МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КРЕМЕНЧУЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ МИХАЙЛА ОСТРОГРАДСЬКОГО

МЕТОДИЧНІ ВКАЗІВКИ
ЩОДО ВИКОНАННЯ ЛАБОРАТОРНИХ РОБІТ
З НАВЧАЛЬНОЇ ДИСЦИПЛІНИ

«ПРИКЛАДНЕ ПРОГРАМУВАННЯ В К ОМП'ЮТЕРНИХ МЕРЕЖАХ»

ДЛЯ СТУДЕНТІВ ДЕННОЇ ФОРМИ НАВЧАННЯ
ЗІ СПЕЦІАЛЬНОСТІ 123 – «КОМП'ЮТЕРНА ІНЖЕНЕРІЯ».
ЧАСТИНА ІІ

КРЕМЕНЧУК 2018

Методичні вказівки щодо виконання лабораторних робіт з навчальної дисципліни «Прикладне програмування в комп’ютерних мережах» для студентів денної форми навчання зі спеціальності 123 – «Комп’ютерні системи та мережі». Частина ІІ

Укладач к. т. н., доц. О. Г. Славко

Рецензент к. т. н., доц. В. М. Сидоренко

Кафедра комп’ютерних та інформаційних систем

Затверджено методичною радою КрНУ імені Михайла Остроградського
Протокол № 11 від 9 липня 2018 р.

Голова методичної ради _____ проф. В. В. Костін

ЗМІСТ

Вступ..... 4
1 Перелік лабораторних робіт..... 6
Лабораторна робота № 1 Створення простого та зовнішнього скрипту мовою JavaScript..... 6
Лабораторна робота № 2 Побудова скрипту з використанням основних операторів, циклів і функцій мови JavaScript..... 9
Лабораторна робота № 3 Створення скрипту з використанням об'єктів, властивостей і методів у JavaScript..... 13
Лабораторна робота № 4 Основи роботи з масивами та їх методами у JavaScript..... 16
Лабораторна робота № 5 Використання функцій-конструкторів об'єктів і дескрипторів їх властивостей у JavaScript..... 20
Лабораторна робота № 6 Основи об'єктно-орієнтованого програмування в JavaScript..... 24
Лабораторна робота № 7 Основи роботи з браузерними подіями в JavaScript: робота з мишкою, клавіатурою, завантаження скриптів і фреймів.... 26
2 Критерії оцінювання якості виконання лабораторних робіт студентами..... 31
Список літератури..... 32

ВСТУП

Навчальна дисципліна є логічним продовженням навчального курсу «Комп'ютерні системи», «Програмування», «Комп'ютерні мережі», «Мережні інформаційні технології», «Організація баз даних». Предметом вивчення навчальної дисципліни є теорія і практика розробки прикладних мережних додатків на базі сучасних технологій розробки програмного забезпечення в комп'ютерних мережах.

У межах начальної дисципліни розглядаються питання прикладного програмування в комп'ютерних мережах, зокрема web-додатків. Студенти набувають практичних навичок і здатності застосовувати мережних технологій, методи проектування та інструменти для розроблення мережних програмних продуктів на сучасних мережних платформах, а саме: теоретичні та практичні аспекти технології розробки сучасних веб-додатків; основні типи веб-додатків, методології їх проектування та розробки; інформаційні системи з використанням веб-додатків, інструментальні засоби, що підтримують розробку програмного забезпечення професійно орієнтованих веб-додатків, технічні засоби інформаційних систем на основі веб-додатків у предметній галузі.

Мета і завдання навчальної дисципліни: набуття студентами загальних теоретичних і практичних знань у галузі створення прикладних мережних додатків, які базуються на сучасних мережних технологіях і платформах, а також організації функціонування та супровід мережних додатків, побудованих за клієнт-серверною технологією.

Місце навчальної дисципліни у навчальному процесі: навчальна дисципліна «Прикладне програмування в комп'ютерних мережах» тісно пов'язана з такими навчальними дисциплінами, як «Комп'ютерні системи», «Програмування», «Комп'ютерні мережі», «Мережні інформаційні технології», «Організація баз даних» та іншими спеціальними дисциплінами навчального плану.

У результаті вивчення навчальної дисципліни студент повинен:

знати:

– основні сучасні технології та методології розробки мережних додатків;

– методи організації процесу розрабоки web-додатків;

– стандарти, покладені в основу web-технологій;

– принципи побудови мережних додатків з урахуванням сучасних вимог інформаційного простору;

– принципи взаємодії web-додатка з іншими компонентами інформаційної системи;

– інструментальні засоби створення серверної та клієнтської частин мережних додатків;

уміти:

– організовувати процес проектування та розробки web-додатка;

– розробляти серверну частину мережних додатків для різних мережних платформ з використанням різних технологій;

– розробляти клієнтську частину мережних додатків для різних мережних платформ з використанням різних технологій;

– обґрунтовано вибирати програмну платформу й інструментарій для розробки web-додатків;

– застосовувати стандарти web-технологій;

– здійснювати супровід мережних додатків.

1 ПЕРЕЛІК ЛАБОРАТОРНИХ РОБІТ

Лабораторна робота № 1

Тема. Створення простого та зовнішнього скрипту мовою JavaScript

Мета: ознайомлення та набуття практичних навичок створення простого та зовнішнього скрипту мовою JavaScript.

Короткі теоретичні відомості

Скрипт – це невелика програма, написана зазвичай для потреб інтернет-сайту або програми.

Скрипти часто поділяють на «серверні», виконувані на сервері перед відправкою інтернет-сторінки користувачеві, і «клієнтські», вбудовані у веб-сторінку і виконувані вашим браузером. Перші пишуться мовами PHP, Perl, Visual Basic, Java, Ruby або Pyton. Другі – майже виключно мовою JavaScript.

Програми мовом JavaScript можна вставити у будь-яке місце HTML за допомогою тегу SCRIPT. Наприклад:

```
<!DOCTYPE HTML>
<html>
<head>
  <!-- Тег meta для указания кодировки -->
  <meta charset="utf-8">
</head>
<body>
  <p>Начало документа...</p>
  <script>
    alert( ‘Привет, Мир!’ );
  </script>
  <p>...Конец документа</p>
</body>
</html>
```

Якщо браузер бачить <script>, він чинить так:

– починає відображати сторінку, показує частину документа до script;

– зустрівши тег script, перемикається в JavaScript-режим і не показує, а виконує його вміст;

– закінчивши виконання, повертається назад у HTML-режим і тільки тоді відображає частину документа.

Якщо JavaScript-коду багато – його виносять в окремий файл, який підключається в HTML:

<script src="/path/to/script.js"></script>

Тут /path/to/script.js – це абсолютний шлях до файлу, що містить скрипт (з кореня сайту). Браузер сам завантажить скрипт і виконає.

Можна вказати і повний URL, наприклад:

<script src="https://cdnjs.cloudflare.com/ajax/libs/lodash.js/4.3.0/lodash.js"></script>

Можна використовувати шлях щодо поточної сторінки. Наприклад, src = "lodash.js" позначає файл з поточної директорії.

Щоб підключити декілька скриптів, слід використовувати декілька ключових тегів:

<script src="/js/script1.js"></script>

<script src="/js/script2.js"></script>

...

Зазвичай у HTML пишуть тільки найпростіші скрипти, а складні виносять в окремий файл.

Браузер завантажить його тільки перший раз і в подальшому, за правильного настроювання сервера, братиме зі свого кеша.

Завдяки цьому один і той же великий скрипт, що містить, наприклад, бібліотеку функцій, може використовуватися на різних сторінках без повного перезавантаження із сервера.

Порядок виконання роботи

1. Вивести простий alert:

– створити сторінку, яка виводить «Я – JavaScript!»;

– створити її на диску, відкрити в браузері та переконатись, що все працює.

2. Вивести alert зовнішнім скриптом:

– узяти розв’язок попереднього пункту, вивести alert і винести скрипт у зовнішній файл alert.js, який розташувати в тій самій директорії;

– відкрити сторінку і перевірити, що висновок повідомлення все ще працює.

3. Надати відповідь, який скрипт виконається першим.

Нижче показано два скрипти small.js і big.js.

Якщо припустити, що small.js завантажується набагато швидше, ніж big.js, то який виконається першим?

```
<script src="big.js"></script>
<script src="small.js"></script>
```

А в такому випадку?

```
<script async src="big.js"></script>
<script async src="small.js"></script>
```

Зміст звіту

1. Назва та мета роботи.
2. Методика проведення роботи з графічними результатами.
3. Письмові відповіді на контрольні питання.

Контрольні питання

1. У чому полягає суть тегу SCRIPT?
2. Описати сучасну розмітку для тегу SCRIPT.
3. Що таке зовнішні скрипти?
4. У чому полягає суть асинхронних скриптів: defer/async?

Література: [1, 2, 4].

Лабораторна робота № 2

Тема. Побудова скрипту з використанням основних операторів, циклів і функцій мови JavaScript

Мета: ознайомлення та набуття практичних навичок створення скрипту з використанням основних операторів, циклів і функцій мови JavaScript.

Короткі теоретичні відомості

У JavaScript можна оголошувати змінні для зберігання даних за допомогою var. Технічно можна просто записати значення і без оголошення змінної, проте через деякі причин це не рекомендується.

Разом з оголошенням можна відразу присвоїти значення: var x = 10.

Змінні, які названі ВЕЛИКИМИ_ЛІТЕРАМИ, є константами, тобто ніколи не змінюються. Зазвичай їх використовують для зручності, щоб було менше помилок.

У JavaScript є логічні значення true (істина) і false (брехня). Оператори порівняння повертають їх. Рядки порівнюються побуквенно.

Значення різних типів наводяться до числа за порівняння, за винятком суворого рівності === (! ==).

Значення null і undefined рівні == один одному та не дорівнюють нічому іншому. В інших порівняннях (за участю ",") їх краще не використовувати, оскільки вони поводяться не як 0.

У js є 5 «примітивних» типів: number, string, boolean, null, undefined і 6-й тип – об’єкти object.

Оператор typeof x дозволяє з’ясувати, який тип знаходиться в x, повертаючи його у вигляді рядка.

Для операцій над логічними значеннями в JavaScript є || (АБО), && (І) і! (НЕ). Хоча вони і називаються «логічними», але в JavaScript можуть застосовуватися до значень будь-якого типу і повертають також значення будь-якого типу.

JavaScript підтримує три види циклів:

while – перевірка умови перед кожним виконанням.

do..while – перевірка умови після кожного виконання.

for – перевірка умови перед кожним виконанням, а також додаткові настройки.

Щоб організувати нескінченний цикл, використовують конструкцію while (true). При цьому він, як і будь-який інший цикл, може бути перерваний директивою break.

Якщо на цій ітерації циклу більше нічого не потрібно робити, але повністю припиняти цикл не слід – використовують директиву continue.

Обидві директиви підтримують «мітки», які ставляться перед циклом. Мітки – єдиний спосіб для break/continue вплинути на виконання зовнішнього циклу.

Мітки не дозволяють перейти в довільне місце коду, у JavaScript немає такої можливості.

Особливості вбудованих функцій

Конкретне місце, де виводиться модальне вікно з питанням – це зазвичай центр браузера, і зовнішній вигляд вікна вибирає браузер. Розробник не може на це впливати.

З одного боку – це недолік, оскільки не можна вивести вікно у своєму, особливо красивому дизайні. З іншого боку, перевага цих функцій порівняно з іншими складнішими методами взаємодії, які вивчатимо в подальшому, саме в тому, що вони дуже прості.

Це найпростіший спосіб вивести повідомлення або отримати інформацію від відвідувача. Тому їх використовують в тих випадках, коли простота важлива, а всякі «красивості» особливої ролі не грають.

Порядок виконання роботи

1. Провести роботу зі змінними:

– оголосити дві змінні: admin і name;

– записати в name рядок "Васьок";

– скопіювати значення з name в admin;

– вивести admin (має вивести «Васьок»).

2. Надати відповідь, чому дорівнюватиме x у цьому прикладі:

var a = 2;

var x = 1 + (a *= 2).

3. Додати сторінку, яка запитує ім’я і виводить його.

4. Відповісти на питання, чи видається alert у такому випадку:

```
if ("0") {
  alert( ‘Привіт’ );
}
```

5. Використовуючи конструкцію if..else, написати код, який запитуватиме: «Яка «офіційне» назва JavaScript?». Якщо відвідувач уводить «ECMAScript», то виводити «Вірно!», якщо щось інше – виводити «Не знаєте? «ECMAScript»! » (див. блок-схему):

6. Виконати перевірку if всередині діапазону.

Написати умову if для перевірки того факту, що змінна age знаходиться між 14 і 90 включно. «Включно» означає – у тому числі що кінці проміжку, тобто age може дорівнювати 14 або 90.

7. За допомогою циклу for вивести парні числа від 2 до 10.

8. Переписати код, замінивши цикл for на while, без зміни поведінки циклу.

```
for (var i = 0; i < 3; i++) {
  alert( "номер " + i + "!" );
}
```

9. Написати цикл if..else, що відповідає наступному switch:

```
switch (browser) {
  case ‘IE’:
    alert(‘О, та у вас IE!’);
    break;
  case ‘Chrome’:
  case ‘Firefox’:
  case ‘Safari’:
  case ‘Opera’:
    alert(Так, і ці браузери ми підтримуємо’);
    break;
  default:
    alert(Ми сподіваємося, що і в вашому браузері все ок!’);
}
```

10. Написати функцію min(a,b), яка повертає найменше з чисел a,b.

Приклад викликів:

```
min(2, 5) == 2
min(3, -1) == -1
min(1, 1) == 1
```

Зміст звіту

1. Назва та мета роботи.
2. Методика проведення роботи з графічними результатами.
3. Письмові відповіді на контрольні питання.

Контрольні питання

1. Для чого потрібні побітові оператори?
2. Які існують види перетворення типів для примітивів?
3. Подати синтаксис альтернативного оголошення та виклику функції «Function Expression».

4. Шо таке «контекст виконання» (execution context) для виклику функції?

5. Що може стек контекстів у js?

6. Описати іменований функціональний вираз в js.

Література: [1–3, 5, 7, 8].

Лабораторна робота № 3

Тема. Створення скрипту з використанням об’єктів, властивостей і методів у JavaScript

Мета: набуття практичних навичок створення скриптів з використанням об’єктів, властивостей і методів js.

Короткі теоретичні відомості

Властивості-функції називають «***методами***» об’єктів. Їх можна додавати і видаляти в будь-який момент, у тому числі і явним привласненням.

Для повноцінної роботи метод повинен мати доступ до даних об’єкта.

Для доступу до поточного об’єкта з методу використовується ключове слово this. Значенням this є об’єкт перед «точкою», в контексті якого викликаний метод. Використання this гарантує, що функція «працює» саме з тим об’єктом, в контексті якого викликана.

Через this метод може не тільки звернутися до будь-якої властивості об’єкта, а й передати посилання на сам об’єкт уцілому.

Будь-яка функція може мати в собі this. Абсолютно неважливо, чи оголошена вона в об’єкті, чи окремо від нього. Значення this називається контекстом виклику і буде визначено в момент виклику функції.

Якщо функція використовує this – це передбачає роботу з об’єктом. Але і прямий виклик func () технічно можливий. Зазвичай така ситуація виникає у разі помилки у розробці.

До того ж this отримує значення window, глобального об’єкта.

Контекст this ніяк «не прив’язаний» до функції, навіть якщо вона створена в оголошенні об’єкта. Щоб this передався, потрібно викликати функцію саме за допомогою точки (або квадратних дужок).

Порядок виконання роботи

1. Переписати підсумовування аргументів.

Нехай є функція sum, яка підсумовує всі елементи масиву:

```
function sum (arr) {
 return arr.reduce (function (a, b) {
   return a + b;
 });   }
alert (sum ([1, 2, 3])); // 6 (= 1 + 2 + 3)
```

Необхідно створити аналогічну функцію sumArgs(), яка буде підсумовувати всі свої аргументи:

```
function sumArgs () {
  / * Ваш код * /
}
alert (sumArgs (1, 2, 3)); // 6, аргументи подані через кому, без масиву
```

Для вирішення слідзастосувати метод reduce до arguments, використовуючи call, apply або позичання методу.

Примітка. Функція sum не знадобиться, вона наведена як приклад використання reduce.

2. Застосувати функцію до аргументів.

Написати функцію applyAll (func, arg1, arg2 ...), яка отримує функцію func і довільну кількість аргументів.

Вона повинна викликати func (arg1, arg2 ...), тобто передати в func усі аргументи, починаючи з другого, і повернути результат.

Наприклад:

```
// Застосувати Math.max до аргументів 2, -2, 3
alert (applyAll (Math.max, 2, -2, 3)); // 3
```

```
// Застосувати Math.min до аргументів 2, -2, 3
alert (applyAll (Math.min, 2, -2, 3)); // -2
```

Галузь застосування applyAll, звичайно, ширша, можна викликати її і зі своїми функціями:

```
function sum () {// підсумовує аргументи: sum (1,2,3) = 6
  return [] .reduce.call (arguments, function (a, b) {
    return a + b;
  });  }
function mul () {// перемножує аргументи: mul (2,3,4) = 24
  return [] .reduce.call (arguments, function (a, b) {
    return a * b;
  });  }
alert (applyAll (sum, 1, 2, 3)); // -> sum (1, 2, 3) = 6
alert (applyAll (mul, 2, 3, 4)); // -> mul (2, 3, 4) = 24
```

Зміст звіту

1. Назва та мета роботи.
2. Методика проведення роботи з графічними результатами.
3. Письмові відповіді на контрольні питання.

Контрольні питання

1. Що отримуватиме функція, якщо її запускати в контексті різних об’єктів?
2. Як потрібно викликати функцію, щоб this передався?
3. Яким буде результат виконання цього коду та чому?

```
var arr = ["a", "b"];
arr.push(function() {  alert( this ); })
arr[2](); // ?
```

Література: [1–2, 4, 9].

Лабораторна робота № 4

Тема. Основи роботи з масивами та їх методами у JavaScript

Мета: набуття практичних навичок роботи з масивами та їх методами у js.

Короткі теоретичні відомості

Масив – це різновид об’єкта, який призначений для зберігання пронумерованих значень і пропонує додаткові методи для зручного маніпулювання такою колекцією.

Вони зазвичай використовуються для зберігання впорядкованих колекцій даних, наприклад, списку товарів на сторінці, студентів у групі тощо.

Синтаксис для створення нового масиву – квадратні дужки зі списком елементів усередині.

Порожній масив:

```
var arr = [];
```

Масив fruits із трьома елементами:

```
var fruits = [ "Яблуко", "Апельсин", "Слива"];
```

Елементи нумеруються, починаючи з нуля.

Щоб отримати потрібний елемент з масив, указується його номер у квадратних дужках:

```
 var fruits = [ "Яблуко", "Апельсин", "Слива"];
alert (fruits [0]); // Яблуко
alert (fruits [1]); // Апельсин
alert (fruits [2]); // Слива
```

Елемент можна завжди замінити:

```
fruits [2] = ‘Груша’; // тепер [ "Яблуко", "Апельсин", "Груша"]
```

Або додати:

```
fruits [3] = ‘Лимон’; // тепер [ "Яблуко", "Апельсин", "Груша", "Лимон"]
```

Загальна кількість об’єктів, що зберігаються в масиві, міститься в його властивості length:

```
var fruits = [ "Яблуко", "Апельсин", "Груша"];
alert (fruits.length); // 3
```

За допомогою alert можна вивести і масив цілком, при цьому його елементи будуть подані через кому:

```
var fruits = ["Яблуко", "Апельсин", "Груша"];
alert (fruits); // Яблуко, Апельсин, Груша
```

У масиві може зберігатися будь-яка кількість елементів будь-якого типу, у тому числі, рядки, числа, об’єкти.

Одне із застосувань масиву – це **черга**. У класичному програмуванні так називають упорядковану колекцію елементів, у якій елементи додаються в кінець, а обробляються – з початку.

Дуже близька до черги ще одна структура даних: **стек**. Це така колекція елементів, у якій нові елементи додаються в кінець і беруться з кінця.

Масиви в JavaScript можуть містити як елементи інші масиви. Це можна використовувати для створення багатовимірних масивів, наприклад матриць.

Сучасні інтерпретатори намагаються оптимізувати їх і зберігати в пам’яті не у вигляді хеш-таблиці, а у вигляді безперервної області пам’яті, по якій легко «пробігтися» від початку до кінця.

Методи масивів:

push / pop, shift / unshift, splice – для додавання і видалення елементів;

join / split – для перетворення рядка на масив і навпаки;

slice – копіює ділянку масиву;

sort – для сортування масиву. Якщо не передати функцію порівняння - сортує елементи як рядки;

reverse – змінює порядок елементів на зворотний;

concat – об’єднує масиви;

indexOf/lastIndexOf – повертають позицію елемента в масиві (не підтримуються в IE8-).

Додатково:

Object.keys (obj) – повертає масив властивостей об’єкта.

Порядок виконання роботи

1. Створити масив:

– створити масив styles з елементами «Джаз», «Блюз»;

– додати в кінець значення «Рок-н-Ролл»;

– замінити передостаннє значення з кінця на «Класика». Код заміни передостаннього значення має працювати для масивів будь-якої довжини;

– видалити перше значення масиву і вивести його alert;

– додати на початок значення «Реп» і «Реггі».

2. Отримати випадкове значення з масиву:

– написати код для виведення alert випадкового значення з масиву:

```
var arr = ["Яблуко", "Апельсин", "Груша", "Лимон"];
```

– код для генерації випадкового цілого від min to max включно:

```
var rand = min + Math.floor (Math.random () * (max + 1 - min));
```

3. Додати клас у рядок.

Нехай в об’єкті є властивість className, яка містить список «класів» – слів, розділених пропуском:

```
var obj = {   className: ‘open menu’   }
```

Необхідно створити функцію addClass (obj, cls), яка додає до списку клас cls, але тільки якщо його там ще немає:

```
addClass (obj, ‘new’); // obj.className = ‘open menu new’
addClass (obj, ‘open’); // без змін (клас вже існує)
addClass (obj, ‘me’); // obj.className = ‘open menu new me’
alert (obj.className); // "open menu new me"
```

Ваша функція не повинна додавати зайвих пробілів.

4. Фільтрація масиву «на місці».

Створити функцію filterRangeInPlace (arr, a, b), яка отримує масив з числами arr і видаляє з нього усі числа в межах дії a..b. Тобто, перевірка має вигляд a ≤ arr [i] ≤ b. Функція повинна змінювати сам масив і нічого не повертати. Наприклад:

```
arr = [5, 3, 8, 1];
```

filterRangeInPlace (arr, 1, 4); // видалені числа поза діапазоном 1..4

alert (arr); // масив змінився: залишилися [3, 1]

5. Сортування об’єктів.

Написати код, який сортує масив об’єктів people по полю age.

Наприклад:

```
var vasya = {name: "Вася", age: 23};
var masha = {name: "Маша", age: 18};
var gleb = {name: "Глеб", age: 6};
var people = [vasya, masha, gleb];
... ваш код ...
// тепер people: [gleb, masha, vasya]
alert (people [0] .age) // 6
```

Виведіть список імен у масиві після сортування.

Зміст звіту

1. Назва та мета роботи.
2. Методика проведення роботи з графічними результатами.
3. Письмові відповіді на контрольні питання.

Контрольні питання

1. Який метод дозволяє перетворити рядок на масив, розділивши його за розділювачем s?
2. Назвати ще один синтаксис для створення масиву, окрім квадратних дужок [].
3. Чи може метод splice вставляти елементи без видалення? Якщо так, то що для цього потрібно?
4. Що виконує метод slice(begin, end)?
5. Чи однаковий синтаксис методу slice для рядків і для масивів?

Література: [4, 6–8].

Лабораторна робота № 5

Тема. Використання функцій-конструкторів об’єктів і дескрипторів їх властивостей у JavaScript

Мета: набуття практичних навичок роботи з функціями-конструкторами об’єктів і дескрипторами їх властивостей у js.

Короткі теоретичні відомості

Створення об’єктів через "new"

Звичайний синтаксис {...} дозволяє створити один об’єкт. Але найчастіше потрібно створити багато однотипних об’єктів.

Для цього використовують «функції-конструктори», запускаючи їх за допомогою спеціального оператора **new**.

Конструктор

Конструктором є будь-яка функція, викликана за допомогою new.

Наприклад:

```
function Animal (name) {
  this.name = name;
  this.canWalk = true;
}
var animal = new Animal ( "їжачок");
```

Зауважимо, що технічно будь-яка функція може бути використана як конструктор. Тобто, будь-яку функцію можна викликати за допомогою new. Додатково вказувати, що вона – конструктор, не потрібно.

Але, щоб виділити функції, задумані як конструктори, їх називають з великої літери: Animal, а не animal.

Детальніше – функція, запущена за допомогою new, чинить таке:

1. Створюється новий порожній об’єкт.
2. Ключове слово this отримує посилання на цей об’єкт.
3. Функція виконується. Зазвичай вона модифікує this (тобто цей новий об’єкт), додає методи, властивості.

4. Повертається this.

Унаслідок виклику new Animal ("їжачок"); отримуємо такий об’єкт:

```
animal = {
  name: "їжачок",
  canWalk: true
}
```

Інакше кажучи, під час виклику new Animal відбувається щось схоже (перший і останній рядок – це те, що робить інтерпретатор):

```
function Animal (name) {
  // this = {};
  // в this пишемо властивості, методи
  this.name = name;
  this.canWalk = true;
  // return this;
}
```

Тепер багаторазовими викликами new Animal з різними параметрами можна створити стільки об’єктів, скільки потрібно. Тому таку функцію називають конструктором – вона призначена для «конструювання» об’єктів.

Зазвичай конструктори нічого не повертають. Їх завдання – записати все, що потрібно, у this, який автоматично буде результатом.

Але якщо явний виклик return все ж є, то застосовується просте правило:

– у разі виклику return з об’єктом, буде повернутий він, а не this;

– у разі виклику return з примітивним значенням, воно буде відкинуте.

Порядок виконання роботи

1. Створити Calculator за допомогою конструктора.

Написати функцію-конструктор Calculator, яка створює об’єкт з трьома методами:

– метод read () запитує два значення за допомогою prompt і запам’ятовує їх у властивостях об’єкта;

– метод sum () повертає суму запам’ятованих властивостей;

– метод mul () повертає добуток запам’ятованих властивостей.

Приклад використання:

```
var calculator = new Calculator ();
calculator.read ();
alert ("Сума =" + calculator.sum ());
alert ("Твір =" + calculator.mul ());
```

2. Написати конструктор Calculator, який створює розгорнуті об’єкти-калькулятори:

– викликати calculate (str), який приймає рядок, наприклад «1 + 2», з жорстко заданим форматом «ЧИСЛО операція ЧИСЛО» (по одному пробілу навколо операції), і повертає результат. Розуміє плюс + і мінус -.

Приклад використання:

```
var calc = new Calculator;
alert (calc.calculate ( "3 + 7")); // 10
```

– додати калькулятору метод addMethod (name, func), який учить калькулятор нової операції. Він отримує ім’я операції name і функцію від двох аргументів func (a, b), яка повинна її реалізовувати.

Наприклад, додамо операції помножити «*», поділити «/» і піднести до ступеня «**»:

```
var powerCalc = new Calculator;
powerCalc.addMethod ( "*", function (a, b) {
  return a * b; });
powerCalc.addMethod ( "/", function (a, b) {
  return a / b; });
powerCalc.addMethod ( "**", function (a, b) {
  return Math.pow (a, b); });
var result = powerCalc.calculate ( "2 ** 3");
alert (result); // 8
```

Примітка. Підтримка дужок і складних математичних виразів у цьому

завданні не потрібна. Числа і операції можуть складатися з декількох символів. Між ними рівно один пробіл. Передбачити обробку помилок.

Зміст звіту

1. Назва та мета роботи.

2. Методика проведення роботи з графічними результатами.

3. Письмові відповіді на контрольні питання.

Контрольні питання

1. Чи зручно у функції-конструкторі оголошувати допоміжні локальні змінні та вкладені функції, які будуть видимі тільки всередині?

2. Чи можливі такі функції A і B у прикладі, що наведений нижче, де відповідні об’єкти a, b дорівнюють один одному?

```
function A () {...}
function B () {...}
var a = new A;
var b = new B;
alert (a == b); // true
```

Якщо так, то наведіть приклад коду з такими функціями.

3. Що буде результатом функції, якщо ця функція не вирішить повернути свій об’єкт?

4. Що поверне виклик return з об’єктом? Що поверне виклик return із чим завгодно, крім об’єкта?

Література: [9, 10, 12, 13].

Лабораторна робота № 6

Тема. Основи об’єктно-орієнтованого програмування в JavaScript

Мета: набуття практичних навичок об’єктно-орієнтованого програмування в js.

Короткі теоретичні відомості

У об’єктно-орієнтованій розробці описується те, що відбувається на рівні об’єктів, які створюються, змінюють свої властивості, взаємодіють один з одним і (у випадку з браузером) зі сторінкою.

Класом у об’єктно-орієнтованій розробці називають шаблон/програмний код, призначений для створення об’єктів і методів.

У JavaScript об’єкти часто використовуються просто як колекції, а класи можна організувати по-різному.

Об’єктно-орієнтоване програмування (ООП) – це наука про те, як робити правильну архітектуру. У неї є свої принципи, наприклад SOLID. Один з найважливіших принципів ООП – відділення внутрішнього інтерфейсу від зовнішнього.

Методи і властивості об’єкта поділяються на дві групи:

1. *Внутрішній інтерфейс* – це властивості і методи, доступ до яких може бути здійснений тільки за допомогою інших методів об’єкта, їх також називають «приватними» (є й інші терміни).

2. *Зовнішній інтерфейс* – це властивості і методи, доступні зовні об’єкта, їх називають «публічними».

Для керованого доступу до стану об’єкта використовують спеціальні функції, так звані «геттери» і «сеттери».

Для кращого контролю над властивістю його роблять приватним, а запис значення здійснюється із застосуванням спеціального методу, який називають «*сетер*» (setter method). Для того, щоб надати можливість зовнішньому коду дізнатися його значення, створюється спеціальна функція – «*геттер*» (getter method).

Порядок виконання роботи

1. Модифікувати готовий код кавоварки, який подано нижче, додавши до нього публічний метод stop(), який зупинятиме кип'ятіння (за допомогою clearTimeout).

```
function CoffeeMachine (power) {
 this.waterAmount = 0;
 var WATER_HEAT_CAPACITY = 4200;
 var self = this;
 function getBoilTime () {
  return self.waterAmount * WATER_HEAT_CAPACITY * 80 / power;   }
 function onReady () {
  alert ( 'Кава готова!');   }
 this.run = function () {
  setTimeout (onReady, getBoilTime ());   };  }
```

Ось такий код повинен нічого не виводити:

```
var coffeeMachine = new CoffeeMachine (50000);
coffeeMachine.waterAmount = 200;
coffeeMachine.run ();
coffeeMachine.stop (); // кава приготовлена не буде
```

Примітки. Поточну температуру води обчислювати і зберігати не потрібно. Для розв'язання завдання знадобиться додати приватну властивість timerId, яка зберігатиме поточний таймер.

2. Написати об'єкт з «гетерами» і «сетерами».

Написати конструктор User для створення об'єктів:

– з приватними властивостями ім'я firstName та прізвище surname;

– з «сетерами» для цих властивостей;

– з «геттером» getFullName (), який повертає повне ім'я.

Повинен працювати так:

```
function User () {   / * Ваш код * / }
var user = new User ();
```

```
user.setFirstName ("Петро");
user.setSurname ("Козак");
alert (user.getFullName ()); // Петро Козак
```

Зміст звіту

1. Назва та мета роботи.
2. Методика проведення роботи з графічними результатами.
3. Письмові відповіді на контрольні питання.

Контрольні питання

1. Чи можна в JS створювати для зручності єдиний метод у контексті ООП, який називається так само, як властивість, і відповідає і за запис, і за читання?
2. Чи можна властивості, записані в this, уважати публічними?
3. Чи має клас-наслідувач доступ до приватних властивостей батьківського класу?
4. Які властивості класу називаються захищеними?

Література: [10–13].

Лабораторна робота № 7

Тема. Основи роботи з браузерними подіями в JavaScript: робота з мишкою, клавіатурою, завантаження скриптів і фреймів

Мета: набуття практичних навичок роботи з мишкою, клавіатурою, завантаженням скриптів і фреймів у js.

Короткі теоретичні відомості

Для реакції на дії користувача та внутрішньої взаємодії скриптів існують події. ***Подія*** – це сигнал від браузера про те, що щось сталося. Існує багато

видів подій.

Події миші:

click – відбувається, коли скликали на елемент лівою кнопкою миші;

contextmenu – відбувається, коли скликали на елемент правою кнопкою миші;

mouseover – виникає, коли на елемент наводиться миша;

mousedown і *mouseup* – коли кнопку миші натиснули або віджали;

mousemove – під час руху миші.

Події на елементах керування:

submit – відвідувач відправив форму <form>;

focus – відвідувач фокусується на елементі, наприклад натискає на <input>.

Клавіатурні події:

keydown – якщо відвідувач натискає клавішу;

keyup – якщо відвідувач відпускає клавішу.

Події документа:

DOMContentLoaded – якщо HTML завантажений і оброблений, DOM документа повністю побудований і доступний.

Події CSS:

transitionend – якщо CSS-анімація завершена.

Також є багато інших подій.

Призначення обробників подій

Події можна призначити обробник, тобто функцію, яка спрацює, як тільки подія відбулася. Саме завдяки обробникам JavaScript-код може реагувати на дії користувача.

Є кілька способів призначити події обробник.

1. Використання атрибута HTML: оброблювач може бути призначений прямо в розмітці, в атрибуті, який називається on <подія>.

2. Використання властивості DOM-об’єкта: можна призначати обробник, використовуючи властивість DOM-елемента on <подія>.

Якщо обробник заданий за допомогою атрибута, то браузер читає HTML-розмітку, створює нову функцію з вмісту атрибута і записує у властивість.

Доступ до елемента за допомогою this

Усередині обробника події this посилається на поточний елемент, тобто на той, на якому він спрацював. Це можна використовувати, щоб отримати властивості або змінити елемент.

addEventListener і removeEventListener

Методи addEventListener і removeEventListener є сучасним способом призначити або видалити обробник, і при цьому дозволяють використовувати скільки завгодно будь-яких оброблювачів.

Призначення обробника здійснюється викликом addEventListener із трьома аргументами:

element.addEventListener (event, handler [, phase]);

event – ім’я події, наприклад click;

handler – посилання на функцію, яку потрібно поставити оброблювачем;

phase – необов’язковий аргумент, «фаза», на якій обробник повинен спрацювати.

Видалення обробника здійснюється викликом removeEventListener.

Порядок обробки подій

Події можуть виникати не тільки почергово, а й «пачкою» по багато відразу. Можливо і таке, що під час обробки однієї події виникають інші, наприклад поки виконувався код для onclick – відвідувач натиснув кнопку на клавіатурі (подія keydown).

Головний потік

У кожному вікні виконується тільки один головний потік, який займається виконанням JavaScript, зарисовкою та роботою з DOM. Він виконує команди послідовно, може робити тільки одну справу одночасно і блокується з виведенням модальних вікон, таких як alert.

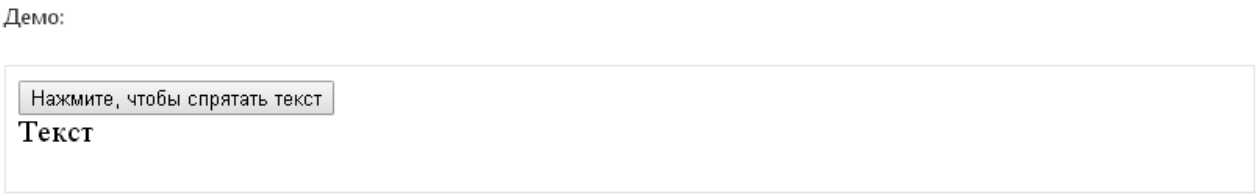
Черга подій

Якщо головний потік наразі зайнятий, то він не може терміново вийти із

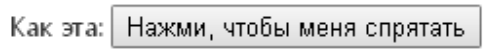
середини однієї функції і стрибнути в іншу, а потім третю. Тому використовується альтернативний підхід. Якщо відбувається подія, вона потрапляє в чергу. Усередині браузера безперервно працює «головний внутрішній цикл», який стежить за станом черги і обробляє події, запускає відповідні обробники і тощо. Іноді події додаються в чергу відразу пачкою.

Порядок виконання роботи

1. Використовуючи JavaScript, зробити так, щоб із кліком на кнопку зникав елемент з id="text".



2. Створити кнопку, при кліці на яку, вона приховуватиме сама себе.



3. Зробити сортування таблиці з кліком на заголовок, використовуючи при цьому делегування. Код не повинен змінюватися зі збільшенням кількості стовпців або рядків. Приклад:

Вік	**Ім’я**
1	Олексій
2	Петро
5	Вася
12	Євген

Примітка. Тип стовпця заданий атрибутом біля заголовка. Це необхідно, адже числа упорядковано інакше, ніж рядки. Відповідно, код це може використовувати. Для виконанні цього завдання допоможуть додаткові навігаційні посилання за таблицями.

Зміст звіту

1. Назва та мета роботи.

2. Методика проведення роботи з графічними результатами.

3. Письмові відповіді на контрольні питання.

Контрольні питання

1. Чи дозволяють сучасні браузери викликати підпроцеси Web Workers, які виконуються паралельно та можуть відправляти/приймати повідомлення, але не мають доступу до DOM?

2. Як називається найглибший елемент, який викликає подія, що є доступний як event.target?

3. Які браузерні події є синхронними?

4. Що буде виведено з кліком після виконання коду, якщо в змінній button знаходиться кнопка і спочатку оброблювачів на ній немає?

```
button.addEventListener ("click", function () {alert ("1");});
button.removeEventListener ("click", function () {alert ("1");});
button.onclick = function () {alert (2); };
```

Література: [12, 13].

2 КРИТЕРІЇ ОЦІНЮВАННЯ ЯКОСТІ ВИКОНАННЯ ЛАБОРАТОРНИХ РОБІТ СТУДЕНТАМИ

У 11-му семестрі студенти виконують 7 лабораторних робіт. Загальна кількість балів, яку отримують студенти за виконання лабораторних робіт, становить 28 балів – сума за захист виконаних лабораторних робіт (максимально по 4 бали на кожну лабораторну роботу).

Шкала оцінювання: національна та ECTS

<table>
<tr><td rowspan="2">Сума балів за всі види навчальної діяльності</td><td rowspan="2">Оцінка ECTS</td><td colspan="2">Оцінка за національною шкалою</td></tr>
<tr><td>Ддля екзамену, курсового проекту (роботи), практики</td><td>Для заліку</td></tr>
<tr><td>90–100</td><td>A</td><td>Відмінно</td><td rowspan="5">Зараховано</td></tr>
<tr><td>82–89</td><td>B</td><td rowspan="2">Добре</td></tr>
<tr><td>74–81</td><td>C</td></tr>
<tr><td>64–73</td><td>D</td><td rowspan="2">Задовільно</td></tr>
<tr><td>60–63</td><td>E</td></tr>
<tr><td>35–59</td><td>FX</td><td>Незадовільно з можливістю повторного складання</td><td>Не зараховано з можливістю повторного складання</td></tr>
<tr><td>0–34</td><td>F</td><td>Незадовільно з обов’язковим повторним вивченням навчальної дисципліни</td><td>Не зараховано з обов’язковим повторним вивченням навчальної дисципліни</td></tr>
</table>

СПИСОК ЛІТЕРАТУРИ

1. Гаевский А. Ю. 100 % самоучитель. Создание Web-страниц и Web-сайтов. HTML и JavaScript / А. Ю. Гаевский, В. А. Романовский. – М.: Наука, 2015. – 464 с.
2. Дронов В. JavaScript в Web-дизайне / В. Дронов. – М. : СПб: БХВ, 2014. – 880 с.
3. Кингсли-Хью К. Э. JavaScript 1.5: учебный курс / К. Э. Кингсли-Хью. – СПб : Питер, 2013. – 272 с.
4. Негрино Т. JavaScript для начинающих / Т. Негрино. – М. : Огни, 2013. – 544 с.
5. Федоров А. Г. JavaScript для всех / А. Г. Федоров. – М. : Машиностроение, 2012. – 384 с.
6. Флэнаган Д. Java Script. Подробное руководство / Д. Флэнаган. – М. : Символ-Плюс, 2013. – 1080 с.
7. Стефанов С. Java Script. Шаблоны / С. Стефанов. – М. : Символ-Плюс, 2011. – 272 с.
8. Марин Хауэрбеке. Элегантный JavaScript / Хауэрбеке Марин. – 178 с. [Электронный ресурс]. – Режим доступа: https://eloquentjavascript.net/
9. Эдди Османи. Изучаем паттерны проектирования JavaScrip [Электронный ресурс]. – Режим доступа: http://sd.blackball.lv/library/Learning_JavaScript_Design_Patterns_(2012).pdf.
10. Илья Кантор. Современный учебник JavaScript. Часть 1 / Кантор Илья. – М. : Самиздат, 2017. – 415 с.
11. Аксель Раушмайер. Говорить на JavaScript / Раушмайер Аксель. – O’Reilly Media, 2015. – 252 с.
12. Герд Вагнер. Создание клиентских веб-приложений на простом JavaScript / Вагнер Герд. – O’Reilly Media, 2014. – 222 с.
13. Эрик Эллиот. Программируем JavaScript-приложения / Эллиот Эрик. – O’Reilly Media, 2014. – 254 с.

Методичні вказівки щодо виконання лабораторних робіт з навчальної дисципліни «Прикладне програмування в комп’ютерних мережах» для студентів денної форми навчання зі спеціальності 123 – «Комп’ютерна інженерія». Частина ІІ

Укладач к. т. н., доц. О. Г. Славко

Відповідальний за випуск зав. кафедри КІС А. В. Луговой

Підп. до др. _____. Формат 60×84 1/16. Папір тип. Друк ризографія.
Ум. друк. арк. ____. Наклад _____ прим. Зам. №_____. Безкоштовно.

Видавничий відділ
Кременчуцького національного університету
імені Михайла Остроградського
вул. Першотравнева, 20, м. Кременчук, 39600