ЛАБОРАТОРНА РОБОТА № 3

**Тема:** Введення до мови програмування JavaScript.

**Мета:** Набуття теоретичних та практичних навичок по роботі з мовою програмування JavaScript.

Теоретичні відомості

Наведений нижче матеріал стосується лише загальних принципів JavaScript, тому його можна пропустити. Але обов’язковим для вивчення є матеріали сайту https://learn.javascript.ru/, де подається вичерпна інформація про дану технологію.

Окрім цього, бажаним є ознайомлення з історією виникнення і розвитку JavaScript. Це можна зробити за посиланням https://habrahabr.ru/company/livetyping/blog/324196/.

Найпопулярнішою технологією написання сценаріїв для браузерів є мова JavaScript. Мова JavaScript – це мова сценаріїв, що виникла на основі мови програмування Java. Вважалось, що вона буде простіша за Java у вивченні та користуванні, оскільки з самого початку була призначена для покращення інтерфейсу користувача і не претендувала на звання повноцінної мови програмування.

Мова JavaScript схожа на Java, але не забезпечує строгої перевірки типів. Середовище виконання JavaScript менше, ніж у Java, і містить меншу кількість типів даних. Мова JavaScript основана на об’єктах, але не є об’єктно-орієнтованою, тобто вона використовує вбудовані поширювані об’єкти, але не підтримує об’єкти, визначені користувачем, та механізм успадкування. Оскільки мова JavaScript не є компільованою, вона підтримує динамічне зв’язування і всі посилання на об’єкти перевіряються у процесі виконання сценарію. JavaScript підтримує функції без жодних спеціальних вимог до їх оголошень, що полегшує використання мови новачками.

Для визначення області сценарію JavaScript в коді HTML використовується дескриптор <script>; він позначає початок коду JavaScript, а </script> – його кінець. Весь текст, що знаходиться між цими дескрипторами, аналізується інтерпретатором JavaScript. Розглянемо наступний приклад:

<script>

alert(‘hello world!!!’)

</script>

В результаті буде висвітлено віконце з написом «Hello World!!!».

Якщо використовується окремий сценарій JavaScript, то відповідний файл має бути з розширенням js.

Події

Сценарії в документі HTML призначені, зокрема, для обробки подій: натискання мишкою на елементі документа, наведення стрілки миші на елемент чи забирання її з нього, натиснення клавіші тощо. Більшість дескрипторів HTML мають спеціальні атрибути, що визначають події, на які можуть реагувати відповідні елементи. Список допустимих подій наведений нижче; він досить великий і розрахований на всі випадки життя. Значенням такого атрибуту-події є рядок, що містить сценарій – код-обробник події. Зазвичай обробники подій оформлюються у вигляді функцій, визначення яких розміщують у дескрипторі <script>. Розглянемо наступні приклади:

Приклад 1

<html>

<script>

function clickimage(){

alert (“Hello!”)

}

</script>

<img src=“pict.gif” onclick=“clickimage()”>

</html>

Приклад 2

<html>

<img src=“pict.gif” onclick=“alert(“Hello!”)”>

</html>

В таблиці 3.1 наведено основні події для документів HTML.

Таблиця 3.1. Основні події для документів HTML

|  |  |  |  |
| --- | --- | --- | --- |
| Подія | До яких елементів застосовна | Коли відбувається | Обробник події |
| abort | Зображення | Користувач перериває завантаження, наприклад, натисканням на нове посилання чи кнопку зупинки | onabort |
| blur | Вікна, фрейми та всі елементи форм | Користувач переносить фокус введення з вікна, фрейму чи елемента форми | onblur |
| click | Всі кнопки, перемикачі, прапорці та посилання | Натискання мишею на елементі | onclick |
| change | Текстові поля та області, списки | Користувач змінює значення елемента | onchange |
| error | Зображення, вікна | Завантаження документа чи зображення призвело до помилки | onerror |
| focus | Вікна, фрейми та всі елементи форм | Користувач переносить фокус введення у вікно, фрейм чи елемент форми. | onfocus |
| load | Тіло документа | Користувач завантажує сторінку у браузер | onload |
| mouseout | Області, посилання | Користувач пересуває стрілку миші за межі області чи посилання | onmouseout |
| mouseover | Посилання | Користувач поміщає стрілку миші на посилання | onmouseover |
| reset | Форми | Користувач повертає форму у вихідний стан (натисканням кнопки Reset) | onreset |
| select | Текстові поля та області | Користувач виділяє поле вводу елемента форми | onselect |
| submit | Кнопки Submit | Користувач передає форму на сервер | onsubmit |
| unload | Тіло документа | Користувач закриває сторінку | onunload |

Функції

Функції користувача – це функції, які ви можете створити самі, за власним вибором, для вирішення своїх задач. Функція задається своїм описанням, яке починається словом function і має наступний синтаксис:

function ім'я\_функції(параметри) {

код

}

Якщо функція повинна повернути деяке значення, то в її тілі використовується оператор return, справа від якого вказується те, що необхідно повернути. Цей оператор може міститись в тілі функції декілька разів.

Ініціалізація в тілі функції змінної з ключовим словом var (let, const) створює локальну змінну, навіть якщо вона вже була оголошена у зовнішній програмі; якщо вона створена лише з допомогою оператора присвоювання, то вона є глобальною.

Приклад 4

function result(a,b) {

c=a+b;

return c;

}

Об'єктна модель документа

Незалежно від базової стратегії розробки клієнта, всі підходи базуються на об’єктній моделі документа DOM (Document Object Model) – платформно-незалежному інтерфейсі браузера з відображуваним документом HTML. Специфікація такої моделі визначена консорціумом W3C (World Wide Web Consortium – W3C – це об’єднання розробників технологій та організацій, відповідальних за стандарти HTTP, HTML, XML, XSL, DOM та інші важливі стандарти Web та Internet), і більшість розробників браузерів реалізували її у останніх версіях своїх продуктів. Основна ідея полягає у використанні спільного інтерфейсу API, який розробники Web-сторінок можуть застосовувати для обробки вмісту документа HTML (або XML), а також ресурсів самого браузера.

При використанні об’єктної моделі документа програми та сценарії можуть динамічно отримувати доступ та оновлювати зміст документа, його структуру та стиль. Потім документ може оброблятись браузером, і результати цієї обробки можуть фіксуватись на відображуваній сторінці. За такої архітектури браузер відповідає як за відображення сторінки HTML, яка може бути змінена після її отримання з сервера, так і за виконання сценаріїв та компільованих програм в документі.

Назва моделі DOM пов’язана з тим, що вона забезпечує об’єктний інтерфейс до документів HTML (та XML). Документи розглядаються як об’єкти, що мають дані та поведінку. Взаємодія цих об’єктів представляє структуру документа.

Об'єкти JavaScript

Об'єктна модель документа, головне джерело об’єктів для JavaScript, забезпечує об’єктний інтерфейс не лише для документів HTML, але й для браузера. Сценарій JavaScript може взаємодіяти з браузером для завантаження нової сторінки, перевіряти журнал браузера (список завантажених раніше Web-сторінок) або взаємодіяти з іншими сторінками у сусідніх фреймах.

Головним об'єктом при роботі з документами є document. Посилання на цей об'єкт можна отримати за допомогою атрибута цього об’єкта window, який є глобальним для будь-якої функції JavaScript. Для звернення до атрибутів в JavaScript використовується оператор „крапка” (.).

В об’єктній моделі документи згруповані у так звані колекції. Колекцію можна розглядати як проміжний об'єкт, що містить об’єкти власне документа. З іншого боку, колекція є масивом об’єктів, відсортованих у порядку слідування відповідних елементів в HTML-документі. Синтаксис звертання до елементів колекції такий же, як і для елементів масиву. Колекція має атрибут length – кількість всіх її документів. Колекція всіх елементів документа називається all; є також тематичні колекції: images, forms, links тощо (колекції усіх зображень, форм, посилань відповідно). Об'єкт може належати до якоїсь тематичної колекції, але обов’язково входить до колекції all. Загальні правила звертання до атрибутів всіх об’єктів такі:

document.колекція.id\_об’єкта

document.колекція[“id\_об’єкта”]

document.колекція[індекс\_об’єкта]

Об'єкти користувача JavaScript

Хоча JavaScript не є повноцінною об’єктно-орієнтованою мовою, вона забезпечує механізм для інкапсуляції. Інкапсуляція (encapsulation) – це можливість приховання даних всередині об’єкта. У мові JavaScript можна створити екземпляр родового об’єкта (generic object) і призначити йому атрибути і навіть методи. Наприклад, у наступному фрагменті коду на JavaScript створюється екземпляр родового об’єкту з іменем cartLineItem (один з товарів у віртуальному кошику покупця). За допомогою оператора „крапка” (.) задаються значення чотирьох вказаних користувачем властивостей.

Приклад 5

LineItem= new Object();

LineItem.productID = ‘MG1234’;

LineItem.productName = ‘MGB Roadster (1935)’;

LineItem.qty =1;

LineItem.unitPrice =36000;

Екземпляр LineItem можна використовувати в подальшому в будь-якому сценарії JavaScript з відповідним посиланням:

alert( ‘В вашому кошику є ’ + cartLineItem.qty + ‘ ’ + LineItem.productName);

Для об’єктів можна також визначати методи. Наприклад, наступну функцію total() можна використовувати як метод. Вона звертається до свого об’єкту за допомогою оператора this:

function total(){

return (this.qty \* this.unitPrice);

}

Доступ до функції здійснюється таким же чином як і до атрибутів.

LineItem.total = total;

Тепер цю функцію можна викликати безпосередньо для об’єкта LineItem:

LineItem= new Object();

LineItem.productID = ‘MG1234’;

LineItem.productName = ‘MGB Mk I Roadster’;

LineItem.qty = 2;

LineItem.unitPrice = 12500;

LineItem.total = total;

document.write(‘<p>’ + LineItem.qty + ‘ ’ + LineItem.productName +‘стоит $’ + LineItem.total() + ‘</p>’);

Прототип в JavaScript нагадує конструктор в С++. Прототип (prototype) – це функція, що створює визначений користувачем об'єкт та ініціалізує його атрибути. Прототип для об’єкта JavaScript повинен також містити функції об’єкта. Прототип об’єкта cartLineItem повинен мати такий вигляд:.

function cartLineItem(id, name, qty, price){

this.productID = id;

this.productName = name;

this.qty = qty;

this.unitPrice = price;

this.total = total;

}

Якщо визначений прототип, то масив екземплярів LineItem можна створити за допомогою наступного фрагменту коду JavaScript:

var LineItem = new Array();

LineItem[0] = new cartLineItem (‘MG123’, ‘MGB Mk I Roadster’, 1, 36000);

LineItem[1] = new cartLineItem (‘AH736’, ‘Austin-Healey Sprite’, 1, 9560);

LineItem[2] = new cartLineItem (‘TS225’, ‘Triumph Spitfire Mk I’, 1, 11000);

Завдання на лабораторну роботу

1. Створити форму, яка міститиме наступні поля:

* ім’я;
* прізвище;
* адреса, місто, країна та поштовий індекс;
* адреса електронної пошти (e-mail).

Виконати перевірку інформації, введеної в поля форми (e-mail на наявність символу @, ім’я та прізвище повинно містити лише літери, поштовий індекс повинен складатися з 5 цифр).

Результат перевірки вивести на екран. Для стилізації помилок та форми вводу використати фреймворк Bootstrap (http://getbootstrap.com/).

1. Написати функцію, яка розділить рядок на пари з двох символів, результат запише у масив і виведе його на екран у вигляді списку (ul/li). Якщо рядок містить непарну кількість символів, то він повинен замінити відсутній другий символ останньої пари з підкресленням «\_».

Приклад:

'abc' => ['ab', 'c\_']

'abcdef' => ['ab', 'cd', 'ef']

1. Створити графічний елемент (наприклад прямокутник), який буде довільно переміщатися по екрану не виходячи за його межі. Переміщення зупинити, коли навести на нього стрілку мишки.
2. **Завдання на додаткову кількість балів.** Реалізувати примітивний слайдер малюнків без використання сторонніх бібліотек, що вже його реалізують. Допускається використання jQuery, але виключно з метою маніпуляцій над елементами та створенням анімації.

Слайдер має мати наступний вигляд:



Зміст звіту

1. Назва й мета лабораторної роботи.
2. Скріншоти результатів роботи.
3. Висновки по лабораторній роботі.

Корисні посилання

1. Навчальні матеріали по JavaScript:

* <https://learn.javascript.ru/>
* <https://developer.mozilla.org/uk/docs/Learn/Getting_started_with_the_web/JavaScript_basics>

1. Історія розвитку технології – <https://habrahabr.ru/company/livetyping/blog/324196/>.