси в стандартних бібліотеках та правилах іменування. Синтаксис обох мов отриманий «у спадок» від мови C, але семантика та дизайн JavaScript є результатом впливу мов Self та Scheme.

У листопаді 1996 року Netscape заявила, що відправила JavaScript в організацію Ecma International для розгляду мови як промислового стандарту. В результаті подальшої роботи з'явилась стандартизована мова з назвою ECMAScript. У червні 1997 року, Ecma International опублікувала першу редакцію специфікації ECMA-262. Рік по тому, у червні 1998 року, щоб адаптувати специфікацію до стандарту ISO/IEC-16262, були внесені деякі зміни і випущена друга редакція. Третя редакція побачила світ в грудні 1999 року.

Четверта версія стандарту ECMAScript так і не була закінчена і четверта редакція не вийшла. Тим не менш, п'ята редакція з'явилася в грудні 2009 року.

У червні 2015 року вийшла шоста версія, починаючи з якої комітет ECMAScript прийняв рішення перейти на щорічні оновлення і нова версія отримала назву ES2015. Вона отримала цілу низку нововведень, серед яких: об'єкт Promise для зручного асихронного виконування коду, деструктуруюче присвоювання, стрілочні функції, функції-генератори, шаблонні рядки, оператори оголошення змінних let та const тощо.

Версія ES2016 вийшла у червні 2016 року, серед нововведень оператор піднесення до степеня ** та метод Array.prototype.includes, який перевіряє, чи міститься переданий аргумент в масиві.

Версія ES2017, що вийшла в червні 2017 року і на сьогодні є актуальною версією стандарту додала можливість використання асихронних функцій, «висячих» ком в параметрах функцій, об'єкт Atomics, декількох нових методів для роботи з рядками.

JavaScript, наразі, є однією з найпопулярніших мов програмування в інтернеті. Але спочатку багато професійних програмістів скептично ставилися до мови, цільова аудиторія якої складалася з програмістів-любителів. Поява AJAX змінила ситуацію та повернула увагу професійної спільноти до мови, а подальші модифікації мови за стандартами ES2015 та ES2017 внесли багато корисних можливостей, яких не вистачало для ефективного програмування. В результаті, були розроблені та покращені багато практик використання JavaScript (зокрема, тестування та налагодження), створені бібліотеки та фреймворки, поширилося використання JavaScript поза браузером.

Завдання на лабораторну роботу

1. Створіть пусту HTML-сторінку. Виведіть з допомогою alert() власне ім’я. Винесіть скрипт в окремий файл. Оголосіть змінну name, збережіть в ній власне ім’я та виведіть з допомогою alert()

2. Створіть змінні a = (ваш номер по списку) і b = (номер місяця вашого народження. Віднімiть від a змінну b і результат надайте змінної c. Потім

створіть змінну d, надайте їй значення 7. Додайте змінні c і d, а результат запишіть в змінну result. Виведіть на екран значення змінної result.

3. Введіть з допомогою prompt номер вашого варіанту. Якщо змінна більше або дорівнює 7, то виведіть 'Вірно', інакше виведіть 'Невірно'. Якщо змінна a ділиться на 2, то додайте до неї 7, інакше відніміть 3. Виведіть нове значення змінної на екран. Перевірте роботу скрипта при a, що дорівнює 5, 0, -3, 2.

4. В стовпчик виведіть квадрати чисел від від 1 до (номеру вашого варіанту +10).

5. Виведіть всі прості числа в діапазоні від 2 до (номеру вашого варіанту +10).

Контрольні запитання

1. Перелічіть діалогові вікна, що викликаються Javascript.
2. Перелічіть види циклів в Javascript.
3. Опишіть принцип роботи циклів з передумовою та післяумовою.
4. Опишіть процедуру роботи структури Switch.
5. Перелічіть типи даних в Javascript.

ЛАБОРАТОРНА РОБОТА № 2

Робота з функціями в Javascript

Мета роботи: ознайомитися з принципами роботи функцій в Javascript.

Теоретичні відомості

Якщо у нас є певна стала часто повторювана дія, краще зробити її функцією яку ми можемо викликати кожного разу із різних частин сайту, а при певних змінах достатньо буде змінити сам вміст однієї функції, а не переглядати весь код для пошуку коду в якому треба змінити розрахунки тощо.

Оголошення функції:

```
function helloWorld(){
    alert('Hello, World!');
}
```

- function - оголошує нашу функцію;
- helloWorld () - назва створюваної функції. В дужках вказуються аргументи, які буде приймати функція, за потреби;

Примітка: ім'я функції так як і змінною повинно передавати її зміст, як наприклад наша функція calcTax(наведена нижче) розраховує податок (ПДВ)

Тіло функції загортаємо в фігурні дужки "{}", задають початок та кінець функції.

Розрахуємо суму ПДВ від введених користувачем даних:

```
<script type="text/javascript">
     function calcTax(sum){
          alert('Розмір ПДВ від суми ' + sum + ' = ' + sum
* 0.2);
     }
```

```
</script>
<input type="button" value="DEMO"
onclick="calcTax(prompt('Введіть суму для обчислення ПДВ:',
''))">
```

Створюючи функцію, їй можна задати перелік аргументів, через кому, який вона може приймати:

```
function test(x, y, z){
    [щось тут робить]
}
```

В JS є певні особливості при передачі аргументів для функції:

- якщо Ви передаєте на функцію більше аргументів ніж вони може прийняти, то всі передані дані що виходять поза діапазон не будуть оброблятись;

- якщо ви передаєте функції менше аргументів ніж вона від Вас вимагає, то всі не вказані аргументи приймуть значення Undefined.

Якщо Ви пропустите один із аргументів який потрібен для розрахунку, Вас про це не повідомлять (як в інших мовах програмування).

Можна створювати функції з необов'язковими аргументами і передавати їх лише за потреби.

Присвоєння функції - звичайне присвоєння значення для змінної, де значення, яке отримує змінна, є функцією.

Перебудуємо трошки нашу функцію calcTax і отримаємо:

```
<script type="text/javascript">
     var payTax = function calcTax(sum){
```

```
            return 'ПДВ до оплати: ' + sum * 0.2;
        };
    </script>
    <input type="button" value="DEMO"
    onclick="alert(payTax(prompt('Введіть суму для обчислення
    ПДВ:', '')))">
```

- prompt - запитує в користувача ввід значення
- payTax - приймає введене значення і передає його на присвоєну функцію calcTax
- calcTax - в свою чергу розраховує суму ПДВ і за допомогою оператора return повертає стрічку результат яку і виводить alert

Примітка: якщо потрібно здійснювати додаткові операції із даними то їх потрібно проводити до оператора return так як при його виконанні функція буде завершена!

Ось для прикладу alert в середині функції ніколи не буде виконаний:

```
    <script type="text/javascript">
        var payTax = function calcTax(sum){
            return 'ПДВ до оплати: ' + sum * 0.2;
            alert("Мене все рівно ніхто не прочитає!");
        };
    </script>
    <input type="button" value="DEMO"
onclick="alert(payTax(prompt('Введіть суму для обчислення ПДВ:',
'')))">
```

Локальна змінна - змінна яка оголошується в тілі функції, чи передана їй аргументом, і може використовуватись лише в межах даної функції

```
function test1(){
```

```
    var x = 2;
    alert('x1 = ' + x);
};
alert('x2 = ' + x);
```

x2 - не буде виведено так як він пробує відобразити не існуючий x, натомість x1 буде добре виводитись на екран так як змінна оголошена в тій же процедурі.

Якщо винести оголошення x за межі функції:

```
var x = 2;
function test1(){
    alert('x1 = ' + x);
};
alert('x2 = ' + x);
```

то будуть виводитись обидва повідомлення.

Завдання на лабораторну роботу

1. Напишіть функцію, що повертає куб числа. Протестуйте її на номері вашого варіанту

2. Напишіть функцію, що додає перше число до другого і ділить результат на третє число. Протестуйте її на наступних значеннях:

1 число: дата вашого народження

2 число: місяць вашого народження

3 число: номер вашого варіанту

3. Напишіть функцію, що приймає число від 1 до 7 і повертає відповідну назву дня (українською).

4. Перепишіть третє завдання, використавши функціональний вираз

5. Реалізуйте двома способами (з допомогою циклу та рекурсії) функцію:

- парні варіанти - факторіал
- непарні варіанти - піднесення до степеня.

Контрольні запитання

1. Наведіть приклади трьох видів оголошення функцій.
2. Опишіть поведінку локальних та глобальних змінних.
3. Як впливає на поведінку локальних та глобальних змінних вживання з var та без?
4. Що таке рекурсія?
5. Опишіть поведінку функцій при роботі з різною кількістю аргументів.

ЛАБОРАТОРНА РОБОТА № 3

Робота з DOM. Бібліотека jQuery

Мета роботи: ознайомитися з інструментарієм DOM, раннім циклом роботи сторінки, а також засобами роботи з DOM, якою володіє бібліотека jQuery..

Теоретичні відомості

DOM - об'єктна модель документа (Document Object Model). DOM це зовсім інше представлення веб-сторінки ніж HTML код. Браузер по вказаній URL адресі відправляє запит і отримує (завантажує) з сервера веб-сторінку у вигляді HTML коду, який часто називається вихідний код сторінки. І якщо у коді вказані інші файли такі як стилі css, js - то завантажує і їх. І уже з завантаженого з сервера HTML коду браузер формує документ - DOM.
Браузер створює DOM для того щоб відобразити веб-сторінку на екрані. Вигляд DOM документа можна глянути у панелі розробника в браузері. DOM схожий на вихідний код HTML але не є ним, а лише формується з нього.

Браузер автоматично виправляє помилки якщо вони є у HTML коді. Тобто закриває не закриті теги HTML, вставляє обов'язкові теги якщо вони опущені.

DOM має деревоподібну ієрархію. Документ DOM складається з вузлів Node. Кожен вузол може містити у собі вбудований вузол, елемент, текст чи коментар.

Кожен кожен вузол DOM формується з HTML тегу і отримує властивості, події, стилі які вказані у самих атрибутах тегу, CSS стилях і в JavaScript коді. DOM підтримує об'єктно орієнтоване представлення

веб-сторінки і дозволяє змінювати документ веб-сторінки за допомогою JavaScript.

Для роботи з DOM у JavaScript є об'єкт document, який дозволяє:

- видалити HTML-елементи і атрибути;
- змінити всі HTML-елементи на сторінці;
- змінити всі атрибути HTML на сторінці;
- змінювати всі стилі CSS на сторінці;
- додавати нові елементи HTML і атрибути;
- створювати нові події на сторінці;
- реагувати на існуючі події на сторінці;

У JavaScript для роботи з DOM є об'єкт document, який містить методи і властивості для роботи з документом.

jQuery — популярна JavaScript-бібліотека з відкритим сирцевим кодом. Вона була представлена у січні 2006 року у BarCamp NYC Джоном Ресіґом (John Resig). Згідно з дослідженнями організації W3Techs, JQuery використовується понад половиною від мільйона найвідвідуваніших сайтів. jQuery є найпопулярнішою бібліотекою JavaScript, яка посилено використовується на сьогоднішній день.

jQuery є вільним програмним забезпеченням під ліцензією MIT (до вересня 2012 було подвійне ліцензування під MIT та GNU General Public License другої версії).

Синтаксис jQuery розроблений, щоб зробити орієнтування у навігації зручнішим завдяки вибору елементів DOM, створенню анімації, обробки подій, і розробки AJAX-застосунків. jQuery також надає можливості для розробників, для створення плагінів у верхній частині бібліотеки JavaScript.

Використовуючи ці об'єкти, розробники можуть створювати абстракції для низькорівневої взаємодії та створювати анімацію для ефектів високого рівня. Це сприяє створенню потужних і динамічних веб-сторінок.

Завдання на лабораторну роботу

1. За посиланням http://138.68.105.99/moodle/mod/book/view.php?id=181 виберіть індивідуальне завдання.
2. Зверстайте головну таблицю завдання.
3. Зверстайте форму додавання нового запису в таблицю та кнопку “Додати запис”
4. Створіть в таблиці ще один стовпчик та додайте туди кнопку “Видалити” до кожного рядка.
5. Реалізуйте функцію додавання та видалення з допомогою DOM.

Контрольні запитання

1. Опишіть поетапно ранній життєвий цикл сторінки.
2. Опишіть існуючі функції селекторів в DOM.
3. Перелічіть можливості селекторів в jQuery.
4. Перелічіть можливості BOM.
5. Опишіть порядок підключення та звернення до jQuery.

ЛАБОРАТОРНА РОБОТА № 4

AJAX. Методи роботи з AJAX в jQuery

Мета роботи: Ознайомитися з принципами розробки клієнт-серверних додатків з використанням AJAX-підходу.

Теоретичні відомості

AJAX (Asynchronous JavaScript And XML) — підхід до побудови користувацьких інтерфейсів веб-застосунків, за яких веб-сторінка, не перезавантажуючись, у фоновому режимі надсилає запити на сервер і сама звідти довантажує потрібні користувачу дані. AJAX — один з компонентів концепції DHTML.

Про AJAX заговорили після появи в лютому 2005-го року статті Джесі Джеймса Гарретта (Jesse James Garrett) «Новий підхід до веб-застосунків». AJAX — не самостійна технологія. Це ідея.

AJAX — це не самостійна технологія, а швидше концепція використання декількох суміжних технологій. AJAX-підхід до розробки, який призначений для користувачів інтерфейсів, комбінує кілька основних методів і прийомів:

- Використання DHTML для динамічної зміни змісту сторінки.
- Використання XMLHttpRequest для звернення до сервера «на льоту», не перезавантажуючи всю сторінку повністю
- альтернативний метод — динамічне підвантаження коду JavaScript в тег <SCRIPT> з використанням DOM, що здійснюється із використанням формату JSON)
- динамічне створення дочірніх фреймів

Використання цих підходів дозволяє створювати набагато зручніші веб-інтерфейси користувача на тих сторінках сайтів, де необхідна активна взаємодія з користувачем. AJAX — асинхронний, тому користувач може переглядати далі контент сайту, поки сервер все ще обробляє запит. Браузер не перезавантажує web-сторінку і дані посилаються на сервер без візуального підтвердження (крім випадків, коли ми самі захочемо показати процес з'єднання з сервером). Використання AJAX стало популярним після того, як компанія Google почала активно використовувати його при створенні своїх сайтів, таких як Gmail, Google Maps і Google Suggest. Створення цих сайтів підтвердило ефективність використання даного підходу.

Класична модель веб-застосунку:

- Користувач заходить на веб-сторінку і натискає на який-небудь її елемент
- Браузер надсилає запит серверу
- У відповідь сервер генерує повністю нову веб-сторінку і відправляє її браузеру і т. д.
- З боку сервера можлива генерація не всієї сторінки наново, а тільки деяких її частин, з подальшою передачею користувачу.

Модель AJAX:

- Користувач заходить на веб-сторінку і натискає на який-небудь її елемент.
- Браузер відправляє відповідний запит на сервер.
- Сервер віддає тільки ту частину документа, яка змінилася.

В деяких застосунках використовуються певні варіації з форматом відповіді сервера, такі варіації набули напівофіційні назви.

AHAH (Asynchronous HTML and HTTP) — це споріднений AJAX підхід для динамічного оновлення веб-сторінок, використовуючи JavaScript. Основною його відмінністю від AJAX є те, що відповіді сервера повинні бути звичайним HTML.[1] Перевага підходу полягає в більшій сумісності і функціональності (підтримка навігаційних кнопок браузера, аплоад файлів тощо). Реалізується у вигляді звичайних фреймів, що автоматично міняють свій розмір під розмір вмісту, або у вигляді прихованих фреймів, що виконують тільки функції завантаження даних.

Asynchronous XHTML and HTTP, або абревіатура AXAH — це майже те ж саме що і AHAH. Різниця тільки в тому, що в AHAH сервер клієнтові повертає HTML, а в AXAH вже XHTML.

Розглянемо реалізацію AJAX запитів за допомогою jQuery. Для роботи з AJAX в jQuery передбачено три функції:

$.ajax() - відправляє GET / POST запити

$.get() - відправляє GET запити

$.post() - відправляє POST запити

Функція $.get() вже була розглянута на початку цього курсу в якості прикладу. Першим аргументом вказується адреса ресурсу і параметри (оскільки це GET, Параметри записуються в URL). Другий аргумент - це анонімна функція, яка залежить від тіла відповіді сервера. Як бачимо, все набагато простіше, ніж чистий JS. Замість другого аргументу (для обробки відповіді) можна використовувати метод done().

Функція $.post() відправляє POST запит. Ця функція приймає три аргументи: адреса ресурсу, об'єкт з даних, які є тілом запиту (POST параметри) і функція для обробки тіла відповіді.

Ці функції є спрощеними варіантами функції $.ajax(), Яка приймає безліч параметрів для більш тонкої настройки запиту.

Розглянемо приклад з функцією $.post(), В якому запишемо дані в текстовий файл.. Все, що нам треба - переписати обробку при натисканні на кнопку на сторінці з текстовим полем:

```
<textarea id="textField" name="text"></textarea> <br/>
<input type="button" id="btn" value="Go"/>

<script>
    $(document).ready(function() {
        $('#btn').click(function() {
            var textValue = $('#textField').val();
            $.post('/test.php',{text: textValue});
        });
    });
</script>

</body>
</html>
```

Тут

- чекаємо поки документ буде повністю завантажений $(document).ready(...)
- обробка кліка по кнопці $('#btn').click(...)
- визначення значення в текстовому полі і відправка запиту $.post(...)

Другим аргументом $.post() вказані параметри запиту - об'єкт, який трансформується в масив $_POST на сервері.

Завдання на лабораторну роботу

Реалізуйте завдання з попередньої лабораторної роботи, відправляючи дані про додавання, видалення та редагування закладок запитами на серверну сторінку, адресу якої надасть викладач.

Контрольні запитання

1. Опишіть класичний підхід до розробки веб-додатків.
2. Опишіть підхід до розробки веб-додатків з використанням AJAX.
3. Опишіть переваги AJAX над класичним підходом.
4. Перелічіть засоби DOM для роботи з AJAX.
5. Перелічіть засоби jQuery для роботи з AJAX.

ЛАБОРАТОРНА РОБОТА № 5

Робота з об’єктами та масивами в Javascript

Мета роботи: ознайомитися з функціями роботи з об’єктами та масивами, а також взаємодією об’єктів та їх відображення через jQuery.

Теоретичні відомості

Усі елементи-об'єкти web-сторінки утворюють ієрархічну структуру. Кожний такий об'єкт має певні властивості й методи. Для керування об'єктами web-сторінки мовою JavaScript потрібно усвідомлювати цю ієрархію.

Вікно проглядача є певним об'єктом window у мові JavaScript. Цей об'єкт містить певні елементи оформлення, наприклад, рядок стану. Всередині вікна розташовано документ HTML (або файл іншого типу — поки обмежимося файлами HTML) — об'єкт document. До властивостей цього об'єкту відносять, наприклад, колір тла web-сторінки. Але для нас важливіше те, що всі об'єкти HTML (посилання, форма тощо) є властивостями об'єкта document.

Коли загальна кількість об'єктів невелика, таку ієрархію подають орієнтованим деревом, у якому вершини (точки площини) є об'єктами, а дуги чи орієнтовані ребра (спрямовані відрізки) вказують на об'єкти, що є властивостями тих об'єктів, від яких вони спрямовані. Але коли кількість об'єктів зростає, постають проблеми розташування елементів такого подання та його сприйняття в цілому. У той самий час доволі просто таку структуру подавати текстом, у якому: кожний рядок подає певний об'єкт, всі властивості якого або властивості властивостей і т.д. розташовують під цим об'єктом з зсувом від початку рядку (див. вище).

Опишемо, як у мові JavaScript влаштований доступ до різних об'єктів. Кожний об'єкт ієрархічної структури має свою назву. Її побудову потрібно починати з кореня — першого об'єкту з назвою document. Далі потрібно через крапку записати назви всіх об'єктів, які зустрічаються на шляху цього об'єкту від document, і його власну назву.

Наприклад, перше зображення на сторінці має назву images[0]. Доступ до цього об'єкту можна отримати, записавши в JavaScript таке: document.images[0].

Звернення до тексту, який користувач ввів у перший елемент форми, отримаємо, записавши: document.forms[0].elements[0]. Однак таке звернення ще не дає доступу до власне тексту. Для цього, як можна встановити з довідників, потрібно використати властивість value цього тексту. Остаточно маємо: document.forms[0].elements[0].value. Саме таким чином можна використати у подальшому текст, введений користувачем у відповідне поле форми. Наприклад, внесенні слова «Петро» при подальшому натисканні на кнопку «Натисни на мене» випаде вікно зі зверненням «Все гаразд, Петро».

Для того, щоб не плутатися в адресації при роботі з великою кількістю об'єктів, радять використовувати унікальні назви. Як це роблять, можна побачити на поданому вище прикладі — див. рядок <form name="myForm">. Цей рядок означає: об'єкт forms[0] отримує другу назву — myForm. Аналогічно, замість elements[0] можна писати name (вказано в атрибуте name тегу <input>). Інакше кажучи, замість document.forms[0].elements[0].value можемо писати document.myForm.name.value .

Зауважте: при написанні назв Ви маєте пильнувати за станом регістру. Наприклад, назви myform і myForm — це різні назви.

У JavaScript багато властивостей доступні не лише для читання. Ви можете надавати їм нових величин. Наприклад, замість порожнього рядка, що задає властивість value можна записати довільну послідовність символів.

Об'єкти JavaScript — це набори властивостей і методів. Властивості об'єктів — це дані, пов'язанні з об'єктом, а методи — функції для опрацювання даних об'єкта. Як вже було показано, адресація властивостей у сценаріях JavaScript можлива або за назвами, або за номерами. Останнє можливе, бо всі властивості об'єкта зберігають як елементи певного масиву, тому кожна властивість має свій.

У мові JavaScript є три типи об'єктів: вбудовані об'єкти, об'єкти проглядача й об'єкти на основі класів, створених у ході виконання програми. Кожний з цих типів має свої призначення й особливості.

Перелічимо вбудовані об'єкти, властивості й методи яких доступні у сценаріях JavaScript без попереднього означення цих об'єктів:

Array — масив;

Boolean — логічні дані;

Date — календарна дата;

Function — функція;

Global — глобальні методи;

Math — математичні сталі й функції;

Number — числа;

Object — об'єкт;

String — рядок.

Вбудовані об'єкти зручні для виконання рутинних операцій з рядками, датами, масивами, числами і т.і.

У сценарії JavaScript проглядач подано ієрархічно організованим набором об'єктів. Звертаючись до властивостей і методів цих об'єктів, можна здійснювати різні операції з вікном, завантаженим у це вікно документом HTML, а також з окремими об'єктами, розташованими в документі HTML. Наприклад, можна створити нові вікна проглядача, завантажити у них документи, динамічно формувати текст документу HTML, звертатися до полів форм і т.і.

Об'єкти проглядача є тим інтерфейсом, з допомогою якого взаємодіють сценарій JavaScript, користувач, проглядач і документ HTML, завантажений у вікно проглядача.

Об'єкт window знаходиться в корені ієрархії. При завантаженні документу HTML всередині цього об'єкту сворюються інші об'єкти: document, parent, frame, location і top.

Якщо у вікно проглядача завантажено документ HTML з фреймами, то для кожного фрейма створюється окреме вікно як об'єкт window.

Об'єкт document містить у собі інші об'єкти, склад яких визначено завантаженим документом HTML. Це можуть бути форми, посилання на інші документи HTML або локальні посилання всередині одного документа, об'єкти, що визначають адресу URL документа тощо.

Якщо документ містить форми, то вони також є ієрархічними наборами об'єктів. Об'єкт-форма може містити у собі такі об'єкти: кнопки, перемикачі, поля для вводення текстової інформації тощо (див. рисунок вище).

Звертаючись до властивостей перелічених об'єктів, сценарій JavaScript може визначити всі характеристики документа HTML.

Завдання на лабораторну роботу

1. Згідно завдання, вибраного в роботі 3, реалізуйте таблицю з діями щодо записів, використовуючи об’єкти.
2. Окрім раніше реалізованих функцій додавання та видалення, реалізуйте функцію редагування.
3. Реалізуйте додавання, редагування та видалення зв’язаних даних.
4. Доповніть раніше зроблені AJAX-запити новими, згідно попередніх пунктів.

Контрольні запитання

1. Опишіть стандартні об’єкти в Javascript.
2. Опишіть доступ до властивостей об’єкта Javascript через крапкову та дужкову нотацію.
3. Опишіть роботу функції filter.
4. Опишіть стандартні методи роботи з числами та рядками.
5. Опишіть стандартні методи роботи з датою та часом.

ЛАБОРАТОРНА РОБОТА № 6

Функціональний та прототипний підхід до розробки ПЗ в Javascript. Стандартизація Javascript. EcmaScript.

Мета роботи: ознайомитися з різними підходами до розробки програмного забезпечення мовою Javascript.

Теоретичні відомості

Ненав'язливий JavaScript (англ. unobtrusive JavaScript) —загальний підхід до web-програмування мовою JavaScript. Термін було запроваджено 2002-го року Стюартом Ленгріджем. Принципами ненав'язливого Javascript зазвичай вважаються такі:

- відділення функціональності веб-сторінки («рівень поведінки») від структури, змісту і представлення веб-сторінки;

- відпрацьовані методи із уникнення проблем традиційного програмування мовою JavaScript (таких як залежність від браузера і нестача масштабованості);

- техніки «поступового покращення» (англ. Progressive enhancement) для підтримки користувацьких агентів, що можуть не повністю підтримувати функції JavaScript.

Через несумісність реалізацій мови і Document Object Model у різних браузерах JavaScript мав репутацію мови, що не підходила до серйозного використання і розвитку. Поява браузерів, що притримувались стандартів, поява інтерфейсів AJAX та веб 2.0 змінило ситуацію, зробивши JavaScript необхідним інструментом. Якщо раніше JavaScript використовували для порівняно простих і несуттєвих завдань, таких як перевірка форм на стороні

браузера або декоративні елементи оформлення сторінок, то згодом він став використовуватись для створення основної функціональності сайтів.

Працездатність веб-сайту для найбільш широкої аудиторії користувачів, включно із доступністю для користувачів із обмеженими можливостями, є головною метою ненав'язливого підходу. Досягнення цієї мети і базується на розділенні представлення і поведінки, за якого поведінка програмується з допомогою зовнішніх скриптів мовою JavaScript і прив'язується до семантичної розмітки.

Завдяки використанню ненав'язливого підходу легше досягти таких результатів:

- Доступність сайту для більшої кількості користувачів;
- Гнучкість при внесенні змін у документ, стилі чи скрипти;
- Експлуатаційна надійність (robustness) і можливість розширення, в тому числі можливість поступового покращення;
- Підвищення продуктивності, наприклад, за рахунок кешування зовнішніх скриптів.

Кріс Хейлман (Cris Heilmann), один із прихильників використання ненав'язливого підходу, склав для нього у 2007 році сім простих правил[4]:

- Не робіть припущень;
- Шукайте зачіпки (hooks) і зв'язки;
- Залиште обхід (traversing) експертам;
- Розумійте роботу браузерів і користувачів;
- Зрозумійте, як працюють події;
- Грайте з іншими розробниками;
- Турбуйтесь про наступного розробника.

Прототипне програмування — стиль об'єктно-орієнтованого програмування, при якому відсутнє поняття класу, а повторне використання (успадкування) проводиться шляхом клонування існуючого примірника об'єкта — прототипу.

У мовах, що базуються на понятті «клас», всі об'єкти розділені на два основних типи — класи та екземпляри. Клас визначає структуру і функціональність (поведінку), однакову для всіх екземплярів даного класу. Екземпляр є носієм даних — тобто володіє станом, що міняється відповідно до поведінки, заданої класом.

Прихильники прототипного програмування часто стверджують, що мови, засновані на класах, призводять до надмірної концентрації на таксономії класів і на відносинах між ними. На противагу цьому, прототипування загострює увагу на поведінці певної (невеликої) кількості «зразків», які потім класифікуються як «базові» об'єкти і використовуються для створення інших об'єктів. Багато прототип-орієнтованих систем підтримують зміну прототипів під час виконання програми, тоді як лише невелика частина клас-орієнтованих систем (наприклад, Smalltalk) дозволяють динамічно змінювати класи.

Хоча переважна більшість прототип-орієнтованих систем це інтерпретовані мови з динамічною типізацією, технічно можливо додати прототипування і в мови зі статичною перевіркою типів. Мова Omega є одним із прикладів такої системи.

В клас-орієнтованих мовах новий екземпляр створюється через виклик конструктора класу (можливо, з набором параметрів). Отриманий екземпляр має структуру і поведінку, жорстко задані його класом.

У прототип-орієнтованих системах надається два методи створення нового об'єкта: клонування існуючого об'єкта, або створення об'єкта «з нуля». Для створення об'єкта з нуля програмісту надаються синтаксичні засоби додавання властивостей і методів в об'єкт. Надалі, з отриманого об'єкта може бути отримана повна копія, клон. У процесі клонування копія успадковує всі характеристики свого прототипу, але з цього моменту вона стає самостійною і може бути змінена. У деяких реалізаціях копії зберігають посилання на об'єкти-прототипи, делегуючи їм частину своєї функціональності, при цьому зміна прототипу може торкнутися всіх його копій. В інших реалізаціях нові об'єкти повністю незалежні від своїх прототипів. Розглянемо кожну з цих реалізацій.

Прихильників клас-орієнтованих об'єктних моделей, що критикують прототипний підхід, часто турбують ті самі проблеми, якими стурбовані прихильники статичної типізації у відношенні до мов з динамічною типізацією. Зокрема, обговорення обертаються навколо таких тем, як правильність, безпека, передбачуваність та ефективність програми.

Що стосується перших трьох пунктів, то класи часто розглядаються як типи (і справді, в більшості об'єктно-орієнтованих мов зі статичною типізацією так воно і є), і передбачається, що класи надають певні домовленості і гарантують, що екземпляри будуть вести себе цілком певним чином.

У частині ефективності, оголошення класів значно спрощує компілятору завдання оптимізації, роблячи ефективнішими як методи, так і пошук атрибутів у примірниках. У випадку мови Self чимала частина часу

була витрачена на розробку таких технік компіляції та інтерпретації, які дозволили б наблизити продуктивність прототип-орієнтованих систем до їхніх клас-орієнтованих конкурентів.

Нарешті, можливо найзагальнішим місцем критики проти прототипного програмування є те, що співтовариство розробників ПЗ недостатньо добре знайоме з ним, незважаючи на популярність і поширеність JavaScript. До того ж, так як прототип-орієнтовані системи є порівняно новими і все ще нечисленними і рідкісними, прийоми розробки з їхнім використанням досі не отримали великого поширення.

ECMAScript — стандарт мови програмування, затверджений міжнародною організацією ECMA згідно зі специфікацією ECMA-262. Найвідомішими реалізаціями стандарту є мови JavaScript, JScript та ActionScript, які широко використовується у Вебі.

JavaScript має низку властивостей об'єктно-орієнтованої мови, але завдяки концепції прототипів підтримка об'єктів в ньому відрізняється від традиційних мов ООП. Крім того, JavaScript має ряд властивостей, притаманних функціональним мовам, — функції як об'єкти першого рівня, об'єкти як списки, каррінг (currying), анонімні функції, замикання (closures) — що додає мові додаткову гнучкість.

JavaScript має C-подібний синтаксис, але в порівнянні з мовою Сі має такі корінні відмінності:

- об'єкти, з можливістю інтроспекції і динамічної зміни типу через механізм прототипів
- функції як об'єкти першого класу
- обробка винятків

- автоматичне приведення типів
- автоматичне прибирання сміття
- анонімні функції

Семантика мови схожа з семантикою мови Self.

Приклад оголошення і використання класу в ECMAScript (клас є одночасно функцією, оскільки функції — це об'єкти першого рівня):

```
function MyClass()
{
  this.myValue1 = 1;
  this.myValue2 = 2;
}
var mc = new MyClass();
mc.myValue1 = mc.myValue2 * 2;
```

Одна з популярних технологій, що дозволила зробити сторінки динамічнішими і забезпечити нові можливості — це динамічне завантаження і вставка даних в документ, що отримала назву AJAX.

При використанні в рамках технології DHTML ECMAScript код включається в HTML-код сторінки і виконується інтерпретатором, вбудованим в браузер. Код JavaScript вставляється в теги <script></script> з обов'язковим по специфікації HTML 4.01 атрибутом type="text/javascript", хоча в більшості браузерів мова сценаріїв за умовчанням саме JavaScript.

Скрипт, що виводить модальне вікно з класичним написом «Hello, World!» усередині браузера:

```
<script type="text/javascript">
  alert('Hello, World!');
```

```
</script>
```

Слідуючи концепції інтеграції JavaScript в існуючі системи, браузери підтримують включення скрипта, наприклад, в значення атрибуту події:

```
<a href="delete.php" onclick="return confirm('Ви впевнені?');">Видалити</a>
```

Тут при натисненні на посилання функція confirm('Ви впевнені?'); викликає модальне вікно з написом «Ви впевнені?», а return false; блокує перехід за посиланням. Зрозуміло, цей код працюватиме тільки якщо в браузері є та увімкнена підтримка JavaScript, інакше перехід за посиланням відбудеться без попередження.

Є і третя можливість підключення JavaScript — написати скрипт в окремому файлі, а по тому підключити його за допомогою конструкції:

```
<script type="text/javascript" src="http://Шлях_до_файла _зі_ скриптом "></script>
```

При розробці великих і нетривіальних веб-застосунків з використанням JavaScript, критично важливим є доступ до інструментів зневадження. Оскільки браузери від різних виробників дещо відрізняються у поведінці (в тому числі і в Об'єктній Моделі Документа, треба мати в руках зневаджувачі для кожного браузера, якщо веб-застосунок орієнтовано на нього.

Firefox, Google Chrome, Opera та Safari мають вбудовані зневаджувачі під себе.

Internet Explorer має три зневаджувачі для себе: Microsoft Visual Studio є найпотужнішим з цих трьох, слідом йде Microsoft Script Editor (компонента Microsoft Office), також існує безкоштовний Microsoft Script Debugger з базовими функціями. Веб-застосунки для Firefox допоможе вдосконалити

додаток Firebug (зручно вбудований безпосередньо в браузер), або давніший зневаджувач Venkman, котрий також працює з браузером Mozilla. Drosera — це зневаджувач з WebKit engine, що супроводжує Apple Safari.

Також існують кілька інструментів, як вільних, наприклад JSLint[4], інструмент перевірки якості коду, що сканує JavaScript програму, шукаючи проблеми коду, так і комерційних продуктів типу інструменту з назвою JavaScript Debugger.

Оскільки ECMAScript є інтерпретатором, без суворої типизації, і може виконуватися в різних середовищах, кожне зі своїми власними особливостями сумісності, програміст має бути дуже уважним, і повинен перевіряти, що його код виконується як очікується в широкому переліку можливих конфігурацій. Дуже часто трапляються випадки, коли скрипт, що чудово працює в одному середовищі, видає некоректні результати в іншому.

Кожен блок сценарію інтерпретатор розбирає окремо. На веб-сторінках, коли треба комбінувати блоки JavaScript та HTML, синтаксичні помилки знайти простіше, якщо тримати функції сценарію в окремому блоці коду, або (ще краще) використовувати багато малих пов'язаних .js файлів. В такий спосіб синтаксична помилка не спричинятиме «падіння» цілої сторінки, і можна надати допомогу, елегантно вийшовши зі сторінки.

Для серверних проектів node.js можна використовувати інтегроване середовище розробки WebStorm.

Суворий режим (strict mode) у ECMAScript 5 — це спосіб використання обмеженого варіанта JavaScript. Суворий режим є не просто підмножиною

JavaScript: це семантика, яка навмисно відрізняється від звичайного коду. Переглядачі, що не підтримують суворий режим, виконуватимуть його код у дещо інший спосіб, аніж ті переглядачі, що таку підтримку мають. Отже перш ніж покладатися на суворий режим, перевірте наявність підтримки його окремих складників. Код у суворому режимі та код поза ним можуть співіснувати, тому скрипти можна переводити у суворий режим поступово.

Суворий режим вносить декілька змін до звичної семантики JavaScript. По-перше, деякі помилки, що в звичайному режимі лишаються непомітними, натомість у суворому режимі викидаються в явний спосіб. По-друге, суворий режим виправляє помилки, що ускладнюють або унеможливлюють виконання певних оптимізацій рушіями JavaScript: код у суворому режимі іноді можна змусити виконуватися швидше, ніж такий самий код у звичайному режимі. По-третє, суворий режим забороняє деякий синтаксис, який, скоріше за все, буде визначено у майбутніх версіях ECMAScript.

Суворий режим застосовується до цілих скриптів, або до індивідуальних функцій. Він не застосовується до блочних виразів, замкнених у фігурних дужках {}; спроба застосувати його в такому контексті не дасть результату. eval код, Function код, атрибути обробника подій, рядки, що передаються у WindowTimers.setTimeout(), і подібні до них є повноцінними скриптами, і виклик суворого режиму в них працює, як і очікується.

Щоб увімкнути суворий режим у цілому скрипті, запишіть точний вираз “use strict”; (або ‘use strict’;) перед будь-якими іншими виразами.

Цей синтаксис приховує пастку, в яку вже втрапив один великий сайт: неможливо наосліп поєднати не суперечливі скрипти. Розглянемо об'єднання

скрипту у суворому режимі із скриптом у звичайному режимі: вся конкатенація виходить у суворому режимі! Зворотнє теж правда: скрипт у звичайному режимі, поєднаний із скриптом у суворому режимі, призводить до звичайного режиму. Об'єднання скриптів у суворому та звичайному режимах є проблематичним. Тому рекомендується включати суворий режим у кожній функції окремо (принаймні, на час переміщення).

Ви також можете застосувати підхід із загортанням всього вмісту скрипту у функцію та наданням цій зовнішній функції суворого режиму. Це позбавляє проблеми конкатенації, але це означає, що вам доведеться явно експортувати будь-які глобальні змінні поза область видимості функції.

Суворий режим змінює як синтаксис, так і поведінку під час виконання. Зміни загалом підпадають під такі категорії: зміни, що перетворюють похибки на помилки (такі, як помилки синтаксису під час виконання), зміни, що спрощують обчислення конкретної змінної для заданого використання імені, зміни, що спрощують eval та arguments, зміни, що спрощують написання “безпечного” коду JavaScript, а також зміни, що передбачають майбутню еволюцію ECMAScript.

Суворий режим перетворює деякі раніше прийнятні похибки на помилки. JavaScript був розроблений таким чином, щоб бути простим для програмістів-початківців, й іноді позначає операції, які мали б бути помилками, як не помилкові. Іноді це вирішує невідкладну проблему, але іноді створює ще гірші проблеми в майбутньому. Суворий режим позначає ці похибки як помилки, що дозволяє виявляти їх та вчасно виправляти.

По-перше, суворий режим робить неможливим випадкове створення глобальних змінних. В звичайному режимі JavaScript неправильно написане

ім'я змінної під час присвоювання створює нову властивість у глобальному об’єкті та продовжує “працювати” (хоча в майбутньому можливе не спрацювання: ймовірно, у новій версії JavaScript). Присвоювання, які могли випадково створити глобальні змінні, у суворому режимі генерують помилку.

По-друге, суворий режим змушує присвоювання, які б в іншому випадку непомітно не спрацювали, генерувати помилку. Наприклад, NaN є глобальною змінною, яку не можна редагувати. В нормальному коді присвоювання у NaN нічого не робить; розробник отримує неуспішний результат. У суворому режимі присвоювання у NaN викликає виняток. Будь-яке присвоювання, яке непомітно не спрацює у нормальному коді (присвоювання властивості, яку не можна редагувати, присвоювання властивості, яка має лише геттер, присвоювання нової властивості нерозширюваного об'єкту) згенерує виняток у суворому режимі.

По-третє, суворий режим генерує винятки при намаганні видалити властивість, яку не можна видаляти (раніше така спроба просто не мала б ніякого ефекту).

По-четверте, суворий режим до версії Gecko 34 вимагає, щоб усі властивості у літералі об’єкту були унікальними. В звичайному коді можливо дублювати імена властивостей, при цьому останній запис встановлює значення властивості. Але, оскільки лише останній запис має значення, дублювання — це просто джерело помилок, якщо в коді змінюється значення властивості не в останньому запису. Дублювання імен властивостей є синтаксичною помилкою у суворому режимі.

По-п’яте, суворий режим вимагає, щоб імена параметрів функцій були унікальними. В звичайному коді останній дубльований аргумент ховає

попередні ідентично названі аргументи. Ці попередні аргументи лишаються доступними через arguments[i], тому вони не остаточно недоступні. Однак, таке приховання не має сенсу і, скоріше за все, є небажаним (воно може, наприклад, ховати помилку у написанні), тому у суворому режимі дублювання імен аргументів є синтаксичною помилкою.

По-шосте, суворий режим у ECMAScript 5 забороняє вісімковий синтаксис. Вісімковий синтаксис не є частиною ECMAScript 5, але він підтримується у всіх браузерах додаванням нуля попереду вісімкового числа: 0644 === 420 and "\045" === "%". У ECMAScript 5 вісімкове число підтримується з допомогою додавання "0o"

По-сьоме, суворий режим у ECMAScript 5 забороняє присвоєння властивостей примітивним (primitive) значенням. Без суворого режиму, присвоєння таких властивостей просто ігнорується (no-op), однак, у суворому режимі генерується помилка TypeError.

Суворий режим спрощує прив'язку імен змінних до певних визначень змінних у коді. Багато оптимізацій компілятора покладаються на спроможність сказати, що змінна X зберігається у цій локації: це критично для повної оптимізації кода JavaScript. Іноді JavaScript робить базову прив'язку імені до визначення змінної в коді неможливим до запуску коду. Суворий режим усуває більшість випадків, в яких це відбувається, тому компілятор може краще оптимізувати код у суворому режимі.

Суворий режим робить arguments та eval менш химерно магічними. Обидва поводяться досить загадково в нормальному коді: eval для додавання або видалення зв’язків або для зміни значень зв’язків, та arguments з їхніми індексованими властивостями в якості псевдонімів імен аргументів. Суворий

режим робить величезний крок до поводження з eval та arguments як з ключовими словами, хоча остаточні зміни з'являться не раніше майбутнього видання ECMAScript.

Суворий режим спрощує написання "безпечного" коду в JavaScript. Деякі веб-сайти зараз надають користувачам можливість писати код JavaScript, який буде виконуватися на цій сторінці від імені інших користувачів. JavaScript у браузерах може мати доступ до приватної інформації користувачів, тому такий код має бути частково перетворений перед виконанням, щоб контролювати доступ до забороненої функціональності. Гнучкість JavaScript робить це практично неможливим без купи перевірок під час виконання. Певні функції мови настільки розповсюджені, що перевірки під час виконання призводять до суттєвих втрат у продуктивності. Кілька вивертів суворого режиму, плюс вимога, щоб код JavaScript, наданий користувачем, був у суворому режимі і щоб він запускався певним чином, суттєво зменшують необхідність в таких перевірках.

У більшості браузерів в наш час реалізовано суворий режим. Однак, не покладайтеся на нього наосліп, оскільки досі існує велика кількість версій браузерів, які лише частково підтримують суворий режим або взагалі його не підтримують (наприклад, Internet Explorer нижче 10-ї версії!). Суворий режим змінює семантику. Покладання на суворий режим призведе до численних помилок у браузерах, в яких не реалізовано суворий режим. Використовуйте суворий режим з обережністю, та, опираючись на суворий режим, підстраховуйтесь тестуванням функціональності, яке перевірить, чи реалізовані відповідні частини суворого режиму. Нарешті, переконайтеся, що

протестували свій код у браузерах, які підтримують та не підтримують суворий режим. Якщо ви тестуєте лише у тих браузерах, які не підтримують суворий режим, ви, ймовірно, будете мати проблеми у браузерах, які його підтримують, та навпаки.

Завдання на лабораторну роботу

Реалізуйте завдання з попередньої лабораторної роботи, використовуючи функціональний або прототипний підхід. Активуйте суворий режим в коді з попередньої лабораторної роботи. Приведіть ваш код у відповідність до правил суворого режиму та стандартів ECMAScript.

Контрольні запитання

1. Опишіть принципи функціонального підходу до програмування в JS.
2. Що таке конструктор? Як працюють конструктори в Javascript.
3. Опишіть модифікатори доступу до об’єкта в Javascript.
4. Опишіть принципи прототипного підходу до програмування в JS.
5. Що таке прототип? Опишіть шляхи взаємодії з прототипами в JS.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Документація по Javascript. [Електронний ресурс] Режим доступу:https://developer.mozilla.org/uk/docs/Web/JavaScript/Reference
2. Современный учебник по Javascript [Електронний ресурс] Режим доступу: learn.javascript.ru
3. JavaScript : Створення функцій. Локальні і глобальні змінні, яка їх відмінність [Електронний ресурс]. Режим доступу: http://programmersworld.xyz/article/9/75
4. DOM - Яваскрипт.укр [Електронний ресурс]. Режим доступу: http://xn--80adth0aefm3i.xn--j1amh/DOM
5. Об'єкти JavaScript [Електронний ресурс]. Режим доступу: http://www.kievoit.ippo.kubg.edu.ua/kievoit/java/part2/part2.htm
6. Ненав’язливий Javascript [Електронний ресурс]. Режим доступу: https://uk.wikipedia.org/wiki/%D0%9D%D0%B5%D0%BD%D0%B0%D0%B2%27%D1%8F%D0%B7%D0%BB%D0%B8%D0%B2%D0%B8%D0%B9_JavaScript
7. jQuery AJAX методи: $ .ajax (), $ .get (), $ .post () [Електронний ресурс]. Режим доступу: https://devionity.com/uk/courses/ dive-into-ajax/jquery-ajax-methods
8. Суворий режим Javascript. [Електронний ресурс]. Режим доступу: https ://developer.mozilla.org/uk/docs/Web/JavaScript/Reference/Strict_mode