ФГБОУ ВО
«Воронежский государственный технический университет»

Кафедра автоматизированных и вычислительных систем

478-2015

ОСНОВЫ ВЕБ-ПРОГРАММИРОВАНИЯ НА JAVASCRIPT

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

к выполнению лабораторных работ № 1-4
по дисциплине «Проектирование и разработка
Web-приложений» для студентов направления
09.03.01 «Информатика и вычислительная техника»
профиля «Вычислительные машины, комплексы, системы
и сети» очной формы обучения

Воронеж 2015

Составитель канд. техн. наук М.Ю. Сергеев

УДК 681.32

Основы веб-программирования на языке JavaScript: методические указания к выполнению лабораторных работ № 1-4 по дисциплине «Проектирование и разработка Web-приложений» для студентов направления 09.03.01 «Информатика и вычислительная техника» профиля «Вычислительные машины, комплексы, системы и сети» очной формы обучения / ФГБОУ ВО «Воронежский государственный технический университет»; сост. М.Ю. Сергеев. Воронеж, 2015. 49 с.

Методические указания содержат теоретические и практические сведения для разработки динамических элементов веб-страниц с помощью языка JavaScript.

Предназначены для студентов четвертого курса.

Методические указания подготовлены в электронном виде в текстовом редакторе Microsoft Word 2007 и содержатся в файле Web_prog_1.doc.

Табл. 7. Ил. 16. Библиогр.: 4 назв.

Рецензент канд. техн. наук, проф. С.В. Тюрин

Ответственный за выпуск зав. кафедрой д-р техн. наук, проф. С.Л. Подвальный

Издается по решению редакционно-издательского совета Воронежского государственного технического университета

ВВЕДЕНИЕ

Данные методические разработки предполагают, что студенты в ходе выполнения практического курса дисциплины «Проектирование и разработка web-приложений» должны ознакомиться с основами создания интерактивных элементов веб-страниц с использованием языка веб-программирования JavaScript.

В ходе выполнения лабораторных работ студентам предстоит освоить базовые методы и принципы использования языка JavaScript для оформления интерактивных элементов веб-страниц. Также при выполнении лабораторных заданий студентам представится возможность ознакомиться с основами использования пользовательских библиотек JavaScript.

1. ЛАБОРАТОРНАЯ РАБОТА № 1. ОСНОВЫ JAVASCRIPT. РАБОТА С МАССИВАМИ

1.1. Общие методические указания по выполнению лабораторной работы

Цели работы:

– ознакомиться с основами программирования на JavaScript;

– освоить программирование разветвляющихся процессов и циклов на JavaScript.

Среда выполнения и отладки:

Текстовый редактор Notepad++, веб-браузер (Firebox, Internet Explorer, Opera или др.).

1.2. Теоретические сведения

Инструкция if/else

Инструкция **if/else** – главная конструкция для принятия решений в JavaScript. Она позволяет указать, что если (**if**) какое-то выражение является истинным (TRUE), будет выполне-

на определенная группа инструкций, в противном случае (**else**) выполнится другая группа. Это выглядит следующим образом:

```
if (условие) {
  группа операторов 1 (выполняется если условие –
        TRUE)
}
else {
  группа операторов 2 (выполняется если условие –
        FALSE)
}
```

Часть, начинающаяся с else является опциональной.

Пример 1. Написать программу, которая реализует вычисление y по следующей схеме:

$$y = \begin{cases} x*(a+b), \text{ если } x \le 5; \\ x/(a+b), \text{ если } x > 5. \end{cases}$$

Код web-страницы:

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=windows-1251">
<style>
  h2 {
        text-align: center;
  }

</style>
<script language="javascript" type="text/javascript">
  var x=5;
  var a=5;
```

```
  var b=7;
  var y;
</script>
</head>
 <body>
 <h2>Лабораторная работа № 1</h2>
<script language="javascript" type="text/javascript">
        document.write('Исходные данные<br>');
        document.write('x = '+x+'<br>');
        document.write('a = '+a+'<br>');
        document.write('b = '+b+'<br>');
          if (x<=5)
          {
                y=x*(a+b);
                document.write('Выбрана ветвь вычисления
1<br>');
          }
          else
          {
                y=x/(a+b);
                document.write('Выбрана ветвь вычисления
2<br>');
          }
        document.write('Результат: ');
        document.write('y = '+y.toFixed(3));

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

Инструкция switch/case

Инструкция **switch** похожа на инструкции типа **if/else**. Она используется при необходимости сравнить одну переменную с несколькими значениями и сделать, что-то зависящее от

самого значения. Вместе со **switch** задействуются следующие дополнительные ключевые слова: **case**, **break** и **default**. Это выглядит следующим образом:

```
switch (avariable) {
        case 1 :
                группа операторов 1;
                break;
        case 2 :
                группа операторов 2;
                break;
        case 3 :
                группа операторов 3;
                break;
        default:
                группа операторов (выполняется, если все
                case возвращают FALSE);
}
```

Каждое выражение **case** в инструкции **switch** сравнивает переменную **switch** со значением **case**, которое может быть строкой. Если данные значения равны, выполняются инструкции, следующие за выражением **case**, после чего выражение **break** отсылает скрипт к первой инструкции после закрывающей фигурной скобки **switch**. Если ни одно из выражений **case** не было истинным, выполняются инструкции, следующие за выражением **default**, а скрипт завершает инструкцию **switch**.

Объект Math

Объект **Math** обеспечивает доступ к различным математическим константам и функциям. Он существует в единственном экземпляре и потому не имеет конструктора. Соответственно все его свойства и методы являются статическими и должны вызываться обращением к объекту **Math**, а не его реа-

лизациям. Прототипа объект **Math** не имеет. Свойства объекта указаны в табл. 1, методы – в табл. 2.

Таблица 1

Свойства объекта Math

Свойство	Описание
E	Основание натуральных логарифмов e.
LN10	Число ln 10
LN2	Число ln 2
LOG10E	Число lg e
LOG2E	Число $\log_2 e$
PI	Число π
SQRT1_2	Квадратный корень из 1/2
SQRT2	Квадратный корень из 2

Таблица 2

Методы объекта Math

Метод	Описание
abs	Возвращает абсолютную величину аргумента
acos	Возвращает арккосинус аргумента
asin	Возвращает арксинус аргумента
atan	Возвращает арктангенс аргумента
atan2	Возвращает арктангенс частного от деления аргументов
ceil	Возвращает наименьшее целое число, большее или равное аргументу

Продолжение табл. 2

cos	Возвращает косинус аргумента.
exp	Возвращает экспоненту аргумента.
floor	Возвращает наибольшее целое число, меньшее или равное аргументу.
log	Возвращает натуральный логарифм аргумента.
max	Возвращает наибольший из аргументов.
min	Возвращает наименьший из аргументов.
pow	Возводит первый аргумент в степень, заданную вторым.
random	Генерирует случайное число в диапазоне от 0 до 1.
round	Округляет аргумент до ближайшего целого числа.
sin	Возвращает синус аргумента.
sqrt	Возвращает квадратный корень из аргумента.
tan	Возвращает тангенс аргумента.

Массивы

Для создания массива и сохранения в нем каких-либо единиц сначала следует объявить его имя (как и в случае с переменной), а затем приложить к имени список значений, разделенных запятыми: каждое значение представляет одну из единиц списка.

Чтобы обозначить массив, следует поместить список единиц в квадратные скобки – [ ]. Например, чтобы создать массив, содержащий названия семи дней недели, следует написать такой код:

var days = ['Mon', "rues', 'Wed', 'Thurs', 'Fri', 'Sat', 'Sun'];

Уникальный номер, называемый индексом, указывает позицию каждого элемента в массиве. Чтобы получить доступ к отдельно взятому элементу, следует использовать его индекс. Массивы индексируются с нуля, это означает, что первый элемент имеет индекс 0, а второй – 1.

Поскольку индексный номер последнего элемента массива всегда меньше общего числа элементов в массиве, вы должны знать, сколько единиц в массиве, чтобы получить доступ к последней из них. К счастью, эта задача несложна, поскольку среди свойств массива есть его длина – общее число единиц. Чтобы получить доступ к свойству «длина», следует набрать после имени массива точку и слово length, например, days.length возвращает количество единиц в массиве, называемом days (если создан иной массив, например, playList, то получите длину в таком виде: playList.length). Таким образом, можно получить доступ к значению, сохраненному в массиве последним:

```
days[days.length-1]
```

Также возможно использовать переменную, содержащую номер в качестве индекса:

```
var i = 0;
alert(days[i]);
```

Циклы

На языке программистов многократное выполнение одной и той же задачи называется циклом. Поскольку циклы очень часто встречаются в программировании, JavaScript предлагает несколько их типов. Все они делают одно и то же, но немного разными способами.

Циклы While

Цикл while повторяет отрезок кода, пока остается истинным определенное условие. Базовая структура цикла while такова:

```
while (условие)  {
//   повторяющийся JavaScript
}
```

Первая строка вводит утверждение while. Условие помещается в скобки, следующие за ключевым словом while. Интерпретатор JavaScript выполняет весь код, находящийся в скобках, если условие истинно.

Однако в отличие от условного выражения, когда интерпретатор JavaScript достигает закрывающей скобки утверждения while, он вместо перехода к следующей строке программы возвращается к началу цикла while и тестирует условие повторно. Если условие вновь оказывается верным, интерпретатор опять выполняет код JavaScript. Процесс продолжается до тех пор, пока условие не станет ложным; после этого программа переходит к выполнению выражения, следующего за циклом.

Циклы for

JavaScript предлагает еще один тип цикла – for. Циклы for обычно используются для повторения серии шагов определенное количество раз. Поэтому они часто включают особую переменную цикла, условие и способ изменения переменной цикла. Во многих случаях цикл for помогает достичь тех же целей, что и цикл while, но с использованием меньшего количества строк кода. Вот, например, код предыдущего примера с использованием цикла for:

```
for  (var num=l; num<=100;  num++)   {
document.write('Number '  + num +  '<br>');
}
```

В первой части (var num = 1;) инициализируется переменная цикла. Это происходит только один раз в самом начале работы цикла. Вторая часть условия предназначена для тестирования того, нужно ли в очередной раз выполнять код. Третья часть – действие, происходящее в конце каждого оборота цикла (обычно это изменение значения переменной цикла) до тех пор, пока условие теста не станет ложным и цикл не закончится.

Циклы Do/While

Есть еще один, менее распространенный тип циклов, известный как do/while. Они работают практически так же, как циклы while. Базовая структура выглядит так:

```
do {
// повторяющийся javascrip
} while (условие);
```

В циклах данного типа условный тест происходит в конце, после того как цикл выполнен. В результате выполнение кода JavaScript, находящегося в фигурных скобках, происходит хотя бы один раз. Даже если условие уже не выполняется, тест более не проводится после выполнения кода.

Пример 2. Написать программу, выводящую на экран содержимое массива из 10 целых чисел.

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=windows-1251">

<script language="javascript" type="text/javascript">
  var mas = [1, 7, 4, 2, 3, 2, 9, 10, 5, 8];

  function printmas(mas)
  {
```

```
            for (var i=0; i<mas.length; i++)
                  document.write(mas[i]+' ')
            document.write('<br>');
    }

</script>
</head>
 <body>
 <h2>Лабораторная работа № 1</h2>
<script language="javascript" type="text/javascript">
            document.write('Массив:<br>');
            printmas(mas);
</script>
</body>
</html>
```

1.3. Задания на лабораторную работу № 1

Задание 1. Составить программу вычисления функции (табл. 3).

Таблица 3

Варианты задания № 1

Номер варианта	Функция
1	$y = \begin{cases} a + x^2 - b; \ если\, x \le 10; \\ a - x + b \cdot x; если\, x > 10. \end{cases}$
2	$y = \begin{cases} (a-b) \cdot x + b \cdot x; \ если\, x > 13.4; \\ a + x \cdot (b-a); если\, x \le 13.4. \end{cases}$
3	$y = \begin{cases} x / b - a \cdot x; \ если\, x \ge 7.3; \\ x / a - b \cdot x; если\, x < 7.3. \end{cases}$

Продолжение табл. 3

4	$y = \begin{cases} \dfrac{x}{(b-a)}; \ если\, x \geq 12; \\ \dfrac{(b-a)}{x}; если\, x < 12. \end{cases}$
5	$y = \begin{cases} \dfrac{x \cdot a}{b}; \ если\, x < 4.3; \\ \dfrac{x}{a \cdot b}; если\, x \geq 4.3. \end{cases}$
6	$y = \begin{cases} x^3 - a; \ если\, x < 2; \\ a - x^3; если\, x \geq 2. \end{cases}$
7	$y = \begin{cases} b \cdot x^2 - c \cdot x^3; \ если\, x \leq 1; \\ c \cdot x^3 - b \cdot x^2; если\, x > 1. \end{cases}$
8	$y = \begin{cases} x + a \cdot x^2; \ если\, x > 3; \\ b + c \cdot x^2; если\, x \leq 3. \end{cases}$
9	$y = \begin{cases} \dfrac{x-a}{b}; \ если\, x \geq 5; \\ a + b + x; если\, x < 5. \end{cases}$
10	$y = \begin{cases} \dfrac{a-x}{b}; \ если\, x \leq 3.4; \\ \dfrac{x-a}{b}; если\, x > 3.4. \end{cases}$
11	$y = \begin{cases} (a-b) \cdot x^2 + \dfrac{b}{x}; \ если\, x > 7; \\ \dfrac{a}{x} + x \cdot (b-a); если\, x \leq 7. \end{cases}$

Окончание табл. 3

12	$y = \begin{cases} \dfrac{a-b}{x};\ если\ x > 5.4; \\ x^3 \cdot (b-a);\ если\ x \le 3.4. \end{cases}$
13	$y = \begin{cases} (x+a) \cdot b^2;\ если\ x > 3; \\ (b+a) \cdot x^2;\ если\ x \le 3. \end{cases}$
14	$y = \begin{cases} \dfrac{x}{b \cdot a};\ если\ x \ge 2; \\ \dfrac{b \cdot a}{x};\ если\ x < 2. \end{cases}$
15	$y = \begin{cases} (a-x) \cdot b^2;\ если\ x > 4; \\ (b-x) \cdot a^2;\ если\ x \le 4. \end{cases}$
16	$y = \begin{cases} x/b + a/x;\ если\ x \ge 3; \\ x/a + b/x;\ если\ x < 3. \end{cases}$
17	$y = \begin{cases} (a-b) \cdot x + (b+a) \cdot x^2;\ если\ x > 1; \\ x^2 \cdot (a-b) + x \cdot (b-a);\ если\ x \le 1. \end{cases}$
18	$y = \begin{cases} a \cdot b \cdot x;\ если\ x \ge 1; \\ a/b/x;\ если\ x < 1. \end{cases}$

Задание 2. Составить программу вычисления функции с использованием методов объекта Math (описание свойств и методов приведено в табл. 1-2). Варианты заданий приведены в табл. 4.

Таблица 4

Варианты второго задания

Номер варианта	Функция
1	$y = \begin{cases} \lvert x \rvert, \text{ если } a = 1; \\ \sin(x), \text{если } a = 2; \\ a\cos(x), \text{если } a = 3. \end{cases}$
2	$y = \begin{cases} a\sin(x), \text{ если } a = 1; \\ x^3, \text{если } a = 2; \\ e^x, \text{если } a = 3. \end{cases}$
3	$y = \begin{cases} \lvert x \rvert, \text{ если } a = 1; \\ \ln x, \text{если } a = 2; \\ \cos(x), \text{если } a = 3. \end{cases}$
4	$y = \begin{cases} tg\,(x), \text{ если } a = 1; \\ x^{-2}, \text{если } a = 2; \\ \sin(x), \text{если } a = 3. \end{cases}$
5	$y = \begin{cases} x^3, \text{ если } a = 1; \\ x^{-2}, \text{если } a = 2; \\ \sqrt{x}, \text{если } a = 3. \end{cases}$
6	$y = \begin{cases} \sqrt{x}, \text{ если } a = 1; \\ x^4, \text{если } a = 2; \\ \sin(x), \text{если } a = 3. \end{cases}$
7	$y = \begin{cases} \lvert x \rvert, \text{ если } a = 1; \\ 2^x, \text{если } a = 2; \\ \cos(x), \text{если } a = 3. \end{cases}$

Продолжение табл. 4

8	$y = \begin{cases} tg(x), \text{ если } a = 1; \\ x^5, \text{если } a = 2; \\ \lvert x \rvert, \text{если } a = 3. \end{cases}$
9	$y = \begin{cases} \cos(x), \text{ если } a = 1; \\ tg(2 \cdot x), \text{если } a = 2; \\ \sin(x), \text{если } a = 3. \end{cases}$
10	$y = \begin{cases} \sin(x), \text{ если } a = 1; \\ x^{-3}, \text{если } a = 2; \\ \sqrt{x}, \text{если } a = 3. \end{cases}$
11	$y = \begin{cases} \sin(2 \cdot x), \text{ если } a = 1; \\ e^{-x}, \text{если } a = 2; \\ x^3, \text{если } a = 3. \end{cases}$
12	$y = \begin{cases} tg(x), \text{ если } a = 1; \\ \sqrt{2 \cdot x}, \text{если } a = 2; \\ \cos(x), \text{если } a = 3. \end{cases}$
13	$y = \begin{cases} \lvert x \rvert, \text{ если } a = 1; \\ \sin(x^2), \text{если } a = 2; \\ \cos^2(x), \text{если } a = 3. \end{cases}$
14	$y = \begin{cases} \sqrt{x}, \text{ если } a = 1; \\ x^3, \text{если } a = 2; \\ tg(2 \cdot x), \text{если } a = 3. \end{cases}$
15	$y = \begin{cases} \cos(2 \cdot x), \text{ если } a = 1; \\ e^{-2 \cdot x}, \text{если } a = 2; \\ x^4, \text{если } a = 3. \end{cases}$

Окончание табл. 4

16	$y = \begin{cases} \lvert x \rvert, \text{ если } a = 1; \\ a^x, \text{если } a = 2; \\ \cos(a \cdot x), \text{если } a = 3. \end{cases}$
17	$y = \begin{cases} tg(3 \cdot x), \text{ если } a = 1; \\ a^x, \text{если } a = 2; \\ e^{ax}, \text{если } a = 3. \end{cases}$
18	$y = \begin{cases} x^{-1}, \text{ если } a = 1; \\ \sin(a \cdot x), \text{если } a = 2; \\ \sqrt{x}, \text{если } a = 3. \end{cases}$

Задание 3. Составить программу, которая:

– создает и выводит на экран массив из 7 – 10 целых элементов;

– рассчитывает сумму и произведение элементов массива;

– сортирует массив по возрастанию;

– добавляет в массив два элемента с помощью методов splice(), unshift() и push();

– удаляет из массива элементы с помощью методов splice(), shift() и pop().

2. ЛАБОРАТОРНАЯ РАБОТА № 2. РАБОТА С ЧИСЛАМИ И ДАТАМИ

2.1. Общие методические указания по выполнению лабораторной работы

Цели работы:

– освоить методы работы с числовыми данными в JavaScript;

– изучить форматы представления даты и времени в JavaScript.

Среда выполнения и отладки:

Текстовый редактор Notepad++, веб-браузер (Firebox, Internet Explorer, Opera или др.).

2.2. Теоретические сведения

Замена строки числом

Для преобразования строки в число JavaScript предлагает несколько способов.

Number() преобразует любую переданную ему строку в число:

```
var a = '3';
a = Number(a);  // a стало теперь числом 3
```

Метод **parseInt()** также превращает строку в число. Однако, в отличие **Number()**, **parseInt()** попытается превратить в число даже буквенную последовательность, если она начинается с чисел. Данная команда может быть удобна, если имеется строка вроде «20 лет» в качестве ответа на вопрос о чьем-то возрасте:

```
var age =  '20 лет';
age = parseInt(age,  10);  // 20
```

Метод **parseInt()** ищет в начале последовательности число, символ + или - и продолжает искать числа, если встречает не-число. Итак, в примере, приведенном выше, он возвращает число 20 и игнорирует остаток последовательности: «лет».

Второй аргумент метода указывает, в какой системе счисления представлено число для поиска. Если указать 2, то будет искаться число в двоичной системе счисления, 8 – в восьмеричной и т.д.

parseFloat() похож на **parseInt()**, но он используется, если строка может содержать разделительную точку десятичной

дроби. Например, если имеется строка вида «4.5 акров», то можно использовать **parseFloat()** для возвращения всего значения, включая десятичные знаки:

```
var space =  '4.5 акров';
space = parseFloat(space);  // 4.5
```

Если бы использовался метод **parseInt()** в примере, рассмотренном выше, то в итоге получилось бы просто число 4, поскольку **parseInt()** возвращает только целые числа.

Чтобы подтвердить, что строка является числом, следует использовать метод **isNaN()**. Он берет строку в качестве аргумента и тестирует, является ли она числом. Если строка содержит что-нибудь, кроме знаков «плюс» или «минус» (для положительных и отрицательных чисел), за которыми следуют числа и возможные десятичные значения, она считается нечислом. Таким образом, «-23.25» – это число, а «24 км» – нет. Данный метод возвращает значение «истина» (если строка не является числом) или «ложь» (если она является числом).

Округление чисел

Возможно округлять число, используя метод **round()** с объектом Math:

```
Math.round(number)
```

Число (или переменная, содержащая число) передается методу round(), и он возвращает целое число. Если первоначальное число содержит после десятичного знака цифры до 5, оно округляется в меньшую сторону, например, 4,4 округляется до 4, а 4,5 – в большую сторону, до 5.

```
var decimalNum = 10.25;
var roundedNum = Math.round(decimalNum) ;  // 10
```

JavaScript предлагает еще два метода округления чисел: **Math.ceil()** и **Math.floor()**, которые схожи с методом **Math.round()**. Однако, Math.ceil() всегда округляет число в большую сторону (например, Math.ceil(4.0001) возвращает 5), тогда как Math.floor() всегда округляет число в меньшую сторону: Math.floor(4.99999) возвращает 4. Рекомендуется использовать мнемоническое правило: ceiling (потолок) — это вверху, а floor (пол) — внизу.

Форматирование вещественных чисел

В данном случае JavaScript предлагает метод toFixed(), который позволяет преобразовывать число в строку, совпадающую с желаемым числом с десятичными знаками. Чтобы пользоваться им, следует добавить после числа точку (или после имени переменной, содержащей число), а затем toFixed(2):

```
var cost = 10;
var printCost = '$'  + cost.toFixed(2);   //  $10.00
```

Число, присваиваемое методу toFixed(), определяет, сколько десятичных знаков нужно указать. В случае с валютой следует использовать 2, чтобы получить в итоге числа: 10.00 или 9.90; используя 3, в итоге получается 3 десятичных знака, например, 10.000 или 9.900.

Генерирование случайного числа

JavaScript предлагает метод Math.random() для генерирования случайных чисел. Он возвращает случайно сгенерированное число в интервале от 0 до 1. Можно выполнять или тестировать простые математические операции, генерируя целые числа от 0 и выше. Например, для генерирования чисел от 0 до 9 следует использовать следующий код:

```
Math.floor(Math.random()*10);
```

Если требуется получить случайное число от 1 до другого числа, следует просто умножить Random() на большее число этой области и добавить к целому 1. Например, если необходимо сымитировать броски игральной кости, чтобы получить числа от 1 до 6, то можно написать следующий код:

```
var roll = Math.floor(Math.random()*6 +1);   //1,2,3,4,5 или 6
```

Функция для выбора случайного числа

Если приходится часто пользоваться случайными числами, то можно применять простую функцию, помогающую при выборе случайного числа между любыми двумя числами, например, между 1 и 6 или 100 и 1000. Следующая функция вызывается с использованием двух аргументов; первый – минимальное возможное значение (например, 1), второй – максимальное возможное значение (например, 6):

```
function rndNum(from,  to)   {
  return Math.floor((Math.random()*(to - from +1))  + from);
}
```

Для использования этой функции следует добавить ее на веб-страницу и вызвать ее:

```
var diceRoll = rndNum(1,6);  // получает число между 1 и 6
```

Работа с датой и временем

Если требуется отслеживать актуальную дату или время, то можно воспользоваться специальным объектом JavaScript – **Date** (Дата). Он позволит определять год, месяц, день недели, час и даже больше. Для его использования следует создать переменную и сохранить в ней новый объект Date:

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

Команда new Date() создает объект Date, в котором содержатся текущие дата и время. Однажды создав его, можно получать доступ к различным образцам даты и времени, используя относящиеся к Date методы, перечисленные в табл. 5. Например, для получения текущего года можно использовать метод **getFullYear()**:

```
var now = new Date();
var year = now.getFullYear();
```

Таблица 5

Методы для доступа к данным объекта Date

Метод	**Что он возвращает**
getFullYear()	Год. Например, 2008
getMonth()	Месяц как целое число между 0 и 11:0 – январь, 11 – декабрь
getDate()	День месяца как число от 1 до 31
getDay()	День недели как число от 0 до 6: 0 – воскресенье, 6 – суббота
getHours()	Количество часов по 24-часовому циферблату (число от 0 до 23). Например, 11 вечера = 23
getMinutes()	Число минут от 0 до 59
getSeconds()	Число секунд от 0 до 59
getTime()	Общее количество секунд, начиная с полуночи 1.01.1970

Получение месяца

Чтобы получить месяц от объекта Date, следует использовать метод **getMonth()**, возвращающий номер месяца:

```
var now = new Date();
var month = now.getMonth();
```

Однако вместо возвращения номера месяца (1 – это январь), данный метод возвращает число на 1 меньше. Например,

январь – это 0, февраль – 1 и т. д. Если нужно возвратить номер, совпадающий с обычным пониманием месяцев, просто следует добавить 1:

```
var now = new Date();
var month = now.getMonth()+1;        // соответствует
                                     // реальной нумерации
                                     // месяца
```

В JavaScript не существует встроенной команды, сообщающей имя месяца. К счастью, способ нумерации в JavaScript удобен, когда нужно определить точное название месяца. Можно достичь этого, создав массив с названиями месяцев, а затем получать доступ к имени, используя его индекс:

```
var months = ['Январь', 'Февраль', 'Март', 'Апрель', 'Май', 'Июнь', 'Июль', 'Август', 'Сентябрь', 'Октябрь', 'Ноябрь', 'Декабрь'];
var now = new Date ();
var months = months[now.getMonth()];
```

В первой строке создается массив с двенадцатью названиями месяцев в том порядке, в котором они идут (январь – декабрь). Следует помнить, что для доступа к элементу массива используется индекс и что массивы нумеруются с 0. Итак, для доступа к первому элементу массива с названиями месяцев следует использовать months[0]. Таким образом, с помощью **getMonth()** можно получить число, используемое как индекс для массива месяцев, посредством чего легко узнать название этого месяца.

Получение дня недели

Метод **getDay()** возвращает день недели. Как и в методе **getMonth()**, интерпретатор JavaScript возвращает число на 1 меньше, чем номер дня: 0 – это воскресенье, первый день неде-

ли по принятой на Западе традиции, 6 – суббота. Поскольку для обычных людей название дня недели обычно важнее, чем его номер, можно использовать массив для сохранения названий дней и метод **getDay()** для доступа к отдельному дню в массиве:

```
var days = [ 'Воскресенье', 'Понедельник', 'Вторник', 'Среда', 'Четверг', 'Пятница', 'Суббота' ];
var now = new Date();
var dayOfWeek = days[now.getDay()];
```

Получение времени

Объект Date также содержит актуальное время, поэтому можно отобразить это время на веб-странице или использовать его, чтобы узнать, в какое время суток посетитель просматривал страницу.

Методы **getHours(), getMinutes()** и **getSeconds()** можно использовать для возвращения часов, минут и секунд. То есть, чтобы отобразить время на веб-странице, можно добавить следующий код в HTML туда, где требуется видеть время:

```
var now = new Date();
var hours = now.getHours();
var minutes = now.getMinutes() ;
var seconds = now.getSeconds();
document .write (hours + ":" + minutes + ":" + seconds);
```

Данный код производит вывод 6:35:56 для обозначения 6 часов, 35 минут и 56 секунд.

Создание не текущей даты

Используя метод **Date()**, также можно указывать дату и время в будущем и прошлом. Основной формат таков:

new Date(год, месяц, день, час, минута, секунда, миллисекунда);

Например, чтобы создать Date для полудня 1 января 2010 г., можно написать следующее:

var ny2010 = new Date(2010, 0, l, 12, 0, 0, 0);

Данный код переводится как «Создать новый объект Date для времени 1.01.2010, 12:00:00:00». Необходимыми параметрами являются только год и месяц. Если не требуется указывать точное время, можно опустить миллисекунды, секунды, минуты и т. д. Например, создание объекта «1 января 2010 года» выглядит так:

var ny2010 = new Date(2010, 0, 1);

Создание даты, являющейся определенным днем недели

Интерпретатор JavaScript считает дату количеством миллисекунд, истекших с 1 января 1970 г. Другими словами, создание даты – это присвоение значения, равного количеству миллисекунд для этой даты:

new Date(количество миллисекунд);

Итак, другой способ создания даты для 1.01.2010 таков:

var ny2010 = new Date(1262332800000);

Конечно, большинство людей – не живые калькуляторы, и дату человек воспринимает не так. Однако миллисекунды очень удобны при создании даты, отстоящей от другой даты на конкретный период времени. Например, при настройке cookie с использованием JavaScript следует указать время, через которое cookie должно быть удалено из браузера посетителя. Чтобы

гарантировать, что cookie исчезнет через неделю, необходимо указать одну неделю, начиная с сегодняшнего дня.

Для создания даты, отстоящей от сегодняшнего дня на одну неделю можно сделать следующее:

```
var now = new Date();                    // текущий момент
var nowMS =  now.getTime();              // миллисекунды, соответствующие
// текущему моменту
var week = 1000*60*60*24*7;              // вычисление количества
// миллисекунд в неделю
 var oneWeekFromNow = new Date(nowMS + week);
```

В первой строке текущие дата и время сохраняются в переменной now. Далее метод getTime() извлекает число миллисекунд, истекших с 1.01.1970 до сегодняшнего дня. Третья строка считает общее количество миллисекунд в неделе (1000 миллисекунд * 60 секунд * 60 минут * 24 часа * 7 дней). В итоге код создает новую переменную, добавляя число миллисекунд в неделе к сегодняшнему дню.

2.3. Задания на лабораторную работу № 2

Задание № 1. Составить программу для работы с массивом, заполнение которого осуществляется с помощью генератора случайных чисел.

Варианты задания №1

1. Составить программу, формирующую массив A[10] из целочисленных элементов в диапазоне 0..20. Вывести на экран массив и среднее арифметическое его элементов.

2. Составить программу, формирующую массив B[12] из целочисленных элементов в диапазоне -5..5. Вывести на экран массив и сумму его положительных элементов.

3. Составить программу, формирующую массив D[11] из целочисленных элементов в диапазоне -10..10. Вывести на экран массив и количество его отрицательных элементов.

4. Составить программу, формирующую массив T[9] из целочисленных элементов в диапазоне 0..10. Вывести на экран массив и количество элементов, больших числа 5.

5. Составить программу, формирующую массив S[10] из целочисленных элементов в диапазоне 0..15. Вывести на экран массив и количество четных элементов.

6. Составить программу, формирующую массив R[12] из целочисленных элементов в диапазоне 0..20. Вывести на экран массив и сумму его нечетных элементов.

7. Составить программу, формирующую массив Y[8] из целочисленных элементов в диапазоне -10..10. Вывести на экран массив и сумму его положительных элементов.

8. Составить программу, формирующую массив P[14] из целочисленных элементов в диапазоне 0..25. Вывести на экран массив и количество элементов, вошедших в диапазон [5..10].

9. Составить программу формирующую массив F[10] из целочисленных элементов в диапазоне 0..10. Вывести на экран массив и сумму элементов, меньших 5.

10. Составить программу формирующую массив Z[10] из целочисленных элементов в диапазоне -5..5. Вывести на экран массив и сумму модулей его элементов.

11. Составить программу формирующую массив D[11] из целочисленных элементов в диапазоне 0..20. Вывести на экран массив и сумму элементов, больших 10.

12. Составить программу формирующую массив W[12] из целочисленных элементов в диапазоне -5..5. Вывести на экран массив и произведение его положительных элементов.

13. Составить программу формирующую массив R[12] из целочисленных элементов в диапазоне 0..10. Вывести на экран массив и количество элементов, равных 5.

14. Составить программу формирующую массив T[10] из вещественных элементов в диапазоне -5..5. Вывести на экран массив и сумму модулей его отрицательных элементов.

15. Составить программу формирующую массив S[11] из целочисленных элементов в диапазоне -5..5. Вывести на экран массив и произведение его элементов с четными индексами.

16. Составить программу формирующую массив B[12] из вещественных элементов в диапазоне -7..7. Вывести на экран массив и сумму его элементов с нечетными индексами.

17. Составить программу формирующую массив R[10] из целочисленных элементов в диапазоне -10..10. Вывести на экран массив и количество четных положительных элементов.

18. Составить программу формирующую массив D[11] из целочисленных элементов в диапазоне -20..20. Вывести на экран массив и сумму нечетных отрицательных элементов.

3. ЛАБОРАТОРНАЯ РАБОТА № 3. РАБОТА СО СТРОКАМИ И РЕГУЛЯРНЫМИ ВЫРАЖЕНИЯМИ

3.1. Общие методические указания по выполнению лабораторной работы

Цели работы:

– освоить базовые функции JavaScript по работе со строками;

– ознакомиться с принципами построения регулярных выражений;

– изучить функции JavaScript по работе с регулярными выражениями.

Среда выполнения и отладки:

Текстовый редактор Notepad++, веб-браузер (Firebox, Internet Explorer, Opera или др.).

3.2. Теоретические сведения

Определение длины строки

Для получения длины строки можно использовать свойство length, которое возвращает длину строки в символах. Для определения числа символов в строке следует добавить точку после имени переменной, причем за точкой следует свойство length: name.length.

Изменение регистра строки

JavaScript предлагает два метода перевода целых строк в верхний или нижний регистр.

Метод **toUpperCase()** переводит все содержимое строки в верхний регистр. Например, следующий код переводит слово ‘Hello’ в верхний регистр и выводит его в виде сообщения ‘HELLO’:

```
var greetings=’Hello’;
alert(greetings.toUpperCase());
```

Чтобы перевести всю строку в нижний регистр, следует использовать метод **toLowerCase()**:

```
var greetings=’Hello’;
alert(greetings.toUpperCase());  //  hello
```

Ни один из этих методов на самом деле не изменяет самой строки, сохраненной в переменной, они просто возвращают строку либо только в нижнем, либо только в верхнем регистре. В рассмотренном примере greetings по-прежнему содержит «Hello» даже после появления предупреждения.

Поиск в строке: метод indexOf()

JavaScript предлагает несколько способов поиска слов, чисел или других серий знаков в строках.

Один из методов поиска строки – **indexOf()**. Порядок такой: после имени строковой переменной string печатается точка, затем **indexOf()** и в скобки вставляется искомая строка:

```
string.indexOf('строка, которую надо найти')
```

Метод **indexOf()** возвращает число: если искомая строка не найдена, то метод возвращает -1.

Когда метод **indexOf()** обнаруживает искомую строку, он возвращает число, равное начальной позиции строки. Рассмотрим пример, который послужит пояснением:

```
var quote = 'быть или не быть';
var searchPosition = quote.indexOf('Быть');  // возвращает 0
```

В данном случае **indexOf()** ищет фрагмент **быть** в строке **быть или не быть**. Длинная строка начинается с **быть**, поэтому **indexOf()** находит искомый фрагмент в самом начале. Первая позиция считается 0, вторая буква («ы») – 1, а третья (в данном случае «т») – 2.

Метод **indexOf()** начинает поиск с начала строки. Можно искать и с конца строки, используя метод **lastIndexOf()**. Например, в цитате из Шекспира слово **быть** встречается два раза, поэтому можно определить первое **быть** с помощью метода **indexOf()**, а второе – с использованием **lastIndexOf()**:

```
var quote = "быть или не быть";
var firstPosition = quote.indexOf('быть');  // возвращает 0
var lastPosition = quote.lastIndexOf('быть');  // возвращает
12
```

Извлечение части строки с помощью метода slice()

Чтобы извлечь часть строки, следует использовать метод **slice()**, который возвращает результат извлечения. Например, имеется строка http://www.sawmac.com и требуется исключить

часть http://. Один из способов сделать это — извлечь часть строки, следующей за http://, вот так:

```
var url =  'http://www.sawmac.com';
var domain = url.slice(7) ;  // www.sawmac.com
```

Метод **slice()** требует в качестве аргумента число, которое присваивается индексу первого символа извлекаемой строки. В данном примере (url.slice(7)) 7 обозначает восьмую букву в строке (не следует забывать, что первой букве соответствует индекс 0). Данный метод возвращает все символы, начиная со стартовой позиции, исходя из переданного в качестве аргумента индекса и до конца строки.

Также возможно извлечь определенное количество символов из строки, присвоив методу **slice()** второй аргумент. Вот базовая структура метода **slice()** в данном случае:

```
string.slice(start, end);
```

Стартовое значение – это число, указывающее первый знак извлекаемой строки; конечное значение может запутать, так как это не позиция последней буквы извлекаемой строки, а позиция последней буквы плюс 1. Например, если требуется извлечь первые 4 буквы строки **быть или не быть**, следует указать 0 как первый аргумент, а 4 – как второй. Как видно на рис. 3, 0 – это первая буква в строке, а 4 – пятая, но последняя указанная буква не извлекается из строки. Другими словами символ, указанный в качестве второго аргумента, никогда не извлекается из строки.

Замечание. Если требуется извлечь из строки определенное количество знаков можно просто добавить это число к стартовому значению. Например, чтобы вернуть первые 10 букв строки, первый аргумент будет 0 (первая буква) а второй – 0 + 10 или просто 10: slice(0,10).

Также возможно указывать отрицательные числа, например, quote.slice(-6,-1). Отрицательное число считается с конца строки.

Замечание. Если требуется извлечь строку, содержащую буквы, начиная с 6-й с конца и до последней, можно просто опустить конечный аргумент:

```
quote.slice(-6);
```

Регулярные выражения

JavaScript позволяет использовать регулярные выражения, чтобы находить в строках шаблоны. Регулярное выражение – это серия символов, образец шаблона, который требуется найти.

Для создания шаблона используются символы, как *, +, ? и \. Интерпретатор JavaScript сопоставляет их с реальными символами из строки: числами, буквами и т. д.

Для создания регулярного выражения в JavaScript необходимо создать объект, который состоит из серии символов между двумя слэшами. Например, для регулярного выражения, совпадающего со словом hello, следует напечатать программный код:

```
var myMatch = /hello/;
```

Существует несколько методов, используемых при работе со строками, которые имеют преимущество над регулярными. Базовый метод – **search()**. Он работает, как метод indexOf(), но вместо того, чтобы пытаться найти одну строку в другой, более крупной, он ищет в строке шаблон. Например, требуется отыскать «быть» в строке «быть или не быть». Ранее это делалось с помощью метода indexOf(). Можно сделать то же самое с помощью регулярного выражения:

```
var myRegEx = /быть/;                    // не надо заключать в кавычки
var quote =  'быть или не быть.';
var foundPosition = quote.search(myRegEx);    // возвращает 0
```

Если **search()** находит совпадение, он возвращает позицию первой совпавшей буквы, а если не находит, то возвращает -1. Так, в примере, приведенном выше, переменная foundPosition равна 0, поскольку «быть» начинается с самого начала строки (с первой буквы).

Регулярные выражения могут содержать различные символы для обозначения разных типов знаков. Например, точка (.) означает отдельный знак, любой; \w соответствует любой букве или числу (но не пробелам или символам $ или %). В табл. 6 приведены символы, наиболее часто используемые при поиске по шаблонам.

Таблица 6

Символы, используемые в регулярных выражениях

Символ	**Значение**
.	Один любой символ. Это может быть буква, число, пробел или другой знак.
\w	Любой знак, который может входить в слово. Это может быть буква (в обоих регистрах, цифра от 0 до 9, знак подчеркивания (_).
\W	Любой знак, который НЕ может входить в слово. То есть любые знаки, не соответствующие \w.
\d	Любая цифра от 0 до 9.
\D	Любой символ, кроме цифры. Протовоположно \d.
\s	Пробел, знак табуляции, знак возврата каретки и знак новой строки.
\ S	Все знаки, кроме знака табуляции, знака возврата каретки и знака новой строки. Противоположно \s.

Продолжение табл. 6

^	Начало строки. Полезно, если обязательным условием является то, что впереди искомой подстроки нет символов (то есть она начинает строку).
$	Конец строки. Полезно, если обязательным условием является то, что искомая подстрока должна заканчивать строку. К примеру, /com$/ соответствует подстрока «com», но только когда она занимает последние три символа строки. То есть /com$/ соответствует подстроке «com» в строке «infocom», но не в строке «communication».
\b	Пробел, начало строки, конец строки, любой другой символ, не являющийся буквой или цифрой (к примеру, +,= или'). Можно использовать \ b для обозначения начала и конца слова, даже если слово находится в начале или конце строки.
[ ]	Любой символ из помещенных в скобки. К примеру,[aeoun] совпадет с любой из букв в квадратных скобках. Для обозначения последовательностей символов можно использовать тире. Так, запись [a-z] ссылается на любую букву латинского алфавита в нижнем регистре. Запись [0-9] ссылается на цифры, она аналогична \d.
[^]	Все символы, исключая те, которые помещены в скобки. К примеру, [^aeuioAEUIO] ссылается на все символы, кроме гласных букв. [^0-9] ссылается на все символы, не являющиеся цифрами (аналогично \D).
|	Сошлется на символ, который справа от |, или же на символ слева. К примеру, a|b сошлется или на a, или на b, но не на оба сразу.

Окончание табл. 6

\	Используется, чтобы «блокировать» служебные символы регулярных выражений (\, *, ., / и т. д.). Это может быть необходимо, если подобный символ должен быть найден в строке. К примеру, «.» в синтаксисе регулярных выражений означает «любой символ» поэтому, если вы хотите включить символ точки в поиск, его нужно «блокировать»: «\.».

Группирование частей шаблона

Можно использовать круглые скобки, чтобы создавать в шаблоне подгруппы. Эти подгруппы очень удобны (если один из символов табл. 7 используется), чтобы найти совпадения со многими различными строками по одному и тому же шаблону.

Таблица 7

Символы для обозначения совпадений

Символ	**Совпадает**
?	Ноль или одно появление предыдущего символа. Он необязателен, но если он имеется, то должен быть только в одном экземпляре. Например, регулярное выражение colou?r совпадет и с color, и с colour, но не с colouur
+	Одно или больше появлений предыдущего символа. Он должен появиться как минимум один раз
*	Ноль или более появлений предшествующего символа. Он необязателен и может являться любое количество раз. Например, .* совпадает с любым количеством знаков
{n}	Точное число появлений предыдущего символа. Например, \d{3} совпадает только с 3 числами в строке
{n,}	n и более появлений предшествующего символа. Например, a{2,} совпадет с буквой a два и более раз: с сочетанием aa в слове aardvark или aaaa в слове aaaahhhh

Продолжение табл. 7

{n,m}	Предшествующий символ появляется минимум n раз, но не более m раз. То есть \d{3,4} совпадет с 3 или 4 числами в строке (но не с двумя и не с пятью числами в строке)

Совпадение с шаблоном

Метод **search()**, описанный на ранее, – один из способов узнать, содержит ли строка шаблон регулярного выражения. Метод **match()** работает по-другому. Возможно использовать его со строкой, не только чтобы проверить, существует ли в данной строке шаблон – также можно извлечь этот шаблон, чтобы позже использовать его в других скриптах.

Следующий код находит и извлекает URL с использованием **match()**:

```
// Создание переменной, содержащей строку с URL
var text='Мой web-сайт это www.mysite.com';
// Создание регулярного выражения '
var urlRegex = /((\bhttps?:\/\/) | (\bwww\.))\S*/
// Нахождение совпадения с регулярным выражением в строке
var url = text.match(urlRegex);
alert (url [0] ) ;  // www.mysite.com
```

Сначала код создает переменную, в которой сохраняется строка, включающая URL www.mysite.com. Далее задается регулярное выражение для поиска URL. Наконец, к строке применяется метод **match()**. Функция **match()** – это метод для строк, поэтому все начинается с имени переменной, содержащей строку, потом к ней добавляется точка, а затем **match()**. Таким образом методу **match()** передается регулярное выражение для поиска совпадения.

В примере, данном выше, переменная url содержит результат совпадения. Если шаблон регулярного выражения не

найден в строке, в результате выдается специальное значение JavaScript, называемое **null**. Если совпадение происходит, то скрипт возвращает массив, первым значением которого является совпавший текст. Например, здесь переменная url содержит массив, и в первом элементе массива записан совпавший с шаблоном текст. В данном случае url[0] содержит www.mysite.com.

Совпадение с каждым образцом шаблона

Метод **match()** работает двумя различными способами в зависимости от того, как было настроено регулярное выражение. В примере, приведенном выше, метод возвращает массив с первым образцом совпавшего текста. Так, если имеется длинная строка, содержащая много URL, возвращается только первый найденный URL.

Однако также можно задействовать свойство регулярного выражения **global** для поиска в строке более чем одного совпадения.

Поиск становится глобальным, когда добавляется g в конец регулярного выражения.

```
var urlRegex = /((\bhttps?:\/\/)|(\bwww\.))\S*/g
```

Следует обратить внимание, что **g** находится за пределами / (используемого для закрывания шаблона). Это регулярное выражение осуществляет глобальный поиск. Когда оно используется с методом **match()**, то ищет все совпадения в строке и возвращает массив совпавших фрагментов текста.

Возможно переписать код из предыдущего примера, используя глобальный поиск, следующим образом:

```
// Создание переменной, содержащей строку
// с гиперссылкой
var text='Не так уж и много замечательных сайтов, подобных www.mysite.com и www.mynewsite.com';
```

```
// Создание регулярного выражения
// с пометкой глобального поиска
var urlRegex = /((\bhttps?:\/\/)|(\bwww\.))\S*/g
// Нахождение совпадения с регулярным выражением в
// строке
var url = text.match(urlRegex);
alert(url[0]); // www.mysite.com
alert(url[1]);  // www. mynewsite.com
```

Можно определить количество совпадений, получив доступ к свойству **length** результирующего массива: url.length. В данном примере будет возвращено число 2, поскольку в строке примера было найдено две гиперссылки. Затем возможно получить доступ к каждой совпавшей строке, используя индекс соответствующего ей элемента. Так, в данном примере url[0] – это первое совпадение, а url[l] – второе.

3.3. Задания на лабораторную работу № 3

Задание № 1. Разработка программы, работающей со строковыми данными.

Варианты задания №1

1. Создать программу, формирующую массив строк A[10] и осуществляющую с ним следующие действия:

– вывод массива на экран;

– вывод количества элементов массива, в которых встречается сочетание ‘abc’ (регистр не должен иметь значения);

– вывод на экран первых пяти символов 4-го элемента массива;

– вывод на экран последних трех символов элемента № 7.

2. Создать программу, формирующую массив строк B[8] и осуществляющую с ним следующие действия:

– вывод массива на экран;

– вывод количества элементов массива, в которых встречается сочетание ‘th’ (регистр не должен иметь значения);

– вывод на экран первых двух символов 6-го элемента массива;

– вывод на экран третьего, четвертого и пятого символов элемента № 2.

3. Создать программу, формирующую массив строк D[10] и осуществляющую с ним следующие действия:

– вывод массива на экран;

– вывод количества элементов массива, в которых встречается сочетание ‘fff’ (регистр не должен иметь значения);

– вывод на экран первых трех символов 1-го элемента массива;

– вывод на экран последних четырех символов элемента № 7.

4. Создать программу, формирующую массив строк R[11] и осуществляющую с ним следующие действия:

– вывод массива на экран;

– вывод количества элементов массива, в которых встречается сочетание ‘hello’ (регистр не должен иметь значения);

– вывод на экран первых двух символов 11-го элемента массива;

– вывод на экран последних трех символов элемента № 3.

5. Создать программу, формирующую массив строк Q[10] и осуществляющую с ним следующие действия:

– вывод массива на экран;

– вывод количества элементов массива, в которых встречается сочетание ‘text’ (регистр не должен иметь значения);

– вывод на экран первых четырех символов 6-го элемента массива;

– вывод на экран последних трех символов элемента № 3.

6. Создать программу, формирующую массив строк S[12] и осуществляющую с ним следующие действия:

– вывод массива на экран;

– вывод количества элементов массива, в которых встречается сочетание ‘en’ (регистр не должен иметь значения);

– вывод на экран первых двух символов 7-го элемента массива;

– вывод на экран символов № 2, 3, 4 второго элемента массива.

7. Создать программу, формирующую массив строк V[9] и осуществляющую с ним следующие действия:

– вывод массива на экран;

– вывод количества элементов массива, в которых встречается сочетание ‘c++’ (регистр не должен иметь значения);

– вывод на экран первых четырех символов 6-го элемента массива;

– вывод на экран последних трех символов элемента № 5.

8. Создать программу, формирующую массив строк Y[10] и осуществляющую с ним следующие действия:

– вывод массива на экран;

– вывод количества элементов массива, в которых встречается сочетание ‘ru’ (регистр не должен иметь значения);

– вывод на экран первых пяти символов 2-го элемента массива;

– вывод на экран последних двух символов элемента № 0.

9. Создать программу, формирующую массив строк Z[9] и осуществляющую с ним следующие действия:

– вывод массива на экран;

– вывод количества элементов массива, в которых встречается сочетание ‘xyz’ (регистр не должен иметь значения);

– вывод на экран первых двух символов 2-го элемента массива;

– вывод на экран последних шести символов элемента № 5.

10. Создать программу, формирующую массив строк M[11] и осуществляющую с ним следующие действия:

– вывод массива на экран;

– вывод количества элементов массива, в которых встречается сочетание ‘java’ (регистр не должен иметь значения);

– вывод на экран первых шести символов 2-го элемента массива;

– вывод на экран последних трех символов элемента № 0.

11. Создать программу, формирующую массив строк K[10] и осуществляющую с ним следующие действия:

– вывод массива на экран;

– вывод количества элементов массива, в которых встречается сочетание ‘xml’ (регистр не должен иметь значения);

– вывод на экран первых пяти символов 4-го элемента массива;

– вывод на экран последних трех символов элемента № 6.

12. Создать программу, формирующую массив строк W[14] и осуществляющую с ним следующие действия:

– вывод массива на экран;

– вывод количества элементов массива, в которых встречается сочетание ‘php’ (регистр не должен иметь значения);

– вывод на экран первых шести символов 3-го элемента массива;

– вывод на экран последних трех символов элемента № 8.

13. Создать программу, формирующую массив строк S[10] и осуществляющую с ним следующие действия:

– вывод массива на экран;

– вывод количества элементов массива, в которых встречается сочетание ‘txt’ (регистр не должен иметь значения);

– вывод на экран первых пяти символов 7-го элемента массива;

– вывод на экран последних двух символов элемента № 4.

14. Создать программу, формирующую массив строк T[11] и осуществляющую с ним следующие действия:

– вывод массива на экран;

– вывод количества элементов массива, в которых встречается сочетание ‘pdf’ (регистр не должен иметь значения);

– вывод на экран первых двух символов 2-го элемента массива;

– вывод на экран последних пяти символов элемента № 4.

15. Создать программу, формирующую массив строк R[10] и осуществляющую с ним следующие действия:

– вывод массива на экран;

– вывод количества элементов массива, в которых встречается сочетание ‘rus’ (регистр не должен иметь значения);

– вывод на экран первых трех символов 2-го элемента массива;

– вывод на экран последних двух символов элемента № 7.

16. Создать программу, формирующую массив строк M[12] и осуществляющую с ним следующие действия:

– вывод массива на экран;

– вывод количества элементов массива, в которых встречается сочетание ‘exe’ (регистр не должен иметь значения);

– вывод на экран первых четырех символов 7-го элемента массива;

– вывод на экран последних трех символов элемента № 2.

17. Создать программу, формирующую массив строк N[10] и осуществляющую с ним следующие действия:

– вывод массива на экран;

– вывод количества элементов массива, в которых встречается сочетание ‘jpg’ (регистр не должен иметь значения);

– вывод на экран первых шести символов 2-го элемента массива;

– вывод на экран последних двух символов элемента № 6.

18. Создать программу, формирующую массив строк K[12] и осуществляющую с ним следующие действия:

– вывод массива на экран;

– вывод количества элементов массива, в которых встречается сочетание ‘docx’ (регистр не должен иметь значения);

– вывод на экран первых шести символов 4-го элемента массива;

– вывод на экран последних трех символов элемента № 5.

Задание № 2. Составить программу, которая:

– формирует массив из 10 строк;

– выводит его на экран;

– выводит на экран те элементы, которые являются адресами веб-сайтов.

Задание № 3. Составить программу, которая формирует массив из 10 строк и выводит на экран все элементы, совпадающие с определенным регулярным выражением, вид которого зависит от выбранного варианта

1. Регулярное выражение – почтовый индекс Воронежа – число вида 394xxx – где x – любая цифра.

2. Регулярное выражение – телефон в городе Воронеже (473) 2xx-xx-xx – где x – любая цифра.

3. Регулярное выражение – IP-адрес – значение вида xxx.xxx.xxx.xxx – где x – любая цифра.

4. Регулярное выражение – мобильный номер – значение вида (+79xx) xxx-xx-xx – где x – любая цифра.

5. Регулярное выражение – номер автомобиля – значение вида bxxxbb – где x – любая цифра, b - любая буква.

6. Регулярное выражение – ИНН субъекта из Воронежского региона – значение вида 36xxxxxxxxxx – где x – любая цифра.

7. Регулярное выражение – СНИЛС – значение вида xxx-xxx-xxx xx – где x – любая цифра.

8. Регулярное выражение – ISBN русских изданий – значение вида 978-5-xxxxx-xxx-x – где x – любая цифра.

9. Регулярное выражение – номер паспорта – значение вида xxxx № xxxxxx – где x – любая цифра.

10. Регулярное выражение – время – выражение формата xx:xx:xx – где x – любая цифра.

11. Регулярное выражение – дата – выражение формата dd/mm/yyyy (все значения цифры).

12. Регулярное выражение – почтовый адрес – выражение формата - bbbb@bbb.bb (значения буквы или цифры).

13. Регулярное выражение – шифр группы – выражение формата xx-yyy (xx – любые буквы, yyy – любые цифры).

14. Регулярное выражение – номер свидетельства о госрегистрации – выражение формата – xx-yy xxxxxx (xx – любые буквы, yy – любые цифры).

15. Регулярное выражение – код типа 32/AX-789 – выражение формата – xx/yy-xxx (xx – любые цифры, yy – любые буквы).

16. Регулярное выражение – код типа A-17(6)/B – выражение формата – x-yy(y)/x (xx – любые буквы, yy – любые цифры).

17. Регулярное выражение – код типа 4593(23)/A+ – выражение формата – xxxx(xx)/y+ или xxxx(xx)/y- (xx – любые цифры, yy – любые буквы).

18. Регулярное выражение – код типа CFR-14/AZ(5) – выражение формата – xxx-yy/xx(y) (xx – любые буквы, yy – любые цифры).

4. ЛАБОРАТОРНАЯ РАБОТА № 4. ДИНАМИЧЕСКОЕ МОДИФИЦИРОВАНИЕ ВЕБ-СТРАНИЦ

4.1. Общие методические указания по выполнению лабораторной работы

Цели работы:

– освоить методы JavaScript для выборки элементов веб-страниц;

– ознакомиться с базовыми принципами динамического модифицирования содержимого элементов страницы.

Среда выполнения и отладки:

Текстовый редактор Notepad++, веб-браузер (Firebox, Internet Explorer, Opera или др.).

4.2. Теоретические сведения

Есть два основных метода доступа к узлам: *getElementById()* и *getElementsByTagName()*.

getElementById()

Получение элемента по ID означает нахождение отдельно взятого узла, имеющего уникальный идентификатор ID. Например, можно получить доступ к узлу с ID ‘header’ можно при помощи такого кода:

```
document.getElementById('header')
```

Часть document из document.getElementById (‘header’) – это ключевое слово, которое указывает на целый документ. Оно обязательно, то есть нельзя просто написать просто getElementById(). Команда **getElementById()** – это не метод объекта document, а часть «header» – это просто строка с именем искомого ID, переданная методу как аргумент.

Также можно передать методу переменную, содержащую строку с именем искомого ID:

```
var lookFor = 'header';
var foundNode = document.getElementById(lookFor);
```

Можно присвоить результаты работы этого метода переменной, чтобы сохранить ссылку на отдельный тег и позже иметь возможность управлять им в программе.

getElementsByTagName()

Принцип работы данного метода схож с **getElementById()**, но вместо ID, указывается имя искомого тега. Например, чтобы найти все гиперссылки на странице, следует написать следующее:

```
var pageLinks = document.getElementsByTagName('a');
```

Метод **getElementsByTagName()** возвращает список узлов, а не отдельно взятый узел. Данный список является массивом, так что можно получить доступ к отдельному узлу в нем, используя индекс. Также можно перебрать все элементы с помощью сочетания свойства length и цикла for.

Методы **getElementById()** и **getElementsByTagName()** можно также использовать вместе. Например, имеется вебстраница, содержащая тег , который имеет ID прикрепленного к нему баннера. Если требуется узнать, сколько ссылок находится в теге , можно использовать **getElementById()**, чтобы вернуть , а затем **getElementsByTagName()**, чтобы искать ссылки в . Вот как это работает:

```
var banner = document.getElementById('banner');
var bannerLinks = banner.getElementsByTagName('a');
var totalBannerLinks = bannerLinks.length;
```

Выборка соседних узлов

Тег, находящийся внутри другого, называется дочерним. Теги, содержащие другие теги, называются родительскими

Объектная модель документов может дать доступ к «родительскому», «дочернему» или «братскому» узлу. Узел, содержащий другой узел, является «родителем». Узлы, у которых один и тот же «родитель», как, например, два текстовых узла Some и strong, называются «братьями».

Существует несколько способов доступа к ближним узлам.

1. .childNodes – свойство узла, содержащее список всех узлов, являющихся дочерними по отношению к данному. Данный список аналогичен массиву, возвращаемому методом getElementsByTagName. Предположим, что в HTML-файл добавляется следующий код JavaScript:

```
var headline = document.getElementById('header');
var headlineKids = headline.childNodes;
```

Переменная headlineKids будет содержать список тегов, являющихся дочерними по отношению к тегу, имеющему ID «header».

2. .parentNode – свойство, ссылающееся на родительский узел выбранного узла. Например, если требуется знать, какой тег содержит элемент с ID ‘header’, то следует написать следующий код:

```
var headline = document.getElementById('header');
var headlineParent = headline.parentNode;
```

3. .nextSibling и .previousSibling – свойства, указывающие на узел, идущий сразу за данным узлом (nextSibling), либо на узел, предшествующий данному (previousSibling):

```
var headline = document.getElementById('header');
var headlineSibling = headline.nextSibling;
```

Переменная headlineSibling – это ссылка на тег, следующий за тегом <h1>. Если осуществляется попытка получить доступ к несуществующему братскому узлу, JavaScript возвращает значение null. Например, можно проверить, есть ли у узла братский узел previousSibling, следующим образом:

```
var headline = document.getElementById('header');
var headlineSibling = headline.prevSibling;
if (! headlineSibling)  {
alert ('Данному узлу другие узлы не предшествуют!');
}  else  {
        // некие действия с предшествующим узлом
}
```

Дополнение. При выборке узлов надо иметь в виду, что в нее включаются не только теги, но и другие элементы кода HTML (атрибуты, комментарии и т.д.). Поэтому следует проверять, к какому типу относится элемент выборки с помощью свойства nodeType. Например:

```
var headline = document.getElementById('header');
var headlineKids = headline.childNodes;
for (var i=0; i<headlineKids.length; i++)
if (headlineKids[i].nodeType==1)
  alert(headlineKids[i]);
```

Если элемент является тегом, то его свойство nodeType равно 1, если атрибутом – то 2 и т.д. (до 12).

Добавление содержимого на страницу

Программы JavaScript часто должны добавлять содержимое на страницу, удалять или изменять его.

Добавление содержимого с использованием объектной модели документа – очень нудная работа. Она заключается в создании каждого узла содержимого и в помещении результата на страницу. Другими словами, если требуется добавить тег , а также еще пару тегов и текст, то следует отдельно создать каждый узел и правильно разместить его по отношении к остальным. Однако, существует более простой метод – **innerHTML**.

Свойство **innerHTML** представляет собой весь HTML, находящийся в узле. Например, в HTML-коде находится тег <p>. Итак, **innerHTML** для этого тега <p> таков: Some important text. Для доступа к этому HTML используется следующий код JavaScript:

```
//получить список всех тегов <p> на странице
var pTags = document.getElementsByTagName('p');
//получить первый тег <p> на странице
var theP = pTags[0];
alert(theP.innerHTML);
```

В данном случае переменная theP ссылается на узел для первого параграфа страницы. Последняя строка кода открывает окно предупреждения и отображает весь код, находящийся внутри этого тега. Например, добавление JavaScript в HTML в исходном примере заставит появляться в окне предупреждения текст Some important text.

Используя **innerHTML**, можно не только узнать, что находится внутри узла. Переопределив данное свойство, можно изменить содержимое узла:

```
var headLine = document. getElementById ('header');
headLine.innerHTML =  'Этот текст был добавлен с помощью JavaScript';
```

В данном примере содержимое тега с ID «header» меняется на 'Этот текст был добавлен с помощью JavaScript'. innerHTML не ограничен текстом: можно переопределять свойство innerHTML для дополнения HTML, включая теги и их атрибуты.

4.3. Задания на лабораторную работу № 4

Задание № 1. Разработка программы, осуществляющей выборку элементов страницы и действия с ними.

Для создания работы следует использовать файл example1.html с уже готовым HTML-кодом. Требуется добавить в файл сценарий JavaScript, осуществляющий следующие действия:

– вывод на экран количества элементов в списке, помеченном ID ‘countryList’;

– вывод на экран суммарного количества населения в странах на основе списка, помеченного ID ‘population’;

– сортировка обоих списков по убыванию населения и вывод на экран результата в виде списка пар «страна - численность населения».

Задание № 2. Разработка программы, осуществляющей выборку элементов страницы и действия с ними.

Для создания работы следует использовать файл example2.html с уже готовым HTML-кодом. Требуется добавить в файл сценарий JavaScript, осуществляющий следующие действия:

– вывести на экран содержимое тега, следующего после тега с ID “header”;

– вывести на экран количество элементов в списке, помеченном ID “list”;

– вывести на экран содержимое тега, следующего перед тегом с ID “razdel2”;

– с помощью свойства innerHTML поменять содержимое заголовков второго уровня так, чтобы они были выделены курсивом.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Сергеев, М. Ю. Web-дизайн: создание Web-сайтов с помощью HTML и CSS [Текст]: учеб. пособие / М.Ю. Сергеев. – Воронеж: ГОУВПО «ВГТУ», 2012. – 219 с.
2. Справочник по HTML и CSS [Электронный ресурс]: Режим доступа: World Wide Web. URL: http://htmlbook.ru/.
3. Макфарланд, Д. JavaScript. Подробное руководство [Текст] / Д. Макфарланд. – М.: Эксмо, 2009. – 608 с.
4. Флэнаган, Д. JavaScript. Подробное руководство [Текст] / Д. Флэнаган. – Спб: Символ-Плюс, 2014. – 1080 с.

СОДЕРЖАНИЕ

ОСНОВЫ ВЕБ-ПРОГРАММИРОВАНИЯ НА JAVASCRIPT

МЕТОДИЧЕСКИЕ УКАЗАНИЯ
к выполнению лабораторных работ № 1-4
по дисциплине «Проектирование и разработка
Web-приложений»
для студентов направления
09.03.01 «Информатика и вычислительная техника»
профиля «Вычислительные машины, комплексы, системы
и сети» очной формы обучения

Составитель
Сергеев Михаил Юрьевич

В авторской редакции

Подписано к изданию 26.11.2015
Уч.-изд. л. 3,1

ФГБОУ ВО «Воронежский государственный
технический университет»
394026 Воронеж, Московский просп., 14