Министерство сельского хозяйства Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
Иркутский государственный аграрный университет имени А.А. Ежевского
Институт экономики, управления и прикладной информатики
Кафедра информатики и математического моделирования

Асалханов П.Г., Бендик Н.В.

Web-программирование: JavaScript

Учебное пособие

Иркутск 2020

УДК 004.43

Печатается по решению научно-методического совета ФГБОУ ВО Иркутского ГАУ (протокол № 11 от 26 ноября 2020 г.).

Рецензенты:

Бузина Т.С. – к.т.н., доцент кафедры информатики и математического моделирования Иркутского ГАУ имени А.А. Ежевского.

Соколов Д.В. - к.т.н., старший научный сотрудник одела трубопроводных систем энергетики №50 ИСЭМ СО РАН

Асалханов П.Г. Web-программирование: JavaScript: Учебное пособие / П.Г. Асалханов, Н.В. Бендик, – Иркутск: Изд-во Иркутский ГАУ, 2020. – 121 с. – ил.

Язык JavaScript до сих пор остается самым популярным языком разработки сценариев для веб-браузера. В данном пособии изучены теоретические основы языка JavaScript. Описаны его управляющие конструкции. Рассмотрены основные процедуры и функции. Изучены объектные модели браузера и документа (DOM). Учебное пособие содержит практические задания для выполнения, а также подробно разобранные примеры и справочный материал.

Учебное пособие предназначено для изучения дисциплин «Интернет-программирование» для студентов 4-го курса направления подготовки 09.03.03 Прикладная информатика, «Web-программирование» для студентов 2-го курса направления 09.04.03 Прикладная информатика.

Содержание

1 ОСНОВЫ ЯЗЫКА JAVASCRIPT 5

1.1 Общие сведения о JavaScript 5
1.2 Базовые события 6
1.3 Переменные и значения 8
1.4 Функции преобразования 11
1.5 Константы 13
1.6 Шаблонные литералы 13
1.7 Область видимости переменной 14

2 УПРАВЛЯЮЩИЕ КОНСТРУКЦИИ 16

2.1 Цикл 16
2.2 Условный переход 18
2.3 Множественный переход 18

3 ПРОЦЕДУРЫ И ФУНКЦИИ 21

3.1 Анонимные функции 25
3.2 Анонимная функция как обработчик события 27
3.3 Стрелочные функции 28
3.4 Функции обратного вызова 29
3.5 Функция как объект 29
3.6 Математические функции. Объект math 30
3.7 Динамическое изменение списка 34
Практическая работа №1 37

4 РАБОТА С МАССИВАМИ 40

4.1 Основные свойства и методы объекта Array 41
4.2 Перебирающие методы объекта Array 43
4.3 Оператор расширения 46
Практическая работа №2 48

5 СТРОКИ И РЕГУЛЯРНЫЕ ВЫРАЖЕНИЯ 50

5.1 Строки. Объект String 50
5.2 Регулярные выражения 53
5.3 Дата и время. Объект Date 57
Практическая работа №3 60
Практическая работа №4 62

6 ОБЪЕКТНАЯ МОДЕЛЬ БРАУЗЕРА 63

6.1 Объект Window 65
6.2 Объект Location 71
6.3 Объект History 72
6.4 Объект Screen 73
6.5 Объект Document 73
6.5 Объект Image 77
6.6 Объект Navigator 78
6.7 Кукиз (cookies) 82
6.8 Классы 85
Практическая работа №5 89
Практическая работа №6 91

7 DOCUMENT OBJECT MODEL (DOM).....93
7.1 Общие сведения о DOM 93
7.2 Узлы 95
7.3 Атрибуты и свойства.....102
7.4 Текстовый узел.....107
7.5 Операции с узлами 111
Практическая работа №7.....116
ЛИТЕРАТУРА 120

1 Основы языка JavaScript

1.1 Общие сведения о JavaScript

Наиболее известными скриптовыми языками на сегодняшний день являются JavaScript, JScript (аналог языка JavaScript от Microsoft), VBScript (Visual Basic Script от Microsoft) и ActionScript (от компании Macromedia). Скриптовый язык — это объектно-ориентированный язык программирования, который добавляет интерактивность, обработку данных, управление браузером и многое другое в содержимое разрабатываемых веб-страниц или flash-приложений (ActionScript).

Скриптовый язык не содержит всех возможностей обычных языков программирования, таких, например, как работа с файлами или управление графикой. Созданные с помощью скриптовых языков программы работают в браузерах, поддерживающих их выполнение. Создаваемые на скриптовых языках программы, называемые сценариями или скриптами, включаются в состав веб- страниц и распознаются и обрабатываются браузером отдельно от остального html-кода. Браузер, встречая ошибки в скриптах, выдает диалоговое сообщение об этом или указывает об ошибке в своей статусной строке.

ActionScript может компилироваться в код для хранения в SWF-файле. SWF-файлы исполняются программой Flash Player, которая существует в виде плагина к веб-браузеру.

Язык JavaScript разработан в 1995 г. фирмой Netscape в сотрудничестве с Sun Microsystems на базе языка Sun’s Java. Название Java было дано языку в честь любимой разработчиками марки кофе. По инициативе компании Netscape была проведена стандартизация языка ассоциацией ECMA. Стандартизированная версия имеет название ECMAScript (сокращенно ES). Первой версии спецификации ES1, появившейся в 1997 г., соответствовал JavaScript версии 1.1. Всего существует 8 версий ES. Режим **strict** (строгий режим), введенный в ECMAScript 5, позволяет использовать более строгий вариант JavaScript.

Операторы JavaScript размещаются в контейнере <SCRIPT> и разделяются символом ”;” в рамках одной строки. Для браузеров, не поддерживающих скрипты, операторы заключают еще в теги комментариев языка HTML, чтобы они не были видны посетителю при просмотре веб-страницы. Контейнер, содержащий скрипт, можно добавлять в любой раздел html- документа.

Часто вызываемые функции лучше добавлять в заголовочный раздел, так как браузер будет выполнять их быстрее (поиск функций браузером осуществляется сверху вниз).

```
<script type=”text/javascript”>
<!--
document.write ("Это JavaScript!")
// -->
</script>
```

Приведенный скрипт выводит на веб-страницу текстовую строку «Это JavaScript!».

Комментарии JavaScript отличаются от комментариев языка HTML.

```
// комментарии на одной строке
/*
комментарии на нескольких строчках
*/
```

Комментарии могут быть использованы для отладки сценариев. Кроме комментариев можно использовать метод log объекта Console, который выводит в консоль браузера значение α.

$$console.log(\alpha)$$

где α — выражение или переменная.

Язык чувствителен к регистру при задании значений параметров!

1.2 Базовые события

Веб-страница, содержащая скрипт, позволяет обрабатывать события, связанные с окном браузера, — такие как загрузка документа, закрытие окна, появление курсора над объектом страницы, нажатие клавиши мыши или клавиатуры и др.

Скрипт может по-разному реагировать на эти события. Скриптовые программы иногда еще называют сценариями просмотра веб-страницы.

Базовые события JavaScript	
onBlur	элемент теряет фокус
onChange	изменение значения текстового поля
onFocus	элемент получает фокус
onCopy	копирование в буфер обмена
onClick	щелчок мышкой в области элемента
onMouseOver	перемещение мышиного курсора на область элемента
onMouseOut	перемещение мышиного курсора за область элемента
onMouseMove	перемещение мышиного курсора в области элемента

onMouseDown	нажатие кнопки мыши
onMouseUp	отпускание кнопки мыши
onReset	нажатие кнопки типа RESET
onSubmit	нажатие кнопки типа SUBMIT
onLoad	завершение загрузки страницы или графического изображения
onUnload	переход на другую страницу или завершение работы браузера
onTransitionEnd	окончание анимационного перехода (для css-свойства transition)

События, как и атрибуты, связываются с тегами языка HTML и не заключаются в контейнер <SCRIPT>.

```
<IMG SRC = “smile.gif” onMouseOver = ”смени_изо ()” onMouseOut = “верни_изо ()”>
<SELECT SIZE=“7” onClick="ch_pict ()">
<BODY onLoad="ss (); clock ()">
<FORM onSubmit=”return проверка_данных ()”>
```

Из примеров видно, что каждому событию сопоставляется вызов функции, код которой должен быть включен в скрипт на языке JavaScript. Вместо имени функции можно написать небольшой фрагмент кода.

```
<INPUT TYPE = "button" VALUE = "Щёлкни по мне" onClick = ’alert ("Ку-ку")’>
```

Команда *alert* (α) выводит диалоговое окно с сообщением. В качестве аргумента можно указывать имена переменных или выражения. Тогда в окне будет размещено значение переменной или выражения.

Операторы JavaScript могут также размещаться в качестве значения параметра *href* тега гиперссылки.

```
<A HREF="javascript: window.alert('Do you speak English?')"> "Don`t click here" </A>
```

Если нам необходимо выполнить некоторые действия при выборе гипертекстовой ссылки, но при этом не перегружать текущие страницы, то в параметре HREF можно указать конструкцию:

```
<A HREF="javascript:void(0)"> kuku </A>
```

Код JavaScript может быть еще размещен и во внешнем файле (расширением js или jsc). При загрузке веб-страницы этот код докачивается программой просмотра и исполняется так же, как если бы он размещался в самом html- документе. При просмотре текста документа через опцию «Источник» текст скрипта не отображается. В файле, который содержит конструкции

JavaScript, HTML-теги не используются.

```
<SCRIPT TYPE=”text/javascript” SRC=”timer.jsc”> </SCRIPT>
```

Для написания скриптов, управляющих содержимым веб-страницы, необходимо представлять себе иерархию объектов HTML-документа. Управление содержимым веб-страницы после ее загрузки на компьютер клиента лежит в основе технологии *Dynamic HTML*. JavaScript вместе с каскадными таблицами стилей (CSS) составляют фундамент этой технологии.

1.3 Переменные и значения

Переменной в языке программирования называют такое слово, которое обладает каким-либо значением, и это значение можно изменить в любой момент времени, а предыдущее значение будет безвозвратно потеряно. Значением переменной в JavaScript может быть либо число, либо логическое значение (true/false), либо строка, либо массив. Присвоить (изменить) значение переменной можно разными способами, о которых будет сказано.

В JavaScript переменные можно объявлять с помощью служебного слова **var** (или **let**), после которого следует имя переменной или нескольких переменных, разделенных запятой. Имена переменных, как и все другие элементы языка, чувствительны к регистру.

var α [{,β}]

```
var x, y, z
var угол, радиус, длина_окружности
```

Переменным могут быть даны следующие типы значений.

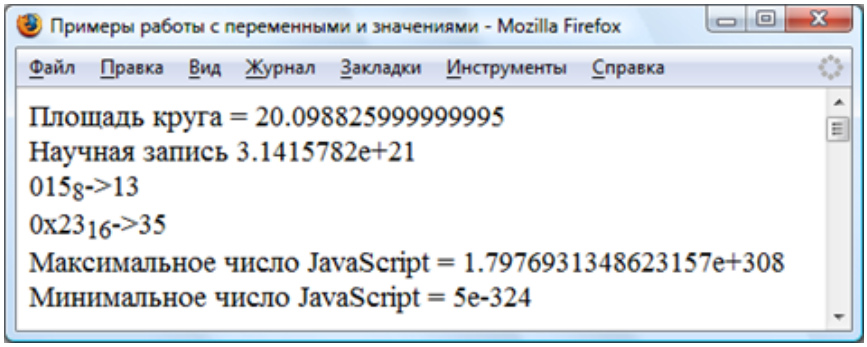
1. Числовой (целые числа и с плавающей запятой).

```
var пи = 3.14
var радиус = 5.06 / 2
document.write ("Площадь круга = " + пи*радиус*радиус + "<br>")
//научная запись
нч = 3.1415782e + 21
document.write (“Научная запись”, нч, "<br>")
// восьмеричные (первая цифра ноль)
var вч = 015
document.write ("015", "<SUB>8</SUB>", "->", вч, "<br>")
//16-ричные (первая цифра ноль)
```

```
var шч = 0x23
document.write ("0x23", "<SUB>16</SUB>", "->", шч, "<br>")
```

Свойства *MAX_VALUE* и *MIN_VALUE* объекта *Number* возвращают информацию о максимальном и минимальном числах, с которыми может оперировать JavaScript.

```
document.write ('Максимальное число JavaScript = ', Number.MAX_VALUE, "<BR>")
document.write ('Минимальное число JavaScript = ', Number.MIN_VALUE, "<BR>")
```



Над числами определены следующие операции: умножение (*), деление (/), остаток от деления (%), сложение (+), вычитание (–). Например:

```
5*x-7%2+x/5
```

2. Логический (true/false).

```
var flag = false
```

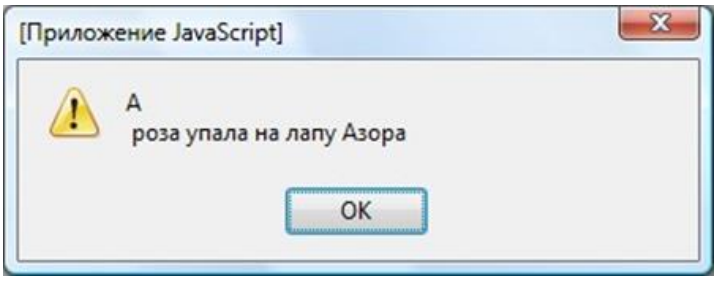
3. Строковый.

Строка — последовательность символов алфавита языка программирования, заключенная в кавычки или апострофы. Если в кавычки или апострофы не заключить ни одного символа, будем иметь **пустую строку**. Каждый символ имеет порядковый номер. Первый символ строки имеет номер 0. В строку можно добавлять управляющие последовательности, такие как, например:

\n переход на следующую строку
\r Enter
\t Tab
\\ Обратная косая черта

```
var строка="А роза упала на лапу Азора"
```

```
document.write (строка,"<br>")
var строка2='A \r\n роза упала на лапу Азора'
alert (строка2)
```



4. Неопределенное значение (*undefined*).

Бывают случаи при разработке и отладке сценария, когда JavaScript выдает сообщение *undefined*. Это означает, что запрашиваемое значение не определено или не задано.

5. Бесконечность (*Infinity*/*-Infinity*).

Иногда JavaScript в качестве значения математической функции или выражения возвращает слово *Infinity*, что означат бесконечность. Например:

```
alert (Math.log(0)); alert (-1/0)
```

6. *NaN* (Not a Number — не число).

Если в качестве значения выражения JavaScript выдает *NaN*, это означает, что в результате вычисления математического выражения не получается числового значения. Например:

```
alert (Math.sqrt (-1))
```

Функция *isNaN* может проверить, не является ли проверяемое значение числом.

```
var x=prompt ("Введите какое-либо значение")
if (isNaN (x)) alert ("Вы ввели нечисловое значение")
```

7. Функция.

В JavaScript имя функции может быть присвоено в качестве значения какой-либо переменной.

```
function случайное_число () {x = Math.random () }
R = случайное_число
```

```
document.write (R)
```

Последняя команда в примере выведет на страницу всю конструкцию функции.

Для определения (проверки во время отладки) типа переменной существует функция typeof.

$$typeof(\alpha)$$

где α — имя переменной любого типа.

1.4 Функции преобразования

Для преобразования (превращения) строки в целое число предусмотрена функция *parseInt*.

$$parseInt\ (\alpha\ [,\beta])$$

где α — строка или элемент, возвращающий строку; β — число, задающее основание числа, в которое преобразуется строка α (по умолчанию 10).

```
x="строка"
document.write ('преобразованное в число значение ', '<B>', x, '</B>', ' = ' , parseInt(x), "<BR>")
x="123строка"
document.write ('преобразованное в число значение ', '<B>', x, '</B>', ' = ', parseInt(x), "<BR>")
x="строка123"
document.write ('преобразованное в число значение ', '<B>', x, '</B>' ,' = ', parseInt(x), "<BR>")
x="54.45"
document.write ('преобразованное в число значение ', '<B>', x, '</B>', ' = ', parseInt(x), "<BR>")
x="-7654321"
document.write ('преобразованное в число значение ', '<B>', x, '</B>', ' = ', parseInt(x), "<BR>")
x="0x23"
document.write ('преобразованное в число значение ', '<B>', x, '</B>', ' = ', parseInt (x,16),
"<BR>")
x="015"
document.write('преобразованное в число значение ', '<B>', x, '</B>', ' = ', parseInt(x,8),
"<BR>")
```

x	parseInt(x)
"строка"	NaN
"123строка"	123
"строка123"	NaN

"54.45"	54
"-7654321"	-7654321
"0x23"	35
"015"	13

Функция *parseFloat* возвращает число с плавающей точкой из преобразованной строки.

parseFloat (α)

где α — строка или элемент, возвращающий строку.

```
x="54.45"
document.write ('преобразованное в число с плавающей точкой строковое значение ', '<B>', x, '</B>', ' = ', parseFloat (x), "<BR>")
```

x	parseFloat (x)
x="54.45"	54.45

Метод *toString* преобразовывает объект α в строку.

α.toString ([β])

где β — основание для представления числовых значений.

```
x=1234567; document.write (x.toString().length)
```

Функция *eval* преобразует строку в исполняемый код JavaScript.

eval (α)

```
eval (“alert ('Helo')”)
document.write (eval (‘Number.MAX_VALUE’))
```

Функция *toFixed* возвращает округленное до β знаков число α.

α.toFixed (β)

где α, β — число или элемент, возвращающий число.

Функция *toExponential* возвращает преобразованное в экспоненциальную

форму число α с β знаками после запятой.

α.toExponential (β)

где α, β — число или элемент, возвращающий число.

Функция *toPrecision* возвращает преобразованное в экспоненциальную форму число α с фиксированной точкой (запятой) и с β значащими цифрами.

α.toPrecision (β)

где α, β — число или элемент, возвращающий число.

1.5 Константы

Объявление **const** делает из переменной величины константу, т. е. переменную, значение которой нельзя изменить. Объявление константы должно всегда происходить вместе с заданием ей значения.

```
const NAME = ”Василий”
const V = 150
```

Обычно имена констант пишут прописными буквами, если они используется в пределах всего html-документа, в остальных случаях имена содержат только строчные символы.

1.6 Шаблонные литералы

Шаблонными литералами называются строки, допускающие использование выражений внутри. С ними можно использовать многострочные литералы и строковую интерполяцию. В спецификациях до ES2015 они назывались «шаблонными строками».

Шаблонные литералы заключаются в обратные кавычки ` ` вместо двойных или одинарных. Они могут содержать **подстановки**, обозначаемые знаком доллара и фигурными скобками.

${выражение}

Шаблонные литералы с подстановками удобно использовать в операторах вывода информации. Например:

```
a=3; b=7; alert (`Сумма ${a} + ${b} равна ${a+b}`) // Сумма 3 + 7 равна 10
```

Расширенной формой шаблонных литералов являются **теговые шаблоны**. Они позволяют разбирать шаблонные литералы с помощью функции. Первый аргумент такой функции содержит массив строковых значений, а остальные содержат выражения из подстановок. В итоге функция должна вернуть собранную строку.

```
var человек = 'Степан'
var возраст = 2
function пример (строка, человекП, возрастП) {
   var str0 = строка[0] // "Этот"
   var str1 = строка[1] // "является"
   if (возрастП > 60){
      возрастП = 'пожилым'
  } else {
      возрастП = 'молодым'
  }
  return `${str0}${человекП}${str1}${возрастП}`
}
var output = пример `Этот ${человек} является ${возраст}`
alert (output); // Этот Степан является молодым
```

Существует метод **String.raw()**, возвращающий такую же исходную строку, какую вернула бы функция шаблона по умолчанию и строковая конкатенация вместе.

```
var str = String.raw`Сумма 2 + 3 \n${2+3}`;
// "Сумма 2 + 3 \n5"
```

1.7 Область видимости переменной

Оператор var применяется для задания переменной с глобальной областью видимости в пределах всего html-документа либо внутри функции (если переменная объявлена в теле функции).

Оператор let применяется для задания переменной с блочной областью видимости. Блок ограничивается фигурными скобками.

```
{
let a=2 a=4
}
console.log(a) // undefined
```

За пределами блока локальная переменная будет иметь неопределенное значение *undefined*.

2 Управляющие конструкции

2.1 Цикл

JavaScript поддерживает все основные виды циклов: цикл пока (while), цикл до (do-while), цикл с параметром (for). Цикл пока (while) выполняет операторы, находящиеся в теле до тех пор, пока условие цикла сохраняет значение истина (true).

$$while\ (\alpha)\ \{\ \beta\ \}$$

где α — условие, являющееся логическим выражением; пока его значение истинно, цикл выполняется; β — тело цикла, включающее в себя последовательность операторов JavaScript.

Пример

```
var s = 0
while (s < 100)         {
    document.write (s + "<br>")
    s += 10
}
```

Цикл do-while очень похож на цикл while. Единственное отличие состоит в том, что в цикле do-while блок кода (тело цикла) выполняется до проверки условия. Поэтому даже если условие становится неверным (false), то тело цикла все равно будет выполнено один раз до этой проверки. Можно смело утверждать, что в данном цикле блок кода будет выполняться всегда как минимум один раз.

$$do\ \{\ \beta\ \}\ while\ (\alpha)$$

Пример

```
do {
document.write(i + "<br>");
   i = i + 2;
} while (i < 20)
```

Цикл с параметром выполняется до тех пор, пока условие имеет значение истина (true). Параметр цикла имеет приращение, поэтому при каждом проходе цикла значение параметра меняется. Таким образом, цикл for удобен при использовании в блоке кода параметра, например, когда необходимо

проанализировать каждый символ строки или каждый элемент массива, или когда нужно накопить сумму или произведение. Параметр может выступать в качестве номера символа строки или номера элемента массива, или очередного слагаемого или множителя.

for (α; β; γ) { λ }

где α — начальное значение параметра цикла (если параметров несколько, то они разделяются запятой); β — условие (может включать параметры цикла); пока его значение истинно, цикл выполняется; γ— приращение параметра (если параметров несколько, то они и приращения разделяются запятой); λ — тело цикла, содержащее команды JavaScript, разделенные символом ";".

Любой из параметров может быть пустым. Если тело цикла содержит только одну команду, то фигурные скобки можно опустить.

Пример

```
for (i = 0; i <= 7; i = i + 1)
    document.write ("<H3>" + i + "</H3>")
for (i = 1, j = 10; i < j; i++, j--)
    document.write ("<p>", i, "</p>")
for (i = 1, j = 10; i < j; ) {
    document.write (i)
    i++
    j--
    }
```

Метод *write* объекта *document* выводит на веб-страницу информацию, указанную в качестве аргумента в круглых скобках. Для форматированного вывода можно использовать теги HTML.

Существует вариант цикла for для перебора всех свойств объекта JavaScript.

for (var α in β) { λ }

где α — имя переменной; β — имя объекта JavaScript; λ — тело цикла, содержащее команды JavaScript, разделенные символом ";".

Пример

```
<script type="text/javascript">
   for (var свойство in location)
      document.write ("<B>" + свойство + ": </B>" + eval ("location."+ свойство) + "<br>")
```

```
</script>
```

Оператор *break* позволяет досрочно прекратить выполнение цикла. Это бывает необходимо, например, при выполнении некоторого условия, когда дальнейшая проверка условия становится бесполезной или даже вредной.

2.2 Условный переход

if (α)
    { β }
[else
    { γ }]

где α — условие; если его значение **true** (истина), то выполняется последовательность команд β; если его значение **false** (ложь), то выполняется последовательность команд γ; блок else может быть опущен.

Для составления условных выражений в словаре JavaScript предусмотрены следующие знаки логических операций и функций

==	равно
===	идентично
>	больше
<	меньше
> =	больше или равно
< =	меньше или равно
&&	и
||	или
!	не

Оператор == сравнивает аргументы на равенство, а оператор === сравнивает на идентичность. Оператор === не приводит аргументы к одному типу.

Примеры

```
var xyz=0
alert (xyz==false) // true alert (xyz===false) // false
<input type = "button" value = "Click me" onClick = "if (document.form1.tf.value < 1) {
document.write ('hello!'); }">
```

2.3 Множественный переход

switch (α)
{ case β_1 :

```
        γ1
        break
    case  β2 :
        γ2
        break
 …
   default:  γd
 }
```

где α — переменная или выражение с переменной; β_1, β_2, … — возможные значения α; γ_1, γ_2, …, γ_d — команды JavaScript.

Оператор *switch* заменяет несколько операторов *if*. При каждом предусмотренном значении α будет вызываться на выполение последовательность команд γ. Оператор *default* является необязательным. Он срабатывает тогда, когда α принимает непредусмотренное значение.

Пример. Моделирование бросания игральной кости

```
<script type="text/javascript">
function случайное_число () {
  var x = Math.random () * 5 + 1
  return Math.round (x)
}
var total1 = 0, total2 = 0, total3 = 0, total4 = 0, total5 = 0, total6 = 0
for (var i = 1; i <= 1000; i++) {
   switch (случайное_число()) {
      case 1 :
        total1++
        break
      case 2 :
        total2++
        break
      case 3 :
        total3++
        break
       case 4 :
        total4++
        break
       case 5 :
         total5++
         break
       case 6 :
         total6++
   }
}
```

```
document.write("<table>")
document.writeln("<tr><th>грань</th><th>число выпадений</th>")
document.writeln("<tr><td>1</td><td>" + total1 + "</td>")
document.writeln("<tr><td>2</td><td>" + total2 + "</td>")
document.writeln("<tr><td>3</td><td>" + total3 + "</td>")
document.writeln("<tr><td>4</td><td>" + total4 + "</td>")
document.writeln("<tr><td>5</td><td>" + total5 + "</td>")
document.writeln("<tr><td>6</td><td>" + total6 + "</td>")
document.write("</table>")
</script>
```

грань	число выпадений
1	100
2	179
3	212
4	204
5	221
6	84

3 Процедуры и функции

Многие структурные языки программирования поддерживают разработку процедур и функций для введения в язык новых команд и функций соответственно. С точки зрения языка команда — это указание компьютеру совершить какое-либо действие, функция — это слово, возвращающее какое-либо значение, явно или неявно зависящее от значения аргумента. Функции могут выступать в качестве параметров команд или могут быть элементом выражения.

Язык JavaScript также поддерживает внедрение команд и функций, но разработчики языка сэкономили на инструментах разработки. И в том, и в другом случае разработка оформляется как функция, но лишь оператор *return*, включенный в тело функции, делает ее таковой во истину.

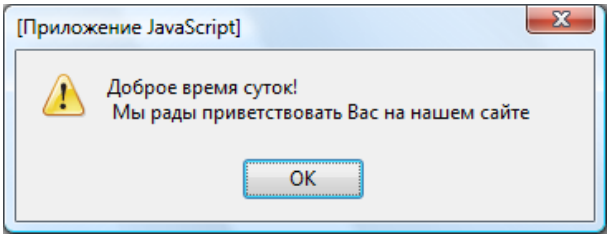
$$function\ \alpha\ ([\beta,\gamma...])\ \{\ \delta\ \}$$

где α — имя функции, которое должно состоять только из одного слова; $\beta,\gamma...$ — необязательные входные аргументы (параметры) функции; δ — тело функции, состоящее из последовательности операторов (команд) JavaScript.

Вызвать функцию на выполнение можно написанием ее имени вместе с круглыми скобками и значениями аргументов либо непосредственно в скрипте, либо в теле другой функции, либо как обработчик события.

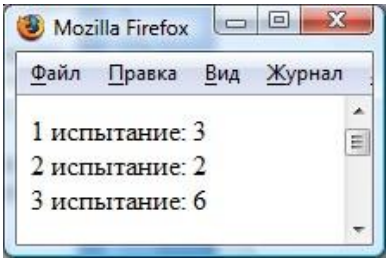
Пример 1. Функция создает оператор *приветствие*. Вызов функции осуществляется в скрипте.

```
<script type=”text/javascript”>
function приветствие () {
   строка='Доброе время суток! \r\n '
   строка += 'Мы рады приветствовать Вас на нашем сайте'
   alert (строка)
}
приветствие ()
</script>
```



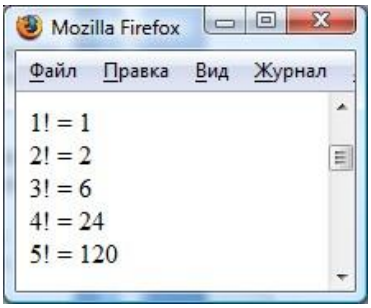
Пример 2. Функция создает слово-функцию *случайное_целое*, возвращающую случайное целое число в диапазоне от 1 до 5. Оператор *return* α присваивает имени функции значение своего параметра α.

```
<script type=”text/javascript”>
function случайное_целое () {
   x = Math.random () * 5 + 1
   return Math.round (x)
}
document.write (“1 испытание: ”, случайное_целое(), '<br>')
document.write (“2 испытание: ”, случайное_целое(), '<br>')
document.write (“3 испытание: ”, случайное_целое(), '<br>')
</script>
```



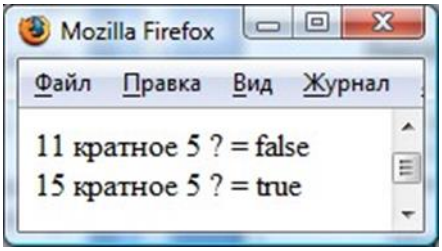
Пример 3. Функция с параметром.

```
function факториал (n) {
  факт = 1
  for (i = 1; i <= n; i++) факт = факт * i
  return факт
}
document.write ("1! = ",факториал(1),"<br>")
document.write ("2! = ",факториал(2),"<br>")
document.write ("3! = ",факториал(3),"<br>")
document.write ("4! = ",факториал(4),"<br>")
document.write ("5! = ",факториал(5),"<br>")
```



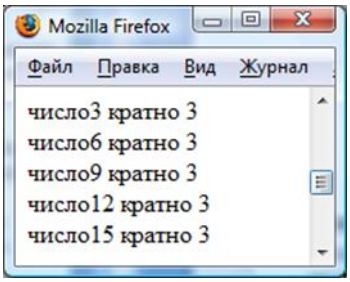
Пример 4. Функция с двумя параметрами.

```
function кратное (число, делитель) {
   if (число % делитель == 0)
      return true
   else
      return false
}
document.write ("11 кратное 5 ? = ",кратное(11,5),"<br>")
document.write ("15 кратное 5 ? = ",кратное(15,5),"<br>")
```



Пример 5. Вызов функции из другой функции.

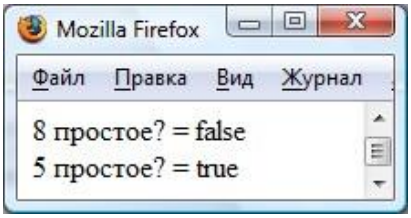
```
function все_кратные_3 (начало_отрезка, конец_отрезка) {
    for (i = начало_отрезка; i <= конец_отрезка; i++) {
       if (кратное (i,3))
          document.write ("число",i," кратно 3 <br>")        }
}
все_кратные_3 (1,15)
```



Пример 6. Досрочное прекращение работы функции (досрочный выход из цикла — break).

```
function простое (число) {
   for (i=2;i<число;i++) {
      if (число%i==0) {return false; break}
     }
   return true
}
document.write ("8 простое? = ",простое(8),"<br>")
```

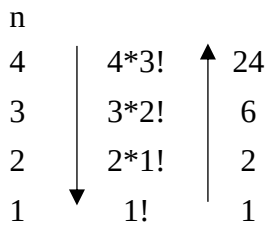
```
document.write ("5 простое? = ",простое(5),"<br>")
```

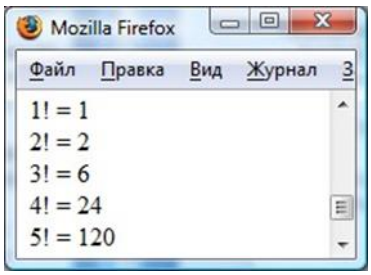


Пример 7. Рекурсивная функция.

```
function факториал_рек (n) {
   if (n==1) return 1
   return n*факториал_рек (n-1)
}
for (i=1; i<=5; i++) { document.write (i, "! = ", факториал_рек (i), "<br>") }
```

Таблица значений стека



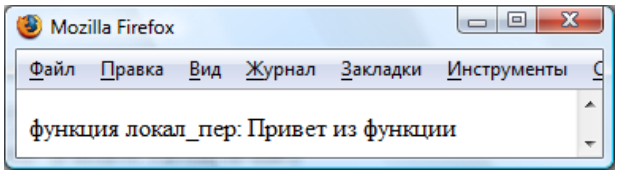


Пример 8. Локальная переменная (область видимости).

В спецификации ES-2015 предусмотрены способы объявления переменных через let и const помимо var. Область видимости переменной let — это блок, заключенный в фигурные скобки {...}. Однако в функции, как и раньше, можно объявлять переменную через var, и ее значение будет доступно только в пределах данной функции.

```
function локал_пер () {
  var лп='Привет из функции'
  document.write ("функция локал_пер: ", лп, "<br>")
}
локал_пер ()
```

```
document.write ("скрипт: ", лп, "<br>")
```

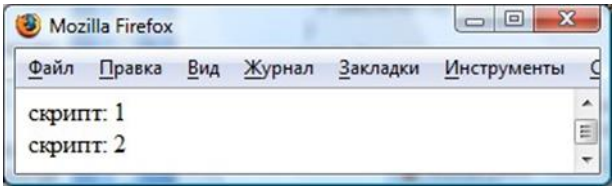


Область видимости глобальных переменных простирается на весь html-документ, тогда как область видимости локальных переменных ограничивается только блоком, в котором она определена. Область видимости — это строки кода, где доступно текущее значение переменной. В примере 8 создается локальная переменная лп, значение которой может быть выведено только из тела функции. Внешний вызов будет проигнорирован браузером.

В следующем примере создается в скрипте глобальная переменная *s*, значение которой увеличивается на 1 при каждом вызове функции *глоб_Пер*.

Пример 9. Глобальная переменная.

```
var s = 0
function глоб_Пер () {
   s++
}
глоб_Пер ()
document.write ("скрипт: ",s,"<br>")
глоб_Пер ()
document.write ("скрипт: ",s,"<br>")
```



3.1 Анонимные функции

В JavaScript можно создавать анонимные функции, т. е. функции без имени. Анонимные функции могут выступать в качестве значения переменной, такого же как строка или число. Поэтому функции можно присваивать переменным, элементам массива и свойствам объектов, а также передавать в качестве аргументов другим функциям.

$$var\ \alpha = function\ ([\beta,\gamma...])\ \{\ \delta\ \}$$

где α — имя переменной, элемента массива, свойства объекта; $\beta,\gamma...$ — необязательные входные аргументы (параметры) функции; δ — тело функции, состоящее из последовательности операторов (команд) JavaScript.

К анонимной функции можно обращаться по имени α.

Анонимные функциии называют еще функциями-выражениями. Функция, определяемая в качестве функции-выражения, в спецификации языка называется Function Expression (сокращенно FE). В отличие от именованной функции Function Declaration (FD), определения FE не поднимаются вверх, поэтому их нельзя вызывать до того момента, пока интерпретатор не достиг строки с определением функции.

Пример 1

```
var случайное_целое = function () {
   return Math.round(Math.random()*100)}
document.write (случайное_целое())
```

Пример 2

```
var приветствие2 = function () {
   строка='Доброе время суток! \r\n Мы рады приветствовать Вас на нашем сайте'
   alert (строка)
}
приветствие2()
```

Пример 3

```
var x = prompt ("Введите значение x");
 if (x > 0)
 вывод = function() { document.write("x больше 0") }
else
вывод = function() {document.write("x меньше или равно 0")}
вывод()
```

В примерах создается функция, а затем переменной присваивается ссылка на нее. Вызов функции на исполнение осуществляется через имя переменной с круглыми скобками, в которых могут быть параметры. При этом вызвать анонимные функции можно только после их описания.

Задание

Придайте функции **кратное** анонимный вид

```
function кратное (число, делитель) {
  if (число % делитель == 0)
    return true
```

```
  else
    return false
}
```

3.2 Анонимная функция как обработчик события

Выше был рассмотрен способ вызова функции обработки события (параграф «Базовые события») путем добавления соответствующего атрибута в тег html-элемента. Существует еще один способ вызова обработчика события непосредственно в скрипте через обращение к элементу по его идентификатору id с использованием анонимной функции.

$$\alpha.\beta = function\ () \{\ \delta\ \}$$

где α — идентификатор элемента; β — имя события; δ — тело обработчика.

Пример

```
<input id="кнопка" type="button" value="Нажми меня">
<script type=”text/javascript”>
    кнопка.onclick = function() {
      alert( 'Привет нажиматель' )
}
</script>
```

Задание

Создайте обработчик события для элемента img с помощью анонимной функции вместо атрибутов-событий.

```
<IMG SRC=“smile.gif” onMouseOver=”смени_изо ()”
onMouseOut=“верни_изо ()”>
```

Методы **addEventListener** и **removeEventListener** позволяют добавить или удалить обработчик события соответственно для какого-либо html-элемента.

$$\alpha.addEventListener(\beta,\gamma[,\delta])$$

где α — идентификатор элемента; β — имя события (пишется только строчными буквами без префикса on); γ — имя обработчика события; δ — необязательный параметр, фаза, на которой обработчик должен сработать.

Добавим еще один обработчик для кнопки предыдущего примера.

```
function обработчик2() { alert(' Хорошего настроения!') }
function обработчик3() { alert(' Успешного изучения!') }
кнопка.addEventListener("click", обработчик2)
кнопка.addEventListener("click", обработчик3)
```

Теперь при нажатии кнопки будет выдано последовательно три диалоговых окна.

Задание

Добавьте с помощью метода addEventListener обработчик события onMouseOut для вышеприведенной кнопки.

Метод removeEventListener удаляет созданный обработчик события.

$$\alpha.removeEventListener(\beta,\gamma[,\delta])$$

Чтобы удалить событие, необходимо передать те же аргументы, что были у addEventListener. Например, чтобы удалить обработчик2 из предыдущего примера, нужно написать оператор

```
кнопка.removeEventListener("click", обработчик2)
```

Задание

Удалите с помощью метода removeEventListener обработчик события onMouseOut для вышеприведенной кнопки.

3.3 Стрелочные функции

В ES2015 появились стрелочные функции. В описании стрелочной функции ключевое слово function убирается и добавляется стрелка после перечисления аргументов. Например, анонимную функцию

```
xxx = function (x) { return x * x * x }
```

можно заменить стрелочной

```
xxx = (x) => x * x * x
```

Если функция возвращает результат выражения, то return и фигурные скобки тоже можно не писать. Стрелку нельзя переносить на новую строку.

Если аргумент один, то круглые скобки необязательны, а вот если их несколько или ни одного — то скобки требуются.

Стрелочные функции очень удобны при работе с массивами в таких методах, как map, reduce, filter, every и some (переборные методы массива). Также стрелочные функции полезны в виде функций обратного вызова.

В обычной функции значение this связывается с элементом, по которому произошел клик, а в стрелочной функции значением this будет объект window.

Задание

Перепишите функцию **случайное_целое** из обычного вида в стрелочный вид.

```
function случайное_целое (x)  {
  return Math.round(Math.random()*x)
}
```

3.4 Функции обратного вызова

Функцию, которая передается в качестве аргумента другой функции, называют функцией обратного вызова или callback-функцией. В JavaScript функции — это объекты класса Function, создаваемые конструктором Function. Поэтому функции могут принимать другие функции в качестве аргументов, а также функции могут возвращать функции в качестве результата. Функции, которые это умеют, называются функциями высшего порядка.

Обратный вызов позволяет в функции исполнять код, который задается в аргументах. Этот код может быть определен в других местах программного кода и может быть недоступным для прямого вызова из этой функции.

Пример

```
function xy (x1,y1,callback) {
   document.write ('для чисел ' + x1 + ' и '+ y1)
   callback (x1,y1)
}
xy (5,7, function (x,y) {s=x+y;document.write ('сумма = ' + s)})
xy (5,7, function (x,y) {p=x*y;document.write ('произведение = ' + p)})
```

В примере показано использование обратной функции с параметрами. При вызове основной функции *xy* создается обратная функция и сразу выполняется.

3.5 Функция как объект

Существует еще один способ создания функции — через конструктор.

α = *new Function* (β,[γ,δ, ...])

где α — имя функции; $\beta,\gamma,\delta,\ldots$ — строковые аргументы, сначала идут параметры, на последнем месте — тело функции, которое тоже выступает строкой.

Если аргумент всего один — то это тело функции.

Пример

```
среднее = new Function('a','b','c','return (a+b+c)/3')
alert(среднее(1,2,3)) // 2
```

3.6 Математические функции. Объект math

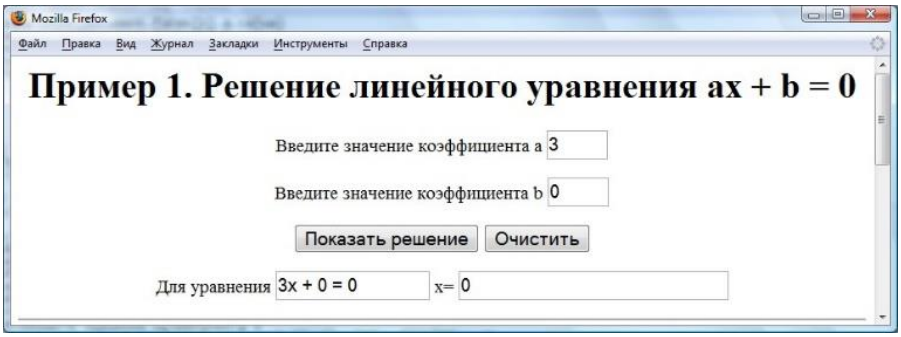
В JavaScript существует объект Math, свойства и методы которого позволяют производить большое число математических операций.

Свойство	**Описание**	**Значение**
Math.E	e	2.718281828459045
Math.LN2	ln2	0.6931471805599453
Math.LN10	ln10	2.302585092994046
Math.LOG2E	$\lg_2 e$	1.4426950408889634
Math.LOG10E	$\lg_{10} e$	0.4342944819032518
Math.PI	π	3.141592653589793
Math.SQRT1_2	$\sqrt{\frac{1}{2}}$	0.7071067811865476
Math.SQRT2	$\sqrt{2}$	1.4142135623730951
Метод	**Описание**	**Значение**
Math.abs (α)	$\lvert\alpha\rvert$	абсолютное значение аргумента (модуль)
Math.acos (α)	$\arccos\alpha$	арккосинус аргумента в диапазоне от 0 до π
Math.asin (α)	$\arcsin\alpha$	арксинус аргумента в диапазоне от $-\frac{\pi}{2}$ до $\frac{\pi}{2}$
Math.atan (α)	$\mathrm{arctg}\alpha$	арктангенс аргумента в диапазоне от $-\frac{\pi}{2}$ до $\frac{\pi}{2}$
Math.atan2 (α,β)	$\mathrm{arctg}\frac{\alpha}{\beta}$	α, β — декартовы координаты
Math.ceil (α)		наименьшее целое, большее или равное α
Math.cos (α)	$\cos\alpha$	косинус аргумента
Math.exp (α)	e^{α}	e в степени α

Math.floor (α)		наибольшее целое, меньшее или равное α
Math.log (α)	$\ln\alpha$	натуральный логарифм числа α
Math.max ($\alpha,\beta,$ ...)		максимальное значение из двух и более аргументов
Math.min ($\alpha,\beta,$ ...)		минимальное значение из двух и более аргументов
Math.pow (α,β)	α^{β}	α в степени β
Math.random ()		случайное число из диапазона от 0 до 1
Math.round (α)		целое число, ближайшее к α
Math.sin (α)	$\sin\alpha$	синус α
Math.sqrt (α)	$\sqrt{\alpha}$	квадратный корень из α
Math.tan (α)	$\mathrm{tg}\alpha$	тангенс α

где α — число либо числовая переменная или выражение.

алг **решение линейного уравнения**
нач цел a,b вещ x
| ввод a,b
| если a=0 и b=0
| | то
| | | вывод «решений бесконечно много»
| все
| если a=0 и b≠0
| | то
| | | вывод «решений нет»
| все
| если a≠0
| | то
| | | x := - α/β
| | | вывод «x = », x
| все
кон




```
<html>
<head>
```

```
<style type = ”text/css”>
    INPUT, SELECT {font-size:14pt}
</style>
    <script type = "text/JavaScript">
    function очистить () {
       document.forms[0].reset()
       document.forms[0].a.value=""
       document.forms[0].b.value=""
       }
    function показать_решение_уравнения () {
       a=document.forms[0].a.value
       b=document.forms[0].b.value
       уравнение=a + "x + " + b + " = 0"
       document.forms[0].eqv.value=уравнение
       if ((a==0) && (b==0))
           document.forms[0].x.value= 'решений бесконечно много'
       if ((a==0) && !(b==0))
           document.forms[0].x.value= 'нет решения'
       if (!(a==0)) document.forms[0].x.value=-b/a
       }
</script>
</head>
<body style = "font-size: 14pt" >
  <H1 style = ”text-align: center”> </H1>
```

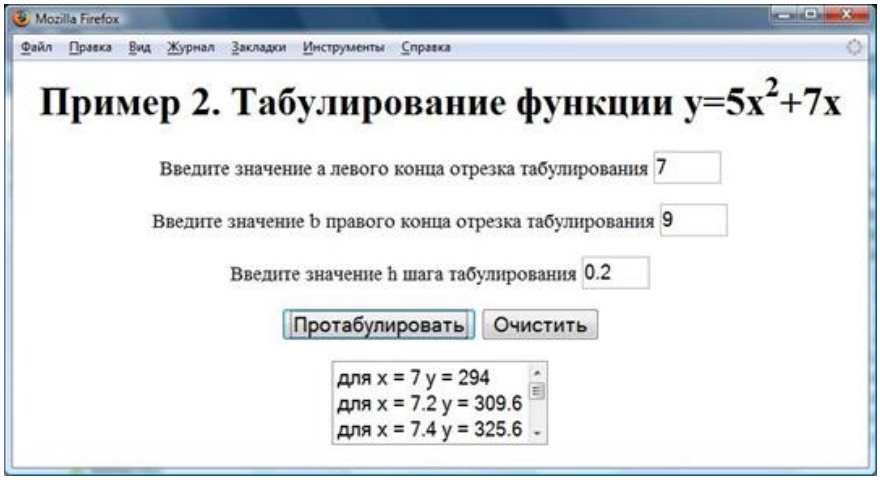
Пример 1. Решение линейного уравнения ax + b = 0.

```
<form> <div style = ”text-align: center”>
  <P>Введите значение коэффициента a <input name="a" type="text" size="3" value="7" > </P>
  <P>Введите значение коэффициента b <input name="b" type="text" size="3" value="9"> </P>
  <input type="button" value="Показать решение" on Click="показать_решение_уравнения ()">
   <input type="button" value=”Очистить” onClick="очистить()">
   <P> Для уравнения
   <input name="eqv" type="text" size="15" readonly> x=
   <input name="x" type="text" size="30" readonly> </P>
 </div> </form>
</body>
</html>
```

В функции *очистить* и *показать_решение_уравнения* встречается конструкция типа *document.forms[0].a.value*, которая возвращает введенное посетителем значение в соответствующий элемент формы (в данном примере в текстовое поле с именем *a*).

Следующий скрипт реализует алгоритм табулирования функции $y = 5x^2 + 7x$ на отрезке от *a* до *b* с шагом *h*.

```
алг табулирование функции y = 5 x^2 + 7 x
нач цел a,b вещ h
|       ввод a,b,h
|       |нц для x от a до b шаг h
|       |       y := 5 * Math.pow (x,2) + 7 * x
|       |       вывод y
|       кц
кон
```




```
<html>
<head>
  <style type = ”text/css”>
  INPUT, SELECT {font-size: 14pt}
  </style>
    function очистить () {
      document.forms[0].a.value = ""
      document.forms[0].b.value = ""
      document.forms[0].h.value = ""
      длина_списка=document.forms[0].tab_list.length
      for (i=0; i<=длина_списка; i++)
      document.forms[0].tab_list.options[0] = null
    }
    function show_tab () {
      a=parseInt(document.forms[0].a.value)
      b=parseInt(document.forms[0].b.value)
      h=parseFloat(document.forms[0].h.value)
      i=0
      for (x=a;x<=b;x=x+h) {
        y=5 * Math.pow (x, 2) + 7 * x
        t="для x = " + Math.round (x*10) / 10 + " y = " + Math.round (y*10) / 10
        document.forms[0].tab_list.options[i]= new Option(t)
        i ++
      }
```

```
    }
  </script>
</head>
<body style="font-size: 14pt" >
  <H1 style = ”text-align: center”>
```

Пример 2. Табулирование функции y=5x²+7x.

```
</H1>
<form>
<div style = ”text-align: center”>
    <P>Введите значение a левого конца отрезка табулирования
    <input name="a" type="text" size="3" value="7" > </P>
    <P>Введите значение b правого конца отрезка табулирования
    <input name="b" type="text" size="3" value="9"> </P>
    <P>Введите значение h шага табулирования
    <input name="h" type="text" size="3" value="0.2"> </P>
    <P> <input type="button" value="Протабулировать" onClick="show_tab ()">
    <input type="button" value="Очистить" onClick="очистить ()"> </P>
    <select name='tab_list' size=3 > </select>
</div>
</form>
</body>
</html>
```

В скрипте используются функции преобразования parseInt и parseFloat для перевода введенных пользователем значений из текстового формата в числовой. Если вводимые значения предполагаются целыми, то эти функции можно не использовать, как в предыдущем примере.

В функции *show_tab* есть строка, которая заполняет список, размещенный в форме значениями функции для соответствующего аргумента

```
document.forms[1].tab_list.options[i]= new Option(t)
```

где *forms[1]* — вторая форма в коллекции форм; *tab_list* — имя элемента select; *options* — коллекция пунктов списка; *i* — новый пункт списка с номером *I*; *Option(t)* — конструктор; *t* — переменная, значение которой будет присвоено новому элементу списка.

3.7 Динамическое изменение списка

Для добавления нового пункта в размещенный список формы и других действий с пунктами списка используется конструктор *Option* (смотри

предыдущий пример).

$$\alpha.options[\beta] = new\ Option\ ([\ \gamma_1, [\gamma_2, [\gamma_3, [\gamma_4\ ]\ ]\ ]\ ])$$

где α — имя элемента *select*; β — номер нового пункта списка; γ_1 — строка, которая размещается как пункт списка, можно обращаться к этому значению, используя свойство *text*; γ_2 — значение, которое передается серверу при выборе данного пункта, можно обращаться к этому значению, используя свойство *value*; γ_3 — пункт выбран или нет по умолчанию (true/false), можно обращаться к этому значению, используя свойство *defaultSelected*; γ_4 — пункт выбран или нет в текущий момент (true/false), можно обращаться к этому значению, используя свойство *selected*.

Для изменения значения пункта списка скриптом используется свойство *text*.

```
document.forms[0].список2.options[5].text="новое значение"
```

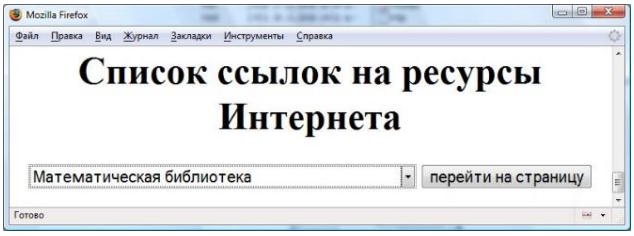
Для определения номера выбранного пункта списка используется свойство *selectedIndex* объекта *select* (смотри пример 3).

Элементы в список можно не только добавлять, но и удалять с помощью служебного слова *null*.

$$\alpha.options[\beta] = null$$

В функции *очистить* следует обратить внимание на необходимость использования переменной *длина_списка*, хранящей число пунктов (значений функции) заполненного списка. После удаления очередного пункта списка число его пунктов сокращается на 1. То есть длина списка во время удаления его пунктов становится величиной динамической. Поэтому в цикле необходимо использовать зафиксированное значение первоначальной длины для задания числа повторений операции удаления.

Пример 3. Поле с выпадающим списком ссылок.

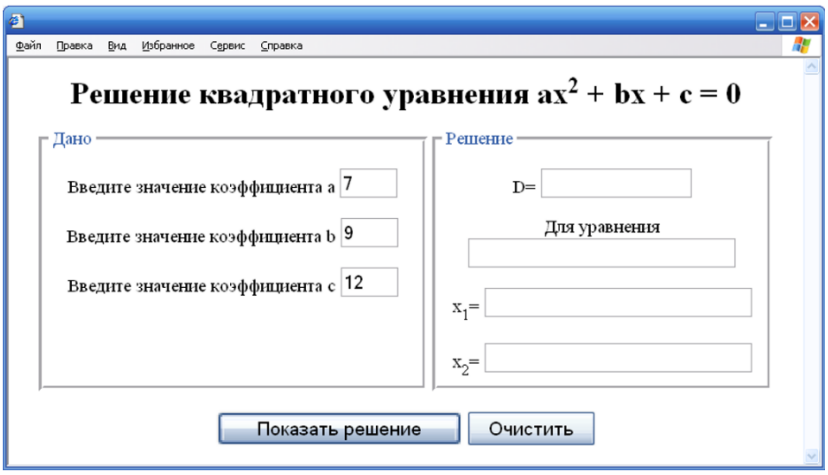


```
<html>
<head>
<title> Поле с выпадающим списком ссылок </title>
<script type = "text/javascript">
function перейти () {
выбранный_пункт_списка=document.forms[0].ссылки.selectedIndex
return (document.forms[0].ссылки.options[выбранный_пункт_списка].value)
}
</script>
</head>
<body>
  <h1> Список ссылок на ресурсы Интернета </h1>
  <form>
    <select name='ссылки' size=1>
    <option value = "http://www.softmath.com/"> Algebrator </option>
    <option value = "http://www.exponenta.ru"> Экспонента </option>
    <option value = "http://www.math.ru/lib/"> Математическая библиотека </option>
    <option value = "htttp://golovolomka.hobby.ru/"> Головоломки для умных людей</option>
    <option value = "http://www.etudes.ru/"> Математические этюды</option>
    <option value = "http://math.fizteh.ru/study/literature.esp">Математический анализ.
Электронные учебники </option>
    </select>
  <input type = “button” value = "перейти на страницу" onClick = "document.location.href =
перейти ()">
  </form>
</body>
</html>
```

Практическая работа №1

Задание 1. Квадратное уравнение

Разработайте сценарий для веб-страницы, который по заданным коэффициентам a, b, c вычисляет и выводит на страницу корни квадратного уравнения.



Задание 2. Табулирование любой функции

Усовершенствуйте пример 2 таким образом, чтобы табулирование осуществлялось для любой выбранной из списка функции. Дополнительно предусмотрите, чтобы сценарий определял монотонность функции на заданном отрезке и находил наибольшее и наименьшее значение.

Задание 3. Свойства числа

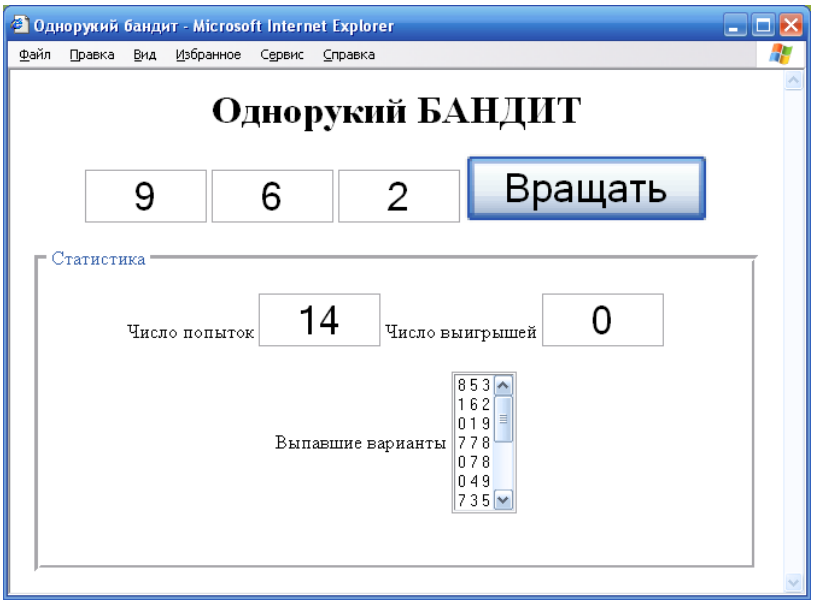
Разработайте сценарий для веб-страницы, который по введенному числу определяет, к какой группе оно относится: простые, четные, нечетные, и добавляет это число в соответствующую группу (список).

Задание 4. Перевод единиц

Разработайте сценарий для веб-страницы, который предлагает перевести введенное число из одной системы в другую: из градусов в радианы, из градусов по Цельсию в градусы по Кельвину, из одной системы счисления в другую и т.д.

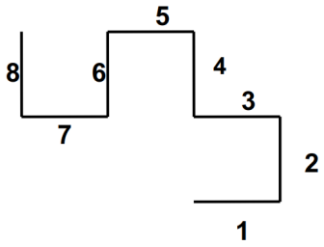
Задание 5. Однорукий бандит

Разработайте сценарий для веб-страницы, который выводит в текстовые поля три случайных целых числа в диапазоне от 0 до 9 включительно по нажатию посетителем кнопки «вращать». В случае выпадения трех одинаковых чисел сценарий увеличивает число выигрышей. Кроме этого, сценарий ведет подсчет числа попыток.



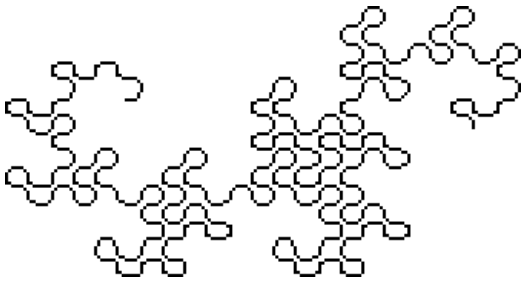
Задание 6. Кривая дракона

Разработайте сценарий для веб-страницы, который генерирует последовательность из N чисел, отражающую ход кривой дракона, т. е. очередность поворотов кривой. Присвоим код 1 для поворота налево и код 3 для поворота направо. Для кривой на рисунке последовательность следующая: 1 1 3 1 1 3 3.



Кривую можно продолжить дальше, и при этом окажется, что если очередной поворот имеет четный номер, то он равен тому члену последовательности, номер которого получается делением номера очередного четного поворота на 2. Например, следующий член в нашей последовательности имеет порядковый номер 8. Это число четное, значит, сам член равен члену с номером 4 (8/2), а это есть 1. Таким образом, восьмой член последовательности равен 1.

В случае, когда номер очередного поворота нечетный, член последовательности равен остатку от деления данного номера на число 4. Например, следующий неизвестный член последовательности имеет номер 9.

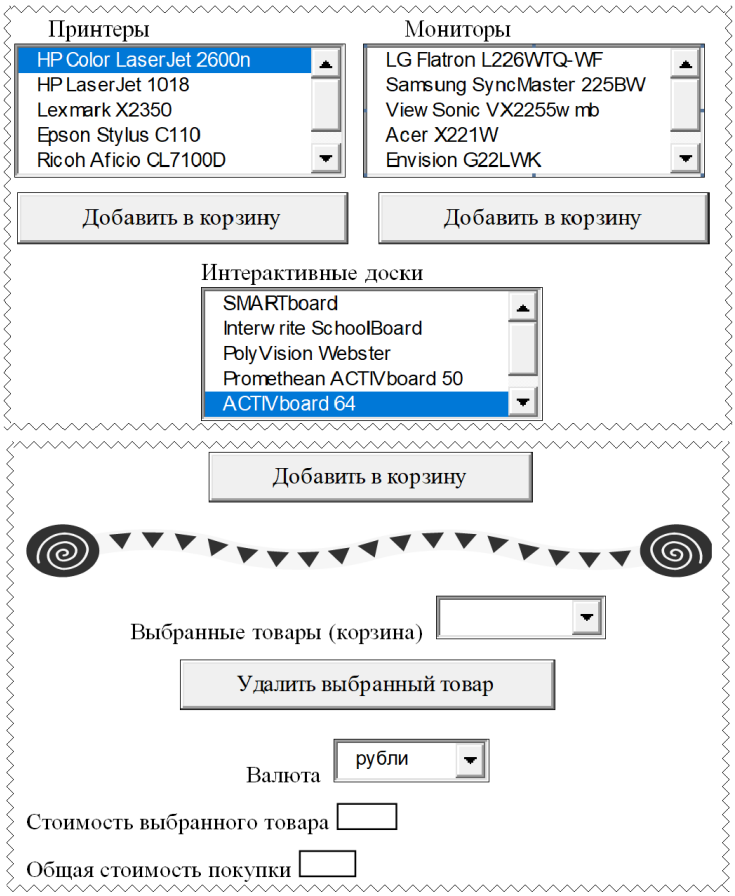


Это нечетное число. Поэтому сам член последовательности равен 1.

Разработайте скрипт, рисующий кривую дракона на холсте веб-страницы.

Задание 7. Магазин

Разработайте сценарий для веб-страницы, который формирует корзину выбранного посетителем периферийного оборудования для ПК.



4 Работа с массивами

Массив — это последовательность однотипных элементов, имеющих общее имя. В случае, когда элементов еще или уже нет, а имя объявлено, то говорят, что массив пустой. Массивы в JavaScript должны быть объявлены обязательно, в отличие от переменных

$$var\ \alpha = new\ Array$$

где α — имя массива; β — число элементов массива.

После объявления массива его можно заполнить элементами.

$$\alpha\,[\delta]=\chi$$

где α — имя массива; δ — порядковый номер элемента; χ — присваиваемое значение.

```
var m=new Array()
m[0]="<a href=news.htm>Новости</a>"
m[1]="<a href=listofmembers.htm>Список учащихся</a>" m[2]="<a
href=timetable.htm>Расписание занятий</a>" m[3]="<a href=docs.htm>Документы</a>"
m[4]="<a href=works.htm>Лабораторные работы</a>"
m[5]="<a href=method_kopilka.htm>Методические материалы</a>" m[6]="<a
href=contact.htm>Обратная связь</a>"
m[7]="<a href=annotation.htm>Аннотация учебных курсов</a>"
```

Существует несколько способов заполнения массива в момент его объявления.

$$var\ \alpha = new\ Array\ (\eta_1\ \{,\eta_2\})$$
$$var\ \alpha = [\eta_1\ \{,\eta_2\}]$$

где α — имя массива, η_1 , η_2, ... — элементы массива.

```
var картинки = new Array (“picture1.jpg”, “picture2.jpg”, “picture3.jpg”, “picture4.jpg”,
“picture5.jpg”)
```

или

```
var картинки = [“picture1.jpg”, “picture2.jpg”, “picture3.jpg”, “picture4.jpg”, “picture5.jpg”]
```

4.1 Основные свойства и методы объекта Array

Свойство *length* объекта *Array* хранит данные о количестве элементов массива.

$$\alpha.length$$

```
for (y = 0; y < m.length; y++) document.write ('<p>', m[y], '</p>')
```

Метод concat объединяет элементы исходного массива с элементами другого массива или с несколькими указанными элементами.

$$\alpha.concat(\{\beta,\})$$

где α — имя массива; β — какое-либо значение или имя массива.

```
картинки_m = m.concat (картинки)
for (y = 0; y < картинки_m.length; y++)
  document.write ('<p>', картинки_m [y], '</p>')
картинки2 = картинки.concat ("pict")
for (y = 0; y < картинки2.length; y++)
  document.write ('<p>', картинки2 [y], '</p>')
```

Метод	Параметры	Действие
$\alpha.join\ ([\beta])$	α — имя массива, β — разделитель элементов (по умолчанию запятая)	преобразование элементов массива в строку
$\alpha.pop\ ()$	α — имя массива	удаление последнего элемента массива
$\alpha.push\ (\{\beta,\})$	α — имя массива, β — какое-либо значение или имя массива	добавление элементов в конец массива
$\alpha.reverse\ ()$	α — имя массива	изменяет порядок следования элементов массива на обратный
$\alpha.shift\ ()$	α — имя массива	удаляет первый элемент из массива и возвращает его
$\alpha.slice\ (\beta_н\ ,[\beta_к])$	α — имя массива, $\beta_н$ — индекс начального элемента фрагмента, $\beta_к$ — индекс конечного элемента фрагмента	возвращает фрагмент массива, начиная с элемента $\beta_н$ и заканчивая элементом $\beta_{к-1}$ включительно

α.sort ([β])	*α* — имя массива,
	β — имя функции, определяющей направление сортировки.
	Умолчание — по возрастанию
		сортировка массива
		// по возрастанию *β*= function f (el1, el2) {
		return (el1 — el2) }
		//по убыванию
		el2–el1

α.splice ($\beta_н$ [,*δ*][,{*γ*,}])	*α* — имя массива, $\beta_н$ — номер элемента, *δ* — число элементов, *γ* — значение	1. Удаляет *δ* элементов массива, начиная с $\beta_н$, когда *δ*>0 или не определено (до конца массива), а также когда не задано *γ*. 2. Удаляет и вставляет элементы, начиная с $\beta_н$, когда *δ*>0 или не определено и заданы значения *γ*. 3. Вставляет элементы перед $\beta_н$, когда *δ*=0 и заданы значения *γ*
α.unshift ({β,})	*α* — имя массива, *β* — какое-либо значение	вставляет элементы *β* в начало массива

В методе sort внутри самого интерпретатора JavaScript реализован универсальный алгоритм быстрой сортировки. Для примера напишем скрипт для сортировки массива одним из простых неэффективных методов. Для упорядочения элементов по возрастанию находим наименьший элемент массива и меняем его местами с первым элементом, затем находим минимальный элемент в массиве начиная со второго члена и меняем его со вторым и так до конца массива.

```
const n=5
let массив = new Array()
for (i=0; i<n; i++) // инициализация массива
   массив.push(Math.round(Math.random()*100))
console.log(массив) // вывод исходного массива
let tmp=0 //сортировка массива
for (i=0; i<n-1; i++) {
  for (j=i; j<n; j++) {
    if (массив[j]<массив[i]) {
      tmp=массив[j];массив[j]=массив[i];массив[i]=tmp
    }
  }
}
console.log(массив) // вывод отсортированного массива
```

4.2 Перебирающие методы объекта Array

Метод	Параметры	Действие
α.forEach(β[, γ])	*α* — имя массива, *β* — функция обратного вызова (имеет три параметра, из которых первый является обязательным), *γ* — позволяет указать контекст `this` для *β*	метод передает функции *β* три аргумента: очередной элемент массива, номер элемента и сам массив. Метод заменяет цикл перебора всех элементов массива, т. е. функция *β* вызывается для каждого элемента массива, первый аргумент функции при этом возвращает очередной элемент массива, второй — его номер

```
var s=0
var arr1 = [1,2,3,4,5]
    arr1.forEach(function(item, i, arr) { s+=item })
alert (s) // 15
var array1 = ['a', 'b', 'c']
    array1.forEach(function(element) { alert(element) })
```

α.filter(β[, γ])	*α* — имя массива, *β* — функция обратного вызова (имеет три параметра, из которых первый является обязательным), *γ* — позволяет указать контекст `this` для *β*	метод создает новый массив, в который войдут только те элементы *α*, для которых вызов *β* возвратит `true`

```
var words = ['яблоко', 'груша', 'апельсин', 'мандарин', 'грейпфрут', 'киви'];
const result = words.filter (word => word.length > 7)
alert(result) // апельсин, мандарин, грейпфрут
var arr = [1, 2, 3, 4, 5];
var четные = arr.filter (number =>number%2==0)
alert(четные); // [2,4]
```

α.`map`*(β[, γ])*	*α* — имя массива, *β* — функция обратного вызова (имеет три параметра, из которых первый является обязательным), *γ* — позволяет указать контекст `this` для *β*	метод вызывает функцию обратного вызова *β* один раз для каждого элемента в массиве по порядку и создает новый массив из результатов

```
var nums = [1, 4, 9, 16]
const nums1 = nums.map (x => x + 1)
```

alert(nums1) // [2, 5, 10, 17]

α.every(*β*[, *γ*])	*α* — имя массива, *β* — функция обратного вызова (имеет три параметра, из которых первый является обязательным), *γ* — позволяет указать контекст this для *β*	метод возвращает true, если функция обратного вызова *β* вернет true для каждого элемента массива *α*
α.some(*β*[, *γ*])	*α* — имя массива, *β* — функция обратного вызова (имеет три параметра, из которых первый является обязательным), *γ* — позволяет указать контекст this для *β*	метод возвращает true, если функция обратного вызова *β* вернет true хотя бы для одного элемента массива *α*

var nums = [1, -1, 2, -2, 3]
const nums1 = nums.**every**(x => x > 0)
alert (nums1) // false, не все положительные
const nums2 = nums.**some**(x => x > 0)
alert (nums2) // true, есть хоть одно положительное

α.reduce(*β*[, *γ*])	*α* — имя массива, *β* — функция обратного вызова (имеет 4 параметра, из которых первый и второй являются обязательными, первый выполняет роль аккумулятора — возвращаемого значения), *γ* — начальное значение	метод передает функции *β* 4 аргумента: последний результат вызова функции, очередной элемент массива, номер элемента и сам массив. Метод возвращает значение первого аргумента возвратной функции. Если задан *γ*, то на первом вызове значение первого аргумента обратной функции будет равно *γ*. Если *γ* не задан, то на первом вызове значение первого аргумента обратной функции будет равно первому эле- менту массива, а перебор начинается со второго

const numbers = [1, 2, 3, 4]
a=numbers.**reduce**((accumulator, currentValue) => accumulator + currentValue)
alert(a) // 10
a=numbers.**reduce**((accumulator, currentValue) => accumulator + currentValue, 5);
alert(a) // 15

α.reduceRight(*β*[,*γ*])	*α* — имя массива, *β* — функция обратного вызова (имеет 4 параметра, из которых первый и второй являются обязательными, первый выполняет роль аккумулятора — возвращаемого значения), *γ* — начальное значение	метод аналогичен reduce, только элементы перебираются справа налево

Вложенные массивы

В JavaScript довольно просто организовать вложенные массивы.

Например:

```
let вМассив = [
   [243, 12, 23, 12, 45, 45, 78, 66, 223, 3],
   [34, 2, 1, 553, 23, 4, 66, 23, 4, 55],
    [67, 56, 45, 553, 44, 55, 5, 428, 452, 3],
    [12, 31, 55, 445, 79, 44, 674, 224, 4, 21],
    [4, 2, 3, 52, 13, 51, 44, 1, 67, 5],
    [5, 65, 4, 5, 5, 6, 5, 43, 23, 4424],
    [74, 532, 6, 7, 35, 17, 89, 43, 43, 66],
    [53, 6, 89, 10, 23, 52, 111, 44, 109, 80],
    [67, 6, 53, 537, 2, 168, 16, 2, 1, 8],
    [76, 7, 9, 6, 3, 73, 77, 100, 56, 100]
]
```

Для обращения к 9 элементу 5 строки нужно составить следующий оператор:

```
console.log (вМассив[4][8]) // 67
```

Ответом будет выдано число 67.

Для генерации элементов массива можно использовать датчик случайных чисел, как показано в следующем примере.

```
{
const n=5 // число строк
const m=3 // число столбцов
let вМассив = new Array() // вложенный массив
let пМассив = new Array //простой массив
for (i=0; i<n; i++) {
   for (j=0; j<m; j++) {
   пМассив.push(Math.round(Math.random()*100))
```

```
    }
    вМассив.push(пМассив) // добавление очередной строки
    пМассив=[] // обнуление простого массива
}
console.log (вМассив)
}
```

Вместо вывода массива в консоль можно написать код для вывода элементов массива на веб-страницу.

```
for (i=0; i<n; i++) {
document.write("<p>",вМассив[i],"</p>");
}
```

4.3 Оператор расширения

В ES2015 появился оператор расширения (…), который позволяет разделять доступные для перебора значения.

При вставке массива в другой массив получается вложенный массив, но если необходимо добавить не сам массив, а лишь его элементы (содержимое), то быстрее всего это сделать с помощью оператора расширения.

```
{
let м1 = [31, 14]
let м2 = [19, 12, mid, 15, 16]
console.log(м2) // [19, 12, [31, 14], 15, 16]
let м2 = [19, 12, ...м1, 15, 16];
console.log(м2)) // [19, 12, 31, 14, 15, 16]
}
```

Часть методов математического объекта Math имеет в качестве аргументов только числа. Оператор расширения может позволить использовать массивы в качестве аргументов этих методов. Например, метод max находит максимальное из нескольких чисел. С оператором расширения можно находить максимальное из элементов массива.

```
{
var массив = [12, 14, 18, 26, -10]
var max = Math.max(...массив) console.log (max) // 26
}
```

Копирование массивов невозможно простым присваиванием имени массива новой переменной. Таким образом можно присвоить только ссылку на массив, что означает одновременное изменение значений у обеих переменных

при изменении значения у любой из них. Оператор расширения позволяет создать реальную копию массива.

```
{
let массив1 = ['a', 'b', 'c']
let массив2 = [...массив1]
console.log(массив2) // ['a', 'b', 'c']
}
```

Слияние нескольких массивов также удобно осуществлять оператором расширения.

```
{
let arr1 = [0, 1, 2];
let arr2 = [3, 4, 5];
let arr3 = [6, 7, 8];
arr1 = [...arr1, ...arr2, ...arr3];
console.log(arr1);
}
```

С помощью оператора расширения можно преобразовать строку в массив символов.

```
{
let строка = "столешница"
let массивБукв = [...строка]
console.log (массивБукв) // ["с", "т", "о", "л", "е", "ш", "н",  "и",  "ц", "а"]
}
```

Оператор расширения может не только разделять значения, но и собирать их в массив. Например, в функции при отсутствии описанных аргументов оператор соберет все значения в массив.

```
function средняяЗарплата (...работники) {
  let сумма=0
  for (i=0;i<работники.length;i++) {
    сумма+=работники[i]
  }
  return (сумма/работники.length)
}
alert (средняяЗарплата(100, 200, 300, 400, 500)) // 300
```

Практическая работа №2

Задание 1

Заполните массив из 7 элементов путем ввода значений с клавиатуры в ходе выполнения сценария. Разработайте сценарий:

a) нахождения квадратного корня из каждого элемента массива;
b) нахождения среднего арифметического двух соседних элементов массива.

Задание 2

Заполните массив из 15 элементов случайными целыми числами, лежащими в диапазоне от –50 до 50. Разработайте сценарий, который:

a) увеличивает все элементы массива в 2 раза;
b) разделит все элементы массива на первый элемент;
c) является ли k-й элемент массива четным числом.

Задание 3

В массиве хранятся сведения о количестве осадков, выпавших за каждый день июня. Разработайте сценарий, определяющий:

a) среднедневное количество осадков в этом месяце;
b) общее количество осадков, выпавших за каждую декаду месяца;
c) среднедневное количество осадков в каждой декаде месяца.

Задание 4

Дан массив из n вещественных случайных чисел в диапазоне от –100 до 100:

a) каждый отрицательный элемент массива замените его абсолютной величиной;
b) все элементы с нечетными номерами замените их квадратным корнем;
c) все элементы, лежащие в диапазоне от –5 до 5, замените 0.

Задание 5

В зрительном зале 25 рядов по 45 мест. Информация о проданных билетах хранится в двумерном массиве, номера строк которого соответствуют рядам зрительного зала, а номера столбцов номерам мест. Если билет продан, в массив заносится единица, если нет — ноль. Заполните массив нулями и единицами случайным образом. Разработайте сценарий, определяющий число проданных билетов в том или ином ряду по запросу.

Задание 6

Коммерческая компания имеет 25 торговых точек. Информация о доходе

каждой точки за каждый месяц года хранится в строке двумерного массива. Заполните массив случайными положительными числами. Разработайте сценарий, определяющий среднемесячный доход любого магазина по запросу.

Задание 7

Заполните двумерный массив размером 10 на 10 случайным образом вещественными числами в диапазоне от –1000 до 1000. Разработайте скрипт, который определяет:

a) сумму отрицательных элементов 5-й строки массива;
b) сумму элементов 4-го столбца массива, меньших k;
c) количество ненулевых элементов второго столбца, меньших s.

Задание 8

Заполните двумерный массив размером m на n случайным образом вещественными числами в диапазоне от минимального числа JavaScript до максимального числа JavaScript. Разработайте сценарий, который определяет:

a) максимальное значение среди элементов любой заданной строки;
b) минимальное значение среди элементов любого заданного столбца;
c) координаты (положение) минимального и максимального элементов всего массива.

Задание 9

Заполните два массива переменной длины случайными числами в диапазоне от –1000 до 1000. Разработайте скрипт, который

a) найдет сумму элементов обоих массивов;
b) найдет максимальный элемент обоих массивов;
c) добавит все четные элементы первого массива во второй массив.

Задание 10

Отсортируйте заданный массив из n целых случайных элементов методом пузырька. Алгоритм состоит из повторяющихся проходов по сортируемому массиву. За каждый проход элементы последовательно сравниваются попарно, и, если порядок в паре неверный, выполняется обмен элементов. Проходы по массиву повторяются N − 1 раз или до тех пор, пока на очередном проходе не окажется, что обмены больше не нужны, что означает — массив отсортирован.

5 Строки и регулярные выражения

5.1 Строки. Объект String

Определение и примеры строки рассмотрены на страницах 10–11 данного учебного пособия. В JavaScript существует объект *String* и каждая строковая переменная может создаваться через вызов конструктора.

$$var\ \alpha = new\ String(\beta)$$

где α — имя строковой переменной; β — строковое значение.

```
var строка1 = new String (‘Черепаха изучает информатику’)
```

Но строковую переменную не обязательно создавать через конструктор, можно так же, как и числовые переменные. Над строками, как и над числами, можно совершать операцию сложения. Для этого используется тот же знак “+”. В результате сложения строк получается одна строка, состоящая из всех слагаемых членов.

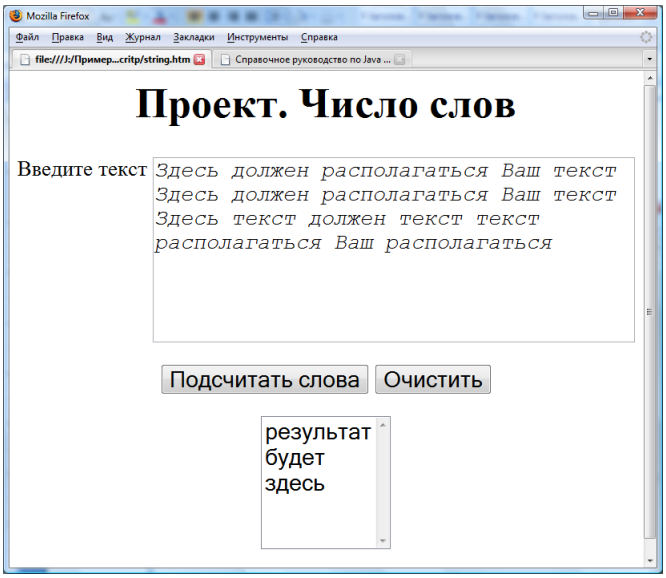
Свойства или методы	**Параметры**	**Значение или действие**
α.length	α — имя строковой переменной	возвращает длину строки, т. е. число символов строки
α.indexOf (β [,γ])	α — имя строковой переменной, β — подстрока, образец поиска, γ — номер элемента строки α, с которого производится поиск	возвращает номер, с которого начинается подстрока β в строке α, при осуществлении поиска слева направо
α.lastIndexOf (β [,γ])	α — имя строковой переменной, β — подстрока, образец поиска, γ — номер элемента строки α, с которого производится поиск	возвращает номер, с которого начинается подстрока β в строке α, при осуществлении поиска справа налево
α.charAt (β)	α — имя строковой переменной, β — номер элемента строки α	возвращает символ строки α с номером β

α.charCodeAt (β)	α — имя строковой переменной, β — номер элемента строки α	возвращает код символа строки α с номером β
String.fromCharCode (β_1 [{,β_2}])	β_1, β_2, ... — число, числовая переменная или функция	возвращает строку из символов по кодам β_1, β_2, ...
α.slice ($\beta_н$ [,$\beta_к$])	α — имя строковой переменной, $\beta_н$ — номер начального элемента фрагмента строки α, $\beta_к$ — номер конечного элемента фрагмента строки α	возвращает подстроку строки α с позиции $\beta_н$ до позиции $\beta_{к-1}$. Если $\beta_к$ не указан, то до конца строки α. Отрицательное значение $\beta_к$ ведет отсчет от конца строки
α.split (β [,γ])	α — имя строковой переменной, β — символ-разделитель слов (подстрок) в строке α, γ — максимальное число элементов	возвращает массив из подстрок строки α, разделенных символом β. Число элементов массива равно γ. Если γ не указан, то в массив войдут все подстроки
α.substr (β [,γ])	α — имя строковой переменной, β — номер начального элемента фрагмента строки α, γ — число символов подстроки	возвращает подстроку строки α, начиная с позиции β длиной γ. Если γ не указан, то с позиции β до конца строки α
α.substring ($\beta_н$ [,$\beta_к$])	α — имя строковой переменной, $\beta_н$ — номер начального элемента фрагмента строки α, $\beta_к$ — номер конечного элемента фрагмента строки α	возвращает подстроку строки α от позиции $\beta_н$ до $\beta_{к-1}$. Если $\beta_к$ не указан, то до конца строки α
α.toLowerCase ( )	переводит все символы строки α в нижний регистр	
α.toUpperCase ( )	переводит все символы строки α в верхний регистр	
α.replace (β, γ)	возвращает строку α, в которой заменено первое вхождение подстроки β на γ. В качестве β может стоять регулярное выражение	

Пример 1. Подсчитайте число вхождений заданной буквы в тексте.

```
количество_а = 0
текст = "Здесь должен располагаться Ваш текст"
   for (i = 0; i < текст.length; i++) {
     if (текст.charAt(i) == "а") количество_а++ }
```

Пример 2. **Число слов.** Разработайте сценарий, который подсчитает число одинаковых слов, встречающихся в текстовом поле веб-страницы.




```
<html>
<head>
<script type="text/JavaScript">
function подсчитать_слова () {
   var строка = document.forms[0].txt.value
   var m, mu = new Array (); m = строка.split (' ')
   var сколько_слов = 0, номер_слова = 0
   while (!m.length == 0) {
      сколько_слов = 1; текущее_слово = m [0]; mu = m.splice (0,1)
      for (i = 0; i < m.length; i++) {
         if (текущее_слово == m [i]) {
         сколько_слов++ mu=m.splice (i,1)
         i--
         }
      }
   document.forms[0].список_оригинальных_слов.
options [номер_слова]=new Option (текущее_слово+" - " + сколько_слов)
   номер_слова++
   }
}
function чистить () {
   n=document.forms[0].список_оригинальных_слов.length
   for (i=0; i<n; i++)
      document.forms[0].список_оригинальных_слов.options[0]=null
      document.forms[0].reset ()
}
</script>
</head>
```

```
<body style="font-size:20pt">
<H1 style=”text-align:center”>Проект. Число слов</H1>
<form>
<div style=”text-align:center”>
<p style=”text-align:top”> Введите текст
<textarea name="txt" rows="7" cols=35 style="display:inline; vertical- align:top; font-size:20pt;
font-style:italic;">
Здесь должен располагаться Ваш текст Здесь должен располагаться Ваш текст Здесь должен
располагаться Ваш текст
</textarea>
</p>
<input type="button" value="Подсчитать слова" style="font-size:21pt"
onClick="подсчитать_слова ()">
<input type="button" value="Очистить" style="font-size:21pt"
onClick = "чистить ()">
<p>
<select name='список_оригинальных_слов' size=5 style="font- size:21pt">
<option> результат
<option> будет
<option> здесь
</select>
</p>
</div>
</form>
</body>
</html>
```

5.2 Регулярные выражения

Регулярное выражение — это еще один тип данных, являющийся маской для строковых данных. Оно задает структуру данных. Регулярное выражение задается между двумя косыми чертами / / и применяется к строке. Например, для проверки ввода адреса электронной почты регулярное выражение в простейшем случае может быть таким: /@/

Регулярные выражения представлены в виде объектов RegExp (от *англ.* regular expression — регулярное выражение) и могут быть созданы динамически с помощью конструктора RegExp().

$$RegExp(\alpha\,[,\,\beta])$$

где α — строка или строковая переменная, являющаяся телом регулярного выражения; β — строка или строковая переменная, являющаяся флагом регулярного выражения.

var γ = new RegExp(*α [, β]*)

Например, конструктор *RegExp (“JavaScript”)* создает регулярное выражение */JavaScript/*. Конструктор *RegExp (“JavaScript”,”g”)* создает регулярное выражение */JavaScript/g*.

Флаг регулярного выражения	**Значение**
i	поиск, нечувствительный к регистру
g	глобальный поиск, т. е. поиск всех соответствий
m	многострочный поиск

Применить регулярное выражение к строке можно с помощью свойства *test* или *match*.

α.test(β)

где *α* — объект, дающий регулярное выражение; *β* — объект, возвращающий проверяемую строку. Свойство *test* возвращает логическое значение *true* или *false* в зависимости от того, содержит ли проверяемая строка *β* подстроку, соответствующую образцу *α*. Свойство match дает саму подстроку. *Например:*

```
alert (/@/.test("an171@mail.ru"))
```

Метод replace заменяет дающую регулярным выражением подстроку заданным образцом.

α.replace(β,γ)

где α — это исходная текстовая строка; β — регулярное выражение; γ — подстрока, которой заменяются части исходной строки в соответствии с регулярным выражением.

Например:

```
var s = "JavaScript исполняется браузером, поэтому JavaScript считается клиентской
технологией";
var re = /JavaScript/g;
alert (s.replace(re, "JScript"));
```

В приведенном примере все вхождения слова JavaScript будут заменены на JScript.

Если часть регулярного выражения заключена в круглые скобки, то соответствующая ей подстрока будет запомнена для последующего использования. Для доступа к запомненным подстрокам используются свойства $1, …, $9 объекта RegExp. *Например:*

```
var re = /([а-я]+)\s([а-я]+)/i;
var s = "Ален Делон";
alert (s.replace(re, "$2 $1"));
```

В приведенном примере порядок слов в исходной строке s изменен на обратный (Делон Ален). Комбинация **[а-я]** соответствует любой букве русского алфавита, символ **+** соответствует повторению предыдущего символа один или более раз. Комбинация **\s** соответствует символу пробела.

Спецсимволы для построения регулярных выражений

Спецсимвол	**Описание**
\d	соответствует цифре. Эквивалентно [0-9]. (d от digital)
\D	соответствует нецифровому символу. Эквивалентно [^0-9].
\w	соответствует цифре, букве латинского алфавита или подчеркиванию (w от word). Эквивалентно / [A-Za-z0-9_]/.
\W	соответствует символам, отличным от цифр, букв латинского алфавита и подчеркивания
.	соответствует символам, отличным от разделителя строки
\s	соответствует неотображаемым символам (которые разделяют данные на слова (пробел), абзацы, строки и т. д.). Эквивалентно /[\f\n\r\t\v]/.
\S	соответствует отображаемым символам. Эквивалентно /[^ \f\n\r\t\v]/.
[xyz]	соответствует символам из заключенных в квадратные скобки
[^xyz]	соответствует любому символу, кроме заключенных в квадратные скобки
\b	соответствует началу или концу слова (искл. для кириллицы)
\B	соответствует любому символу, кроме начала или конца слова (b от begin)
?	соответствует одному символу или указывает на отсутствие символа
*	соответствует нескольким символам или указывает на отсутствие символа

+	соответствует повторению предыдущего символа один или более раз
{n}	соответствует n вхождениям предыдущего символа
{n,}	соответствует n или более вхождениям предыдущего символа /x{1,}/ эквивалентно /x+/ /x{0,}/ эквивалентно /x*/
{n,m}	соответствует не менее чем n и не более чем m вхождениям предыдущего символа /x{0,1}/ эквивалентно /x?/
^	соответствует началу строки
$	соответствует концу строки
[a-z]	соответствует любому символу в указанном диапазоне

Метод exec сопоставляет строку с образцом, заданным регулярным выражением. Если не нашлись совпадения, то возвращается значение null. В противном случае результатом является массив подстрок, соответствующих заданному шаблону. Первый элемент массива будет равен исходной строке, удовлетворяющей образцу.

$$\alpha.exec(\beta)$$

Например:

```
var re=/(\d+\s)([а-я]+\s)(\d+\s)/
var arr=re.exec("Выборы президента России состоялись 20 марта 2018
года")
document.write("Дата выборов: ", arr[0], "<br>")
document.write("День выборов: ", arr[1], "<br>")
document.write("Месяц выборов: ", arr[2], "<br>")
document.write("Год выборов: ", arr[3], "<br>")
```

Результат работы скрипта:
Дата выборов: 20 марта 2018
День выборов: 20 Месяц выборов: марта Год выборов: 2018

Пример 1

```
Построение регулярного выражения — шаблона, проверяющего введенный текст на
соответствие фамилии и инициалам.
var re = /^[А-Я]\D{0,15}\D$/ var строка = "Иванов А.И."
var проверка_строки = re.test(строка) alert (проверка_строки)
```

Пример 2

Замена последовательности идущих подряд пробелов одним пробелом в строке.

```
текст=” Здесь должен располагаться Ваш текст”
текст=текст.replace(/ + /g," ")
```

5.3 Дата и время. Объект Date

Время в JavaScript представлено в миллисекундах (10^{-3}), прошедших с 1 января 1970 года по Гринвичу (GMT — Greenwich Mean Time, или UTC — Universal Coordinate Time, Всеобщее скоординированное время).

Объект *Date* — это мгновенный снимок, определяющий момент, когда он был сделан.

var α = new Date ()

где *α* — имя переменной, которой будет присвоено значение текущих даты и времени.

α = new Date(год, месяц, день[, часы, минуты, секунды])

В качестве аргумента объекта Date мы можем поместить любую дату и время. Метод `toLocaleString` возвращает дату в длинном формате по умолчанию.

```
let дата = new Date(2000,11,13)
console.log (дата.toLocaleString()) // 13.12.2000, 0:00:00
```

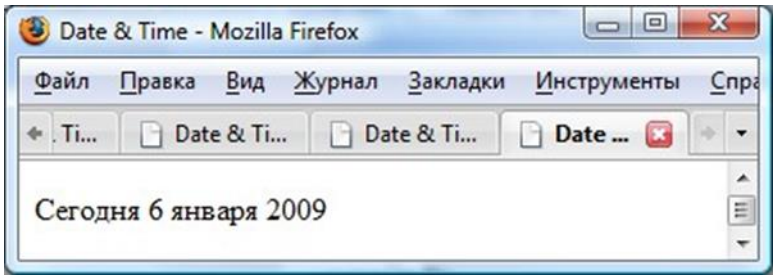
Метод объекта Date	**Возвращаемое значение**
α.getFullYear()	возвращает год в полном формате
α.getYear()	возвращает год в кратком или полном формате (в зависимости от браузера)
α.getMonth()	возвращает номер месяца (от 0 до 11)
α.getDate()	возвращает число месяца (от 1 до 31)
α.getDay()	возвращает день недели (от 0 до 6)
α.getHours()	возвращает час (от 0 до 23)
α.getMinutes()	возвращает минуты (от 0 до 59)
α.getSeconds()	возвращает секунды (от 0 до 59)
α.getMilliseconds()	возвращает миллисекунды (от 0 до 999)
α.getTime()	возвращает миллисекунды (от 01.01.1970 GMT)
α.setFullYear(β)	устанавливает год в полном формате
α.setYear(β)	устанавливает год в кратком или полном формате (в

	зависимости от браузера)
α.setMonth(β)	устанавливает номер месяца (от 0 до 11)
α.setDate(β)	устанавливает число месяца (от 1 до 31)
α.setHours(β)	устанавливает час (от 0 до 23)
α.setMinutes(β)	устанавливает минуты (от 0 до 59)
α.setSeconds(β)	устанавливает секунды (от 0 до 59)
α.setMilliseconds(β)	устанавливает миллисекунды (от 0 до 999)
α.setTime(β)	устанавливает миллисекунды (от 01.01.1970 GMT)

где α — имя переменной типа Date; β — число.

Пример 1

```
дата = new Date ();
день = дата.getDate ()
месяц = дата.getMonth ()
год = дата.getFullYear ()
var название_месяца = ["января", "февраля", "марта", "апреля", "мая", "июня", "июля",
"августа", "сентября", "октября", "ноября", "декабря"]
document.write ("<p> Сегодня " + день + " " + название_месяца
[месяц] + " " + год + "</p>")
```



Пример 2. Календарь с выделением текущей даты.

```
дата  = new Date ();  день_сегодня = дата.getDate()
d = 1
document.write ("<table border = 1>")
for (r = 1; r < 6; r++) {
   document.write ("<tr>")
   for (c = 1; c < 8; c++) {
      if (! (r == 1 && c < 4) && d <= 31) {
      if (день_сегодня == d)
         document.write ("<td bgcolor = pink>", d, "</td>")
      else
         document.write ("<td>", d, "</td>")
      d++
      }
      else
```

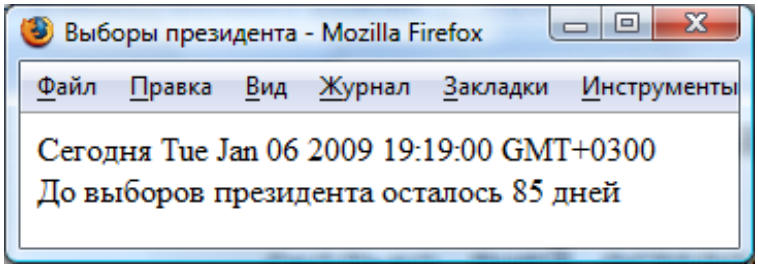
```
        document.write ("<td> </td>")
    }
    document.write ("</tr>")
}
document.write ("</table>")
```

			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

Над датами можно выполнять арифметические действия. Например, мы хотим знать, сколько осталось ждать дней до выборов президента.

Пример 3. Выборы президента.

```
ОДНА_СЕК = 1000
ОДНА_МИН = ОДНА_СЕК * 60
ОДИН_ЧАС = ОДНА_МИН * 60
ОДИН_ДЕНЬ = ОДИН_ЧАС * 24
сегодня = new Date ()
var сколько_миллисекунд_осталось = new Date ()
выборы_президента = new Date ()
день_выборов_президента = 01
месяц_выборов_президента = 04
год_выборов_президента = 2012
выборы_президента.setDate (день_выборов_президента)
выборы_президента.setMonth (месяц_выборов_президента)
выборы_президента.setFullYear (год_выборов_президента)
сколько_миллисекунд_осталось = выборы_президента - сегодня
сколько_дней_осталось = Math.round (сколько_миллисекунд_осталось /  ОДИН_ДЕНЬ)
document.write ("Сегодня ", сегодня, "<br>")
document.write ("До выборов президента осталось " +
сколько_дней_осталось + " дней ")
```



Практическая работа №3

Задание 1. Число слов

Усовершенствуйте скрипт из примера 2 таким образом, чтобы он учитывал знаки препинания, т. е. чтобы знаки препинания не влияли на подсчет одинаковых слов.

Задание 2. Транслит

Разработайте сценарий, который сможет перевести текстовую информацию, размещенную на веб-странице с русского языка в кириллице на русский, но в латинском алфавите, и предусмотрите обратный перевод.

Задание 3. Шифрограмма

Разработайте сценарий, который сможет зашифровывать многострочный текст, размещенный на веб-странице методом сдвига по алфавиту. Предусмотрите возможность расшифровки сообщения.

Задание 4. Число словами

Разработайте сценарий, который сможет словами написать заданное число. Например, 5 535 → пять тысяч пятьсот тридцать пять.

Задание 5. Автодобавление

Разработайте сценарий, который сможет по нажатию на соответствующую кнопку добавлять часто употребляемые слова, например: добрый день, спасибо, пожалуйста, к сожалению и т. д.

Задание 6. Автозамена

Разработайте сценарий, который сможет по введенному при наборе текста шаблону (например, дбр) автоматически заменять его на соответствующее слово (например, добрый).

Задание 7. Поиск и замена

Разработайте сценарий, который сможет осуществлять поиск по образцу и замену найденного фрагмента на заданный. Предусмотрите подсчет числа замен.

Задание 8. Лишние пробелы

Разработайте сценарий, который сможет удалять идущие подряд несколько пробелов, оставляя только один. Предусмотрите счетчик удалений.

Задание 9. Палиндром

Разработайте сценарий, который сможет из введенного посетителем слова построить слово-перевертыш и определить, совпадает ли слово-перевертыш с исходным словом.

Задание 10. Удаление ненужных слов

Разработайте сценарий, который сможет удалять из текста заданные слова. Предусмотрите подсчет числа удалений.

Задание 11. Заглавная буква

Разработайте сценарий, который переведет в верхний регистр первую букву каждого слова, следующего за точкой.

Задание 12. Разрядка числа

Разработайте сценарий, который в заданное число добавит пробелы так, чтобы они отделяли по 3 разряда справа налево.

Задание 13. Недостающие нули

Разработайте сценарий, который к заданному рациональному числу добавит недостающие нули справа так, чтобы число имело только два знака после десятичной точки (запятой).

Практическая работа №4

Задание 1

Составьте функцию, которая возвращает текущую дату в формате *день:число месяц:название год:число день_недели:название*

27 февраля 2015 пятница

Задание 2

Напишите сценарий, который по введенному дню рождения высчитывает и выводит возраст человека.

Задание 3

Напишите сценарий, который определяет число дней между двумя введенными датами.

Задание 4

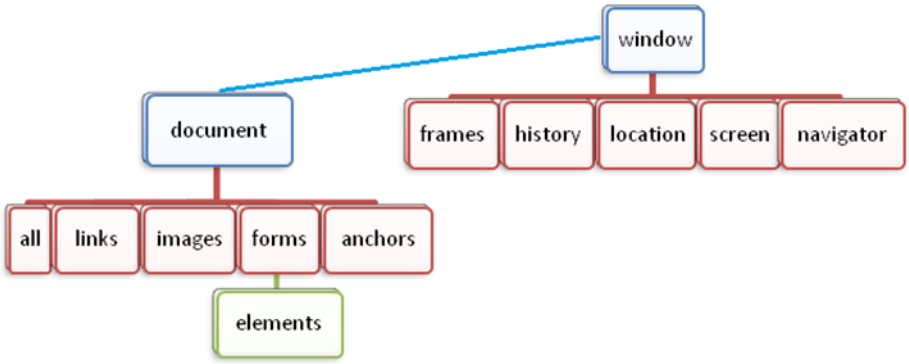
Напишите сценарий, который выводит в таблице календарь на один месяц. Название месяца и год выбирает посетитель из соответствующего размещенного на веб-странице списка. Предусмотрите в сценарии возможность составления посетителем списка выбранных из календаря дат.

Задание 5. Автоматизация разработки сайта

Напишите сценарий для всех страниц сайта, который размещает на каждой из них верхний заголовочный баннер и левую навигационную панель. Баннер для каждой страницы содержит соответствующий заголовок.

6 Объектная модель браузера

Совокупность объектов JavaScript включает две модели: объектную модель браузера **BOM** (Browser Object Model) и объектную модель документа **DOM** (Document Object Model). Модели имеют древовидную структуру. В данном разделе рассмотрим объектную модель браузера.



Объектная модель браузера — это иерархическая система объектов, предназначенных для управления окнами браузера и обеспечения их взаимодействия. Каждое из окон браузера представляется объектом window, центральным объектом BOM. Объектная модель браузера до сих пор не стандартизирована, однако спецификация находится в разработке.

На более низком уровне иерархии объектная модель браузера обеспечивает поддержку следующих возможностей:

- управление фреймами,
- поддержка задержки в исполнении кода и организация периодического вызова фрагмента кода на выполнение,
- диалоги с пользователем,
- управление адресом загрузки браузером веб-страницы,
- доступ к информации о браузере,
- доступ к информации о параметрах монитора,
- ограниченное управление историей просмотра страниц,
- поддержка работы с HTTP cookie.

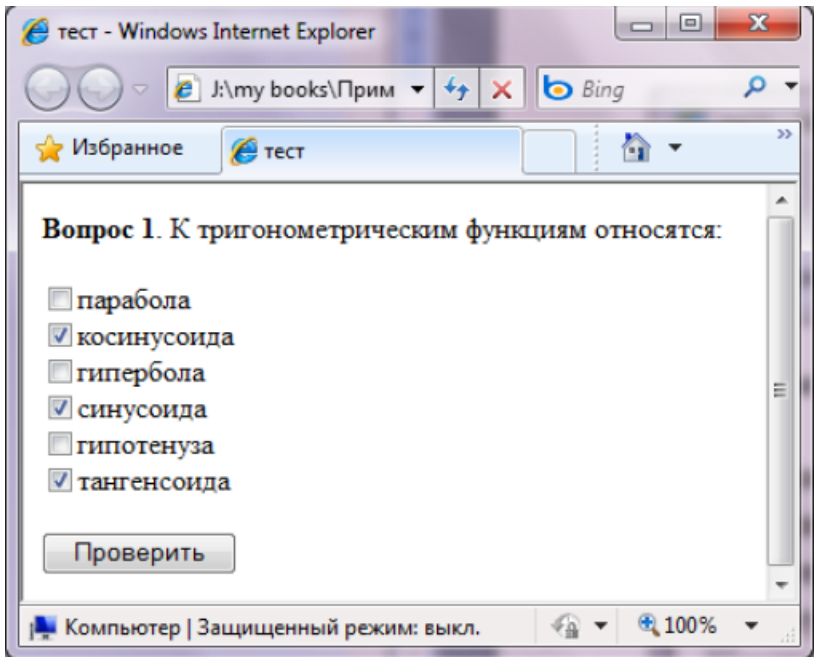
На диаграмме перечислены далеко не все объекты. Более подробную информацию можно найти в литературе, указанной в библиографии. Доступ к свойствам объекта осуществляется через точечную нотацию, которая начинается с вершины иерархии (*window* можно опускать) и заканчивается названием свойства объекта. Свойства объектов сформулированы на языке HTML в виде атрибутов тегов соответствующих объектов.

```
window.document.forms[0].elements[2].checked
```

Пример показывает доступ к свойству checked одного из элементов формы. Каждая форма, расположенная на веб-странице, имеет свой порядковый номер, соответствующий последовательности создания форм (нумерация начинается с 0). Каждый элемент формы также пронумерован в порядке создания. Но если объектам даны имена через атрибут NAME, то вместо имен коллекций с порядковыми номерами можно использовать оригинальные имена.

Рассмотрим *пример:*

```
<html>
<head> <title> тест </title>
<script type="text/JavaScript"> function проверка ()
{
   var doc=document.form1;
   if ((doc.вариант2.checked) && (doc.вариант4.checked) && (doc.вариант6.checked))
      alert ("Правильно") else
      alert ("Неправильно");
}
</script>
</head>
<body>
<form method="post" name=”form1”>
   <p><strong>Вопрос 1</strong>. К тригонометрическим функциям относятся:</p>
   <p>
      <input type = ”checkbox” name = “вариант1”> парабола <br>
      <input type = ”checkbox” name = “вариант2”> косинусоида <br>
      <input type = ”checkbox” name = “вариант3”> гипербола <br>
      <input type = ”checkbox” name = “вариант4”> синусоида <br>
      <input type = ”checkbox” name = “вариант5”> гипотенуза <br>
      <input type = ”checkbox” name = “вариант6”> тангенсоида </p>
   <p>
      <input type=”button” value=”Проверить” onClick=”проверка ()”>
   </p>
</form>
</body>
</html>
```



6.1 Объект Window

Объект *window* — наивысший объект в иерархии JavaScript, представляющий собой открытое окно браузера.

Одним из свойств этого объекта является свойство defaultStatus, отвечающее за хранение стандартного (по умолчанию) сообщения в статусной строке браузера.

$$window.defaultStatus = \alpha$$

где α — какое-либо сообщение.

Свойство *status* позволяет изменять текущее сообщение статусной строки, например то, которое появляется, когда курсор мышки проходит над гиперссылкой.

$$window.status = \alpha$$

где α — какое-либо сообщение.

Свойство, возвращающее url ссылки: *this.href*.

$$window.opener$$

Свойство *opener* возвращает ссылку на объект Window, скрипт которого открыл текущее окно.

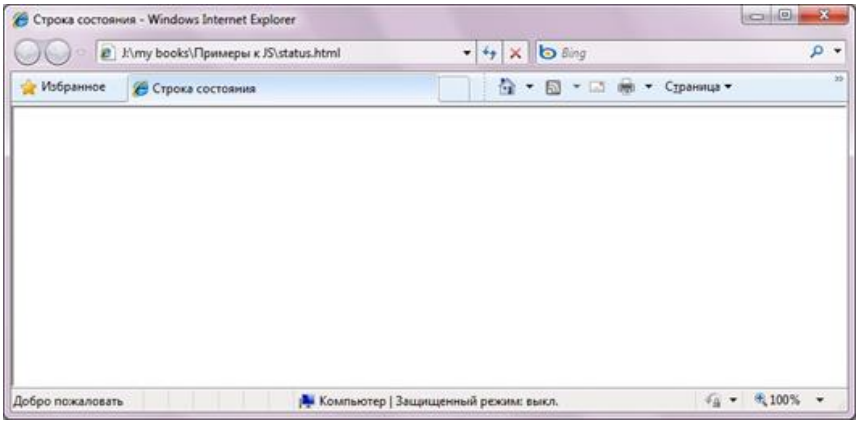
```
document.write (opener.location)
```

window.closed

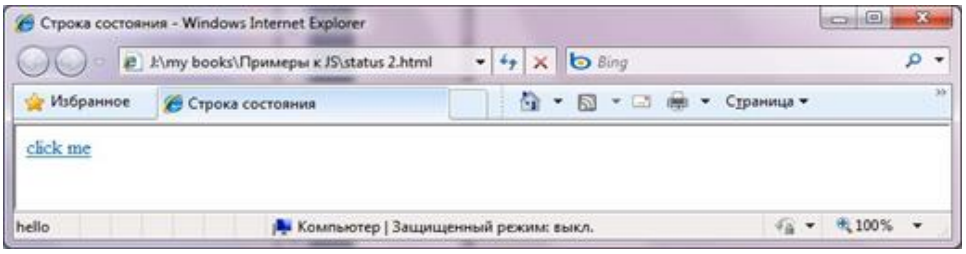
Свойство *closed* возвращает *null*, если окно никогда не открывалось; значение *true*, если окно закрывается; *false* — еще открыто.

Примеры

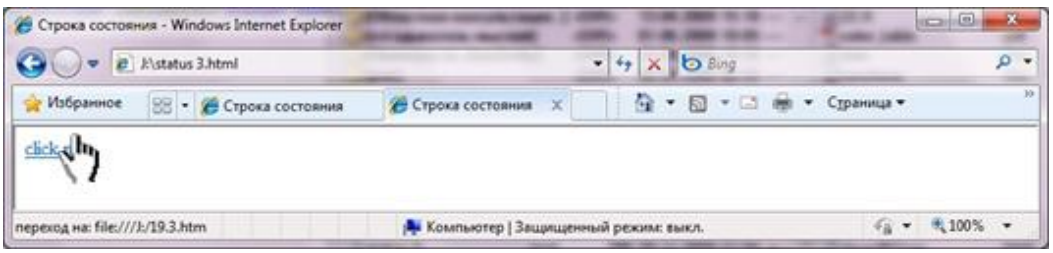
```
<body onLoad="status='Добро пожаловать'">
```




```
<a href="19.3.htm" onMouseOver="status='hello'; return true"
onMouseOut= "status=' '; return true"> click me </a>
```




```
<a href="19.3.htm" onMouseOver="status='переход на: '+this.href; return true"
onMouseOut="status=''; return true"> click me </a>
```



Методы

1. open

window.open ("α"[, "β"] [, "γ"])

где α — указатель ресурса Интернета; β — имя окна; γ — необязательные параметры.

Метод открывает новое окно с именем β и загружает в него документ, расположенный по адресу α.

Если ни один из списка параметров γ не указан, используются значения по умолчанию. Если же указан хотя бы один параметр из списка γ, все неуказанные параметры принимают значение *no*.

Параметр	**Применение**
toolbar	стандартная панель инструментов, включая кнопки Forward, Back, перехода к домашней странице и печати
location	адресная строка
directories	панель стандартных гиперссылок
status	строка состояния
menubar	строка меню
scrollbars	линейки прокрутки
resizable	изменение размера окна
width	ширина окна в пикселях
height	высота окна в пикселях
top	отступ от верхнего края экрана
left	отступ от левого края экрана

Примеры

```
<script type = "text/JavaScript">
function o (f) {
window.open (f, 'tmp', "status=yes, toolbar=yes, location=yes, height=433, width=283")
}
</script>
<A href="javascript: o('w1.html')"> <IMG src="w1small.jpg"> </A>
<A href="javascript: window.open ('19.4.htm?269000.JPG', 'foto', 'height=400, width=300')">
<IMG src="image2.jpg"> </A>
```

2. close

α.close ()

Метод закрывает окно α, где α — имя окна. Например:

```
window.close () self.close () FotoWin.close ()
```

3. setInterval

window.setInterval (α, β, γ, ...)

где α — совокупность операторов или функций, вызываемых периодически через β миллисекунд; γ, ... — необязательные аргументы.

Метод задает периодическое выполнение α через β миллисекунд и возвращает уникальный идентификатор, который можно использовать для отмены его действия.

4. clearInterval

window.clearInterval (ε)

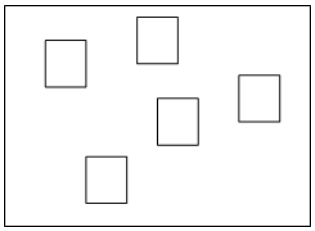
где ε — уникальный идентификатор.

Метод отменяет *setInterval* с указанным идентификатором.

Пример

Скрипт генерирует новые окна с интервалом в 1 с и располагает их случайным образом по экрану.

```
<script type="text/JavaScript"> var число_окон = 5
var интервал_id = setInterval ("открыть_окно ()", 1000) function открыть_окно () {
s = число_окон-- n = 'foto' + s
x = Math.random () * 1000 y = Math.random () * 300
window.open ('', n, 'height=400, width=500, top=' + y + ', left=' + x) if (число_окон == 0)
clearInterval (интервал_id)
}
</script>
```



5. setTimeOut

window.setTimeout (α, β, γ, ...)

где α — совокупность операторов или функций, вызываемых через β миллисекунд; γ, ... — необязательные аргументы.

Метод задает задержку выполнения α на β миллисекунд и возвращает уникальный идентификатор, который можно использовать для отмены его действия. Данный метод можно применять для задания времени ожидания ответа на вопрос теста.

6. clearTimeout

window.clearTimeout (ε)

где ε — уникальный идентификатор.

Метод отменяет *setTimeout* с указанным идентификатором.

7. Методы alert, confirm, prompt предназначены для создания диалоговых окон ввода-вывода информации.

window.alert (α)

Метод выводит значение α в диалоговое окно.

γ = window.prompt (α [,β])

Метод создает окно ввода данных, где α — выводимая в окно информация-приглашение; β — запрашиваемое значение по умолчанию; γ — переменная, в которую передается введенное значение, если нажата кнопка Ok, или служебное значение *null*, если нажата кнопка Cancel.

γ = window.confirm (α)

Метод создает диалоговое окно выбора из двух вариантов — «Да» или «Нет» — и возвращает соответственно значения *true* или *false*, где α — выводимая в окно информация-подсказка.

8. moveTo

window.moveTo (α,β)

Метод перемещает окно в точку с координатами *(α,β)*. Эта координата попадает в левый верхний угол окна.

9. moveBy

window.moveBy (α,β)

Метод перемещает окно на расстояние *α* пикселей по оси *x* и на расстояние *β* пикселей по оси *y*.

10. resizeTo

window.resizeTo (α,β)

Метод изменяет размеры окна до значений *α и β* пикселей по ширине и высоте соответственно.

11. resizeBy

window.resizeBy (α,β)

Метод изменяет размеры окна на значения *α и β* пикселей по ширине и высоте соответственно (приращение окна).

```
<html>
   <script type="text/JavaScript">
      function o () {
         self.resizeTo (100,100)
         for (i=1; i<50; i++) self.resizeBy (i, i)
      }
      </script>
   <body>
      <A href="javascript: o()"> click me! </A>
   </body>
</html>
```

12. external.addFavorite

window.external.addFavorite (α [,β])

Метод помещает название веб-страницы в папку «Избранное» (закладки),

где α — уникальный указатель ресурса Интернета, β — название ресурса.

```
<a href="javascript: window.external.addFavorite (location.href, document.title)">
<img src="images/favorites2.gif" width="125" height="124" border="0"
alt="Добавить в избранное">
</a>
```

6.2 Объект Location

В объекте *location* представлен адрес (URL) загруженного HTML-документа. Существует возможность записывать в *location.href* свои значения. В приведенном ниже примере щелчок по кнопке загружает в текущее окно новую страницу:

```
<form>
<input type=button value="Rambler"
   onClick="location.href='http://www.rambler.ru'; ">
</form>
```

Каждое свойство объекта *location* представляет различную часть URL. Следующий список показывает связи между частями URL и свойствами *location*:

protocol:// hostname:port pathname search hash

где *protocol* - строка, содержащая начальную часть URL, до двоеточия включительно, *hostname* - доменное имя или IP-адрес, *port* - часть URL, содержащая номер порта, *pathname* - часть URL, содержащая путь, *search* - строка, содержащая любую информацию запроса, присоединенную к URL, начинающаяся с вопросительного знака, *hash* - часть URL, начинающаяся с символа (#).

Не путайте объект *location* со свойством *location* объекта *document*, которое нельзя изменять (*document.location*), но можно менять значение свойства объекта *location.*

Пример

Следующие два утверждения эквивалентны и изменяют URL текущего окна:

```
window.location.href = "http://mozilla-russia.org/"
window.location = "http://mozilla-russia.org/"
location.search.substring (1)
```

С помощью свойства *search* можно передавать информацию между веб-страницами. На одной странице строится URL со знаком вопроса и данными, а в принимающей странице эти данные можно извлечь строкой, показанной в предыдущем примере.

Метод *substring* возвращает подстроку из строки.

Методы

reload ([α])

Повторная загрузка страницы, где α — необязательный параметр. Принимает значение False для загрузки из КЭШ, True для загрузки с сервера. Пример: *location.reload (true).*

replace (α)

Замена страницы, где α — указатель на новую страницу.

Пример. Передача данных между страницами.

```
page1.htm
<html>
<body onLoad="location.href='page2.htm?DOG07.GIF'”>
</body>
</html>
page2.htm
<html>
   <script type="text/JavaScript">
      h=window.location.search.substring(1)
      document.write (h + "<BR>")
      document.write ("<img src=" + h + ">")
   </script>
</html>
```

6.3 Объект History

Объект *History* хранит список веб-страниц, посещенных в текущем сеансе работы. Часто такой список называют журналом. Можно выделить три части списка. Первая часть содержит страницы, на которые можно вернуться по щелчку на кнопке «Назад» окна браузера, вторая часть — текущая страница, и третья часть содержит страницы, на которые можно вернуться по щелчку на кнопке «Вперед». Однако если происходит возврат на посещенную страницу, а затем переход на новую, то третья часть журнала очищается.

Методы

window.history.back([α])

Перемещение назад по первой части журнала, где α — число страниц.

window.history.forward([α])

Перемещение вперед по третьей части журнала.

window.history.go(α)

Перемещение на любую страницу в журнале. Если α — отрицательное число, то перемещение идет по первой части журнала, если α — положительное — по третьей части, если $\alpha = 0$, то происходит обновление текущей страницы.

Пример

```
<body>
   <a href="page3.htm"> страница 3 </a>
   <a href="page4.htm"> страница 4 </a>
   <a href="page1.htm"> страница 1 </a>
   <input type=button value=назад onClick="history.back()">
   <input type=button value=вперед onClick = window.history.forward()>
</body>
```

6.4 Объект Screen

Объект *Screen* предназначен для определения разрешения монитора компьютера, в котором выполняется скрипт.

Свойства (только для чтения)

availHeight	высота доступной области экрана
height	высота экрана (в пикселях)
availWidth	ширина доступной области экрана
width	ширина экрана (в пикселях)
colorDepth	глубина цвета (в битах)

6.5 Объект Document

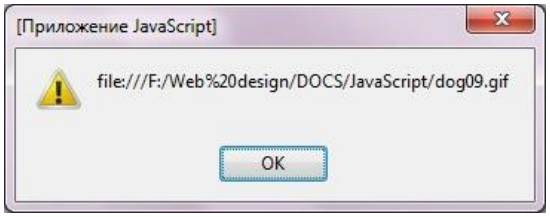
Объект *Document* представляет содержимое веб-документа или фрейма, т. е. это совокупность элементов html-документа. Так как элементов на веб-странице может быть достаточно много, они группируются в массивы и коллекции. Именно через свойства и методы этого объекта можно изменять содержимое html-документа.

Свойства

this - в обработчике события возвращает элемент, вызвавший событие. Слово *Document* как бы включается в возвращаемое значение.

Например:

```
<img src=dog09.gif onClick="alert (this.src)">
```



alinkColor - возвращает цвет активной ссылки

linkColor - возвращает цвет непосещенной ссылки

vlinkColor - возвращает цвет посещенной ссылки bgColor возвращает цвет фона документа fgColor возвращает цвет текста документа

lastModified - возвращает дату последнего изменения документа

location - возвращает адрес документа

referrer - возвращает адрес документа, с которого посетитель перешел к текущей странице. Если текущий документ открылся не по ссылке, а по введенному в браузер адресу, то возвращается пустая строка

title - возвращает заголовок документа

cookie - возвращает набор кукиз

Пример

```
<hr>
<script type = "text/JavaScript">
document.write ("<p align=center> Последнее обновление: <i>", docu- ment.lastModified, "</i>
</p>")
</script>
```

Коллекции

all - коллекция всех тегов и элементов, расположенных в теле html-документа

anchors - коллекция всех тегов <a name>

forms - коллекция всех форм (теги <form>) документа

images - коллекция всех изображений (теги <img>) документа

links - коллекция всех ссылок (теги <a href>) документа

Элементы коллекции, так же как и члены массива, имеют общие имя и порядковый номер (индекс). В отличие от массивов коллекции содержат наборы элементов html-документа. Через коллекции веб-программисты имеют доступ к значениям многих атрибутов тегов html-документа.

Свойства коллекции

Length - возвращает число элементов коллекции

item (α) - возвращает элемент, соответствующий указанному индексу или идентификатору

tags ("α") - возвращает коллекцию, содержащую указанные теги, α — содержимое тега

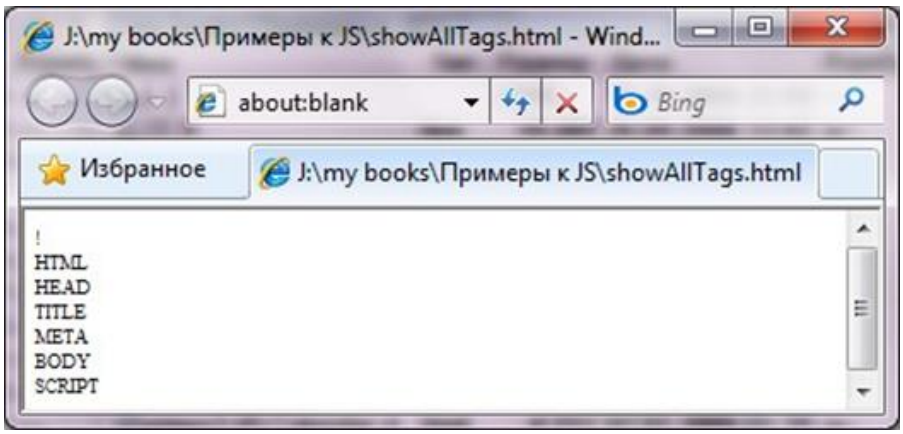
tagName - возвращает содержимое тегов

type - возвращает тип элемента формы

innerHTML - возвращает текст и внутренние теги данного элемента

Примеры

```
document.forms[0].list1.value
document.forms[0].elements[1].type
document.all.item(6).tagName
document.all.tags("p").length
function showAllTags ()
   {//функция вывода всех тегов документа в отдельное окно
      w=window.open ()
      for (i=0; i < document.all.length; i++)
         {w.document.write (document.all(i).tagName, "<br>")}
   }
```



Методы

clear () - очищает документ от текста и дескрипторов

close () - закрывает все выходные потоки, используемые для записи текста в документ

open () - открывает все выходные потоки, позволяющие записывать текст в документ

write ($\alpha, \beta, \gamma, \ldots$) - записывает данные в документ, включая html-теги

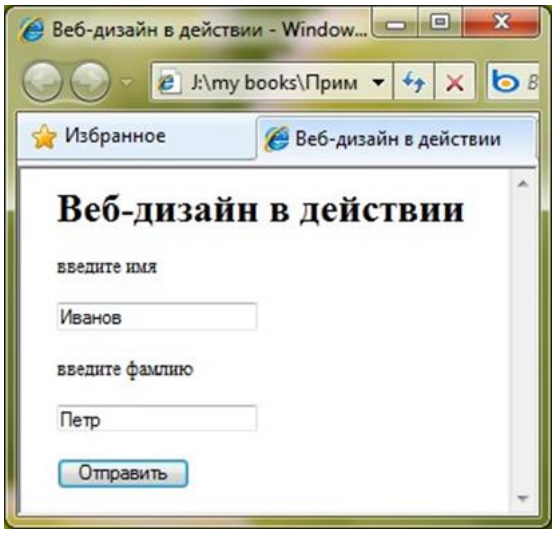
writeln ($\alpha, \beta, \gamma, \ldots$) - записывает данные в документ, включая html-теги, и добавляет в конец вывода символ перехода на новую строку

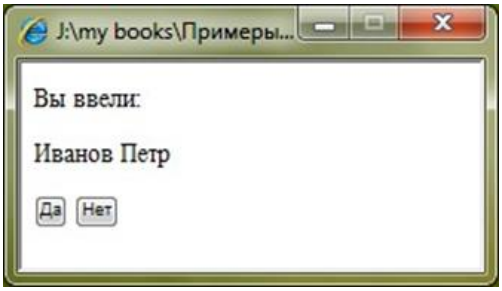
Пример

Заполнение формы с открытием окна-запроса на подтверждение введенных данных.

```
<HTML>
<HEAD>
<TITLE>Веб-дизайн в действии</TITLE>
<SCRIPT type = "text/JavaScript">
var w
function send () {
  fio=document.f.n1.value+" "+document.f.n2.value
  if (!window_available())
    w=open("","k","width=200,height=150")
  else
    w.focus()
w.document.writeln ("<p>Вы ввели:</p>")
w.document.writeln ("<p>",fio,"</p>")
w.document.writeln ("<input type=button value=Да name=b1 on- Click='doSend (this)'>")
w.document.writeln ("<input type=button value=Нет name=b2 on- Click='doSend (this)'>")
w.document.writeln ("<script>")
w.document.writeln ("function doSend(o)")
w.document.writeln ("{if (o.value=='Нет') {alert('Повторите
ввод');opener.document.forms[0].reset();self.close()}")
w.document.writeln("if(o.value=='Да'){opener.close();self.close()}}")
w.document.writeln ("<\/script>")
w.document.close()
};
function window_available() {
  if (!w) {return false}
    else
  if (w.closed) {return false} else {return true}
}
</SCRIPT>
</HEAD>
<BODY leftmargin=30>
<H1>Веб-дизайн в действии</H1>
<FORM METHOD = POST NAME = “f”>
<P>введите имя</P>
<P><INPUT TYPE = text NAME = “n1”></P>
```

```
<P>введите фамлию</P>
<P><INPUT TYPE=text NAME = “n2”></P>
<P><INPUT TYPE = “button” VALUE = "Отправить" onClick = "send()"></P>
</FORM>
</BODY>
</HTML>
```





6.5 Объект Image

Графические изображения на веб-странице являются объектами Image. Все графические объекты на странице составляют коллекцию графических объектов, т. е. они пронумерованы, начиная с 0, и имеют общее имя *Images*. В JavaScript можно через точечную нотацию обращаться к свойствам объекта Image, которые являются атрибутами того же объекта, сформулированными на языке HTML.

```
document.images[0].width = w
document.images[0].src = "new_pic.jpg"
```

В последнем примере происходит смена изображения на новое. При этом новая картинка принимает размеры предыдущей. Смену изображений часто используют при наведении курсора мышки на графическую гиперссылку.

JavaScript поддерживает предварительную загрузку изображений. Это, например, может понадобиться для считывания свойств графического объекта и соответствующего изменения параметров окна, в котором оно отображается.

Для предварительной загрузки рисунков сценария применяется следующий конструктор:

$$var\ \alpha = new\ Image\ ([\beta, \gamma])$$

где α — имя переменной или массива; β, γ — ширина и длина изображения в пикселях.

Пример

Изменение размеров окна под размер загружаемого изображения и перемещение окна в центр экрана.

```
<script type = "text/JavaScript">
  if (location.search)  {
  var fi=location.search.substring (1)
  var w=location.hash.substring (1, 4)
  var h=location.hash.substring (4)
  var ws=screen.availWidth/2 - w/2
  var hs=screen.availHeight/2 - h/2
  document.write ("<img src="+fi+">")
  self.resizeTo (w, h)
  self.resizeBy (30, 100)
  self.moveTo (ws, hs)
  }
</script>
```

6.6 Объект Navigator

Известно, что для отображения веб-страниц предназначены браузеры.

Наиболее популярными «отображателями» веб-страниц являются:

1. Microsoft Internet Explorer (*англ*. Explorer — путешественник, исследователь).
2. Mozilla Firefox (*англ*. Firefox — огненная лиса, в честь которой, по словам разработчиков, и назван браузер).
3. Opera (*англ*. Opera — опера).
4. Apple Safari (*арабск*. Safari — поход, путешествие, турпоездка, длительное путешествие в экзотические страны).
5. Konqueror (Конкерор, от *англ*. Conqueror + KDE — завоеватель, победитель).
6. Google Chrome (*англ*. Chrome — металл хром или цветной фотографический транспарант (постер) с блестящей, сияющей поверхностью).

При разработке браузеров, как и другого программного обеспечения, выделяют основные модули и компонуют их в самостоятельный так называемый *движок*, который затем может быть включен в состав других разрабатываемых программ, решающих подобные задачи.

Браузерные движки, как и браузеры, имеют собственное имя.

Браузер	**Логотип**	**Движок**	**Использование движка**
Microsoft Internet Explorer		Trident	Проводник (Windows Explorer), справка Microsoft Windows, Outlook и Outlook Express, InfoPath, Media Player, Encarta
Mozilla Firefox		Gecko	Почтовый клиент Thunderbird, календарь Sunbird и IRC-клиент Chatzilla, свободный набор программ для работы в Интернете SeaMonkey
Opera		Presto	Adobe GoLive, Dreamweaver
Konqueror		KHTML	Как менеджер файлов в KDE
Apple Safari		WebKit	Adobe Integrated Runtime (AIR) — платформо-независимая среда для запуска приложений, Android — платформа для мобильных телефонов, LeechCraft — кроссплатформенный модульный интернет-клиент с плагином- браузером Poshuku
Google Chrome			

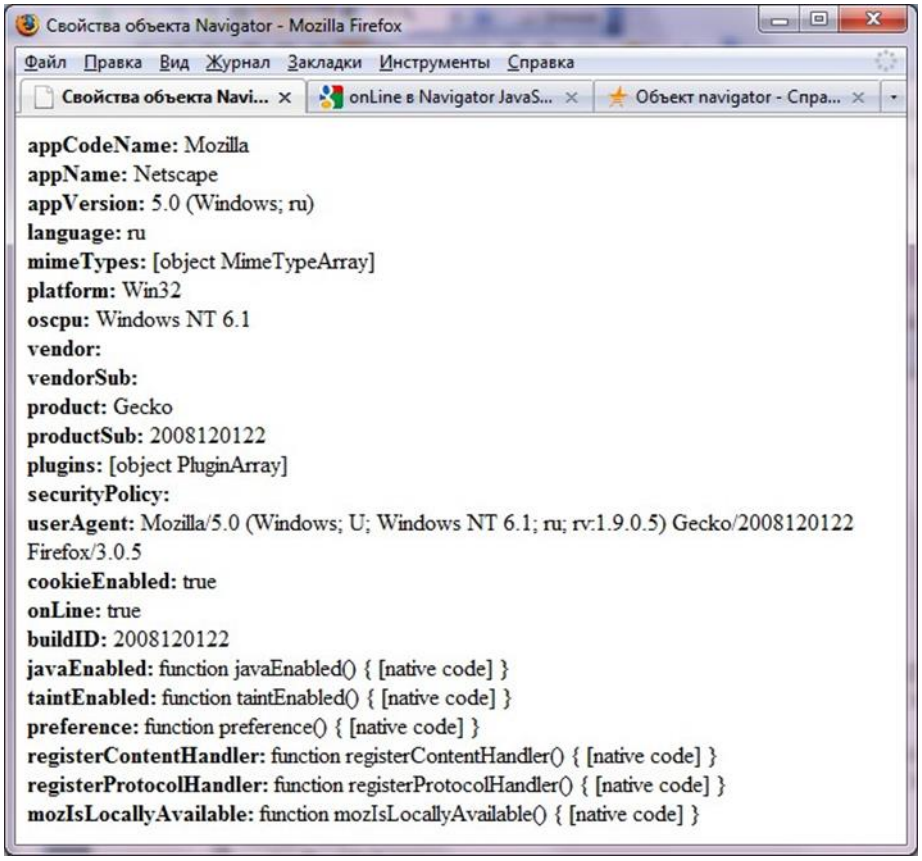
Все браузеры стремятся поддерживать стандарты разработки веб-страниц, такие как HTML, CSS, JavaScript, DOM, форматы отображения веб-графики. Однако в реальности существует различие в поддержке веб-стандартов различными браузерами. Кроме того, каждый движок «понимает» отличные от стандарта методы, «непонятные» другим движкам. Все это несколько усложняет разработку веб-документов и веб-приложений.

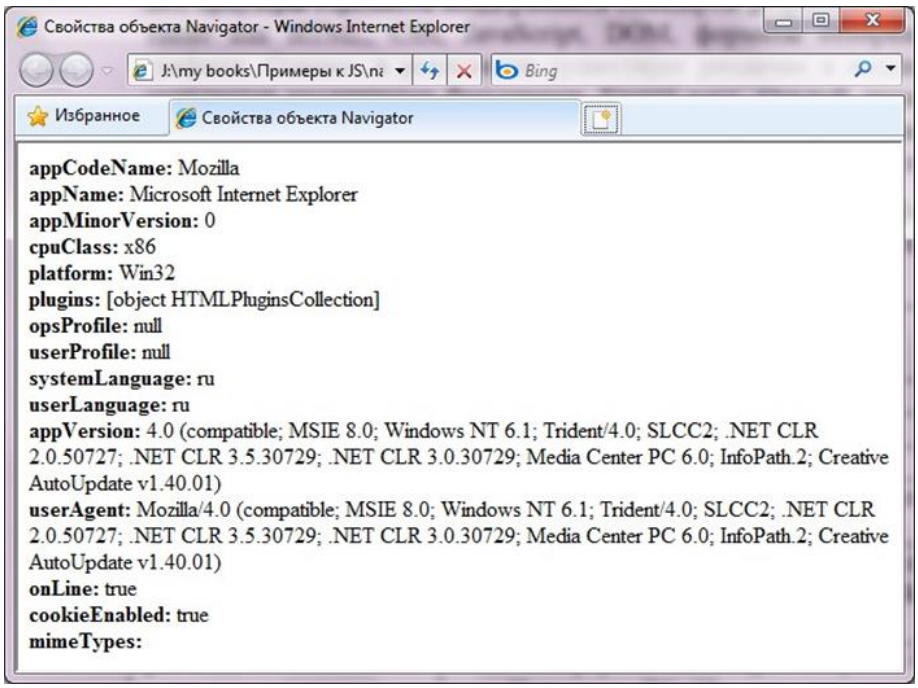
В JavaScript существует объект Navigator, который хранит информацию о браузере посетителя (клиента) текущей веб-страницы. В таблице собраны свойства, общие для нескольких браузеров.

Свойство объекта Navigator	**Возвращаемое значение**
appCodeName	Внутреннее кодовое имя браузера
appName	Название браузера
appVersion	Внутренний номер версии браузера
cookieEnabled	true/false — разрешено/запрещено использовать кукиз
onLine	true/false — установлен браузер для online или для offline просмотра

	(Файл->Работать автономно)
platform	Имя платформы, на которой запущен браузер. Например, Win32
plugins	Коллекция названий всех подключенных к браузеру модулей
userAgent	Строка, содержащая значения свойств appCodeName, appName, appVersion и некоторых других

У каждого браузера помимо перечисленных в таблице есть свой дополнительный набор свойств. На рисунках названия свойств выделены полужирным шрифтом. Скрипт, с помощью которого можно получить список всех свойств любого объекта JavaScript, приведен в разделе Управляющие инструкции.





Например, метод создания модального диалогового окна showModalDialog хорошо работает пока только в Internet Explorer. Поэтому для корректности работы кода и в других браузерах можно использовать стандартный метод open объекта window.

```
var браузер = navigator.appName
if (браузер == "Microsoft Internet Explorer")
  окно_проверка_доступа = window.showModalDialog
   ("tests/PanelCheckAvtor.htm", null, "dialogWidth:20em; dialogHeight:20em, center:1")
else {
  окно_проверка_доступа = window.open ("tests/CheckAvtor.htm", "проверка_доступа",
"width=430, height=300, status=no, resizable=yes, scrollbars=yes, menubar=yes, toolbar=yes,
status=yes")
  окно_проверка_доступа2.focus ()
}
```

Как видно из скриншотов, значение свойства userAgent представляет собой строку с большим числом данных, включающих в себя название браузера и его кодовое имя, версию браузера и название движка, название операционной системы и т. д. Для браузера от Microsoft в качестве названия браузера указано короткое имя MSIE, что раскрывается как MS Internet Explorer. Извлечь нужную информацию из строки можно с помощью перечисленных ранее методов объекта String. Например, следующие строки кода отвечают на вопрос о движке браузера клиента, является ли он Gecko или Trident.

```
Это_Gecko = navigator.userAgent.indexOf ("Gecko")
Это_Trident = navigator.userAgent.indexOf ("Trident")
if (Это_Trident !== -1) alert ("IE")
```

```
if (Это_Gecko! == -1) alert ("Gecko")
```

Для написания корректного кода разработчики применяют еще и непосредственную проверку поддержки используемого в скрипте объекта и его свойств и методов. Например, проверка поддержки объекта all показывает, что в Firefox он не распознается, в отличие от MSIE и Google Chrome. Объект navigator знаком всем этим браузерам.

```
if (document.all) alert ("all работает")
if (navigator) alert ("navigator работает")
```

Проверку свойства или метода объекта веб-дизайнеры рекомендуют проверять вместе с существованием самого объекта, чтобы условие могло выполниться без ошибки. Если объект не функционирует на данном движке, то первый аргумент сложного условия даст значение false, и второй аргумент проверяться не будет.

Например, следующей строкой кода можно проверить поддержку свойства appMinorVersion объекта navigator и убедиться в том, что браузер Firefox его не поддерживает, а MSIE поддерживает.

```
if (navigator && navigator.appMinorVersion)
  alert ("appMinorVersion в работе")
```

Следующая строка проверяет функционирование свойства product того же объекта navigator. Проверка показывает, что браузер Firefox его поддержи- вает, а MSIE нет.

```
if (navigator && navigator.product) alert ("product в работе")
```

6.7 Кукиз (cookies)

Cookies (кукиз, в переводе с *англ*. домашнее печенье, булочка) — механизм, позволяющий серверу сохранять на жестком диске клиента небольшой объем информации и использовать его при повторном посещении страницы.

Информация сохраняется в одноименной папке (Internet Explorer) в текстовом файле. Каждая веб-страница записывает данные в свой файл. В большинстве случаев имя cookie-файла строится как $\alpha@\beta.txt$, где α — имя пользователя (регистрационного профиля), β — имя домена соответствующей веб-страницы. Cookie представляют собой последовательность пар *переменная*

= значение, разделенных точкой с запятой.

Cookie позволяют сохранять необходимую информацию на компьютере клиента, например идентификационные данные, настройки страницы, статистическую информацию. Однако у этого механизма есть существенные недостатки. К ним относятся:

- однопользовательская поддержка,
- программная несовместимость (разные браузеры по-разному реализуют эту технологию),
- аппаратная несовместимость (невозможность использования одних и тех же кукиз для нескольких клиентских компьютеров, так как хранятся они не на стороне сервера);
- возможность со стороны клиента отключения механизма cookies,
- возможность удаления файла, хранящего кукиз.

Записывать куки в файл можно через свойство *cookie* объекта *document*.

document.cookie = "α [; expires=β; path=γ; domain=δ; secure]"

где α — поле, содержащее данные в виде пары *имя = значение*; β — срок годности, т. е. момент истечения действия куков по гринвичскому меридиональному времени (например, *Monday, 19-May-2011 23:45:00 GMT*). Если этот параметр не установлен, то куки утратят годность сразу же по окончании сеанса работы браузера; γ— строка, задающая папку верхнего уровня, документы котоhой имеют доступ к кукиз. По умолчанию кукиз доступна любой странице узла, расположенной в той же папке (и любой вложенной в нее), что и исходная. Например, *path=/stuff*; δ — строка, задающая домен, которому разрешен доступ к кукиз. По умолчанию имя домена есть имя сервера, сгенерировавшего кукиз. Этот параметр часто используется, когда на одном домене находится несколько веб-узлов. Чтобы, например, разрешить доступ к кукам всех узлов домена *sura*, нужно составить следующее поле: *domain=.sura.ru*; *secure* — уровень безопасности. Если параметр установлен, то данные передаются по протоколу HTTPS (HTTP-SSL — Secure Socket Level).

Примеры

```
document.cookie = "fileN=" +
  document.forms[0].elements[1].value
document.cookie = ”test=comp1”
document.cookie = document.f.n1.name + "=" +
  document.f.n1.value + "; expires=Monday, 19-May-2011 23:45:00 GMT"
```

Прочитать кукиз можно через то же самое свойство *cookie*.

¶

α = *document.cookie*

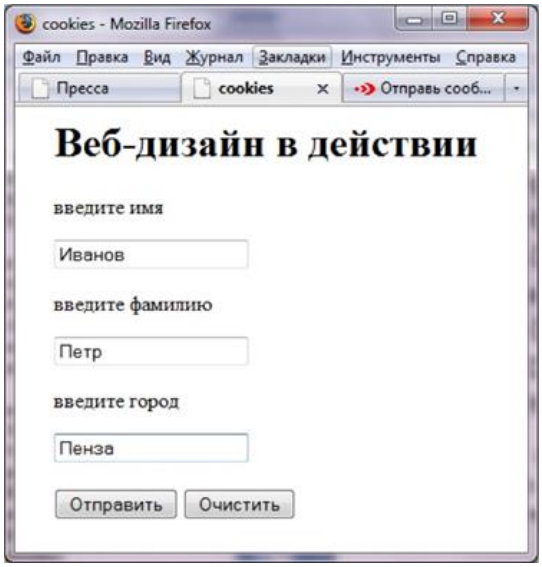
где α — имя переменной. В α записываются все пары *имя = значение*, установленные сервером для данной страницы. Для того чтобы прочитать значение какого-либо параметра кукиз, необходимо писать код JavaScript. В приведенном ниже примере функция set_cookie() записывает в кукиз все значения элементов формы, а функция get_cookie(n) позволяет считывать кукиз и выделять из всех полей значение заданного в аргументе параметра. Функция check_cookie() проверяет существование первого поля кукиз и при положительном результате сама заполняет хранящимися значениями соответствующие области формы при повторной загрузке веб-страницы.

```
<HTML>
<HEAD>
<TITLE>cookies</TITLE>
<script type="text/JavaScript">
  function сохранить_cookie() {
  document.cookie = document.регистрация.n1.name + "=" +
  document.регистрация.n1.value + "; expires=Monday, 19-May-2011 23:45:00 GMT"
  document.cookie=document.регистрация.n2.name + "=" +
  document.регистрация.n2.value + "; expires = Monday, 19-May-2011 23:45:00 GMT"
  document.cookie=document.регистрация.n3.name + "=" +
  document.регистрация.n3.value + "; expires=Monday, 19-May-2011 23:45:00 GMT"
  }
function прочитать_cookie (n) {
  cookie_array=document.cookie.split ("; ")
  for (i=0; i<cookie_array.length; i++)
    {pole=cookie_array[i].split ("=")
      cookie_name=pole[0]
      cookie_value=pole[1]
      if (n==cookie_name) return (cookie_value)}
  return null
}
function проверить_cookie ()
  n1= прочитать_cookie ("n1")
  n2= прочитать_cookie ("n2")
  n3= прочитать_cookie ("n3")
if (n1) {
  document.регистрация.n1.value=n1
  document.регистрация.n2.value=n2
  document.регистрация.n3.value=n3}
}
</SCRIPT> </HEAD>
<BODY leftmargin=30 onLoad="проверить_cookie()">
  <H1> Веб-дизайн в действии </H1>
  <FORM METHOD=POST NAME="регистрация">
    <P>введите имя</P>
    <INPUT TYPE = ”text” NAME =
```

```
    <P>введите фамилию</P>
    <INPUT TYPE = ”text” NAME =
    <P>введите город</P>
    <INPUT TYPE=”text” NAME=”n3”>
    <BR> <BR>
    <INPUT TYPE = button VALUE = "Отправить"
    onClick="сохранить_cookie ()">
    <INPUT TYPE = ”reset” VALUE = "Очистить">
  </FORM>
</BODY>
</HTML>
```

Метод *split*, используемый в функции *прочитать_cookie()*, разбивает строку на части и сохраняет их в массиве. Разделителем для первого массива выступает точка с запятой, для второго — знак равенства.

¶



6.8 Классы

Из Википедии следует, что класс — это элемент программного обеспечения, описывающий абстрактный тип данных и его частичную или полную реализацию. В JavaScript классы, как и функции, можно описывать как классы-декларации (class declarations) или как классы-выражения (class expressions).

Классы в JavaScript появились с внедрением стандарта ES2015 (ES6). Таким образом появился новый способ определения совокупности объектов — с помощью класса. Класс представляет собой описание абстрактного объекта, его состояния и поведения, а объект является конкретным воплощением или экземпляром класса. На основе класса можно создать множество похожих объектов.

```
class α [extends β]  {
   γ
   δ
   get  α1 () { }
   set  α2 () { }
}
```

где α — название класса; β — родитель; γ — конструктор. Конструктор определяется с помощью метода с именем constructor. Это метод, который может принимать параметры. Основная цель конструктора — инициализировать объект начальными данными. Функция constructor запускается при создании объекта — экземпляра класса (new α); δ — методы. Все методы класса работают в строгом режиме (strict), даже если он не указан. Это значит, что все переменные требуют объявления (var, let, const); *get* — методы, возвращающие значения; *set* — методы, устанавливающие новые значения.

Например:

```
class Student {}
```

или в варианте класса-выражения

```
let Student = class{}
```

Добавим в класс Student конструктор и методы.

```
class Student {
  constructor (имя, фамилия, годПоступления, специальность, общежитие,сессия) {
    this.имя=имя
    this.фамилия=фамилия
    this.годПоступления=годПоступления
    this.специальность=специальность
    this.общежитие=общежитие
    this.оценкаСессии=[]
    }
    курс() {
      let сегодня=new Date()
      let текущийГод=сегодня.getFullYear()
      let курс=текущийГод-this.годПоступления
      return (`${курс} курс`)
    }
    get полноеИмя() {
    return `${this.имя} ${this.фамилия}`
  }
  set сменитьИмя(newValue) {
    this.фамилия = newValue
  }
```

```
  set выселитьИзОбщежития(newValue) {
    this.общежитие = newValue
  }
  set результатыСессии(newValue) {
    this.оценкаСессии.push(newValue) // добавить оценку
  }
}
```

Создадим объект stud1 на основе класса Student и дадим значения встроенным свойствам, воспользуемся встроенными методами.

```
stud1=new Student("Дина","Дымова",2017,"физика","есть")
alert (`${stud1.фамилия}
${stud1.курс()}
${stud1.полноеИмя}
${stud1.общежитие}`)
stud1.сменитьИмя = "Петрова"
stud1.выселитьИзОбщежития="нет"
alert(`${stud1.полноеИмя}
${stud1.общежитие}`)
stud1.результатыСессии=5
stud1.результатыСессии=4
alert (`сессия ${stud1.оценкаСессии}`)
```

Как видно из примера, обычные методы вызываются по имени с круглыми скобками, тогда как гетеры и сетеры вызываются без скобок.

¶Если создавать класс, опираясь на родительский (extends), то нужно иметь в виду, что конструктор родителя наследуется автоматически, т. е. его можно опустить в потомке. Если потомку необходим свой конструктор, то для вызова в нем конструктора родителя используется ключевое слово super() с аргументами для родительского класса. Ключевое слово super необходимо указывать до обращения к this. Кроме того, можно через super вызывать родительские методы (super.метод()) в методах потомка.

```
class Город {
  constructor (имя, площадь, население) {
    this.имя=имя
    this.площадь=площадь
    this.население=население
  }
  плотность() {
    let pl=this.население/this.площадь
    return (`${this.имя} ${Math.round(pl)} чел./км^2`)
  }
}
class Поселок extends Город { }
let пос1 = new Поселок ("Лопуховка",50,167)
```

```
console.log (пос1.имя) // Лопуховка
let город1 = new Город ("Пенза", 290, 523553)
console.log (город1.плотность()) // Пенза 1805 чел./км^2
console.log (пос1.плотность()) // Лопуховка 3 чел./км^2
```

В примере выше создан класс Город, и на его основе создан класс-потомок Поселок. Тело потомка пустое, но он унаследовал и конструктор и метод родителя, поэтому мы можем узнать плотность населения не только города Пензы, но и поселка Лопуховка.

Задание 1

Добавьте в описание класса метод, вычисляющий средний балл всех сессий.

Задание 2

Создайте класс «Студенческая группа», в котором опишите параметры: название группы, староста, студенты. Студентов включите как массив объектов класса Student. Создайте методы по смене старосты группы и определению успеваемости группы в целом исходя из успеваемости каждого студента.

Задание 3

Создайте класс «Автомобиль», в котором опишите параметры: марка, цвет, мощность (в л/с), максимальная скорость (км/ч), объем бака (л); расход топлива (л) на 100 км пути. Создайте метод расчета длины пути на полный бак топлива.

Практическая работа №5

Задание 1. Ролловер

Создайте на веб-странице графическую гиперссылку и с помощью скрипта JavaScript сделайте ее интерактивной, т.е. когда мышиный курсор будет проходить над ней и при щелчке мыши, изображение должно меняться.

Задание 2. Список ролловеров

Создайте на веб-странице список ролловеров (список ссылок), при наведении курсора мышки на пункт которого он начинает подсвечиваться, или слева от пункта появляется стрелка.

ГЛАВНАЯ
→ КАРТА
САЙТА
СПИСОК
ССЫЛОК
КОНТАКТЫ

Задание 3. Слайдер 1

Составьте список ссылок (либо с помощью тега <OL>/<UL>, либо с помощью соответствующих тегов формы) на рисунки каких-либо предметов, по щелчку на которые на этой же веб-странице происходит смена текущего рисунка на выбранный.

- Яблоко
- груша
- слива
- банан
- ананас

Задание 4. Слайдер 2

Изучите способы организации массивов в JavaScript. Создайте массив из 10 изображений и реализуйте их последовательный просмотр на веб-странице с помощью гиперссылок «вперед», «назад».

вперёд назад

Задание 5. Фотогалерея

С помощью технологии CSS наложите несколько фотографий друг на друга и поместите название альбома. Напишите скрипт, который при

перемещении по фотографии будет менять фото.

Задание 6. Фотоальбом

С помощью CSS обрамите фотографии альбома в красивую графическую рамку. Напишите скрипт, который выведет на веб-страницу в несколько строк и столбцов обрамленные в красивую рамку фотографии.

Задание 7. Слайдшоу

Напишите скрипт, который будет организовывать просмотр фотографий в виде слайдшоу. В верхней части веб-страницу расположите прокрутку миниатюр, в основной части — оригинальный (или больший) размер выбранной фотографии. С помощью CSS сделайте выделение выбранной фотографии.

Задание 8. Плавный переход

Напишите скрипт, который по какому-либо событию растворяет имеющееся на веб-странице изображение и снова его проявляет. Добавьте на страницу регулятор скорости проявления/растворения.

¶

Практическая работа №6

Задание 1. Бегущая строка

Создайте бегущую надпись в статусной строке браузера.

Задание 2. Тест

Создайте веб-страничку с тестом на любую тему и напишите скрипт на JavaScript для проверки правильности ответов или интерпретации результатов. Вопросы в тесте должны быть разного типа: на выбор одного варианта из нескольких, на выбор нескольких вариантов из множества, ответ одним словом или числом и т. д.

Задание 3. Кукиз

Изучите возможности JavaScript по работе с cookies. Напишите сценарий, который отслеживает число посещений веб-страницы отдельным пользователем и выводит эту информацию на экран.

Задание 4. This

В веб-страницы, созданные ранее, добавьте сценарий JavaScript, выводящий в диалоговом окне тег элемента, по которому щелкает посетитель. Используйте ключевое слово this.

Задание 5. Нестандартная гиперссылка

Сделайте несколько ячеек таблицы (фон ячейки плюс содержимое) гиперссылками. Используйте свойство location объекта document и свойство стилей padding.

Задание 6

Для готовой веб-страницы составьте сценарий JavaScript, выводящий в отдельное окно теги всех заголовков html-документа.

Задание 7. Контроль доступа

Организуйте контролируемый доступ к некоторым страницам сайта. Для этого создайте диалоговое окно для ввода пароля и логина и проверки введенных данных.

Задание 8

Составьте сценарий на JavaScript, выводящий в диалоговом окне текст и внутренние теги элемента, по которому щелкает мышкой посетитель веб-страницы.

Задание 9. Анимированная кнопка

Создайте сценарий, который меняет размер размещенной на веб-странице кнопки за счет вывода на нее надписи путем прибавления или удаления по одной букве.

¶

Задание 10. Выбрать/сбросить всё

Создайте сценарий, который при нажатии на соответствующую кнопку поставит флажки или сбросит их во всей совокупности чекбоксов.

Задание 11. Переливающаяся кнопка

Создайте сценарий, который плавно меняет цвет размещенной на веб-странице кнопки.

Задание 12. Сумасшедшее окно

Создайте сценарий, который создает новое окно и перемещает его по рабочему столу таким образом, как будто его трясет.

Задание 13. Шпион

Разработайте скрипт, который сможет собрать следующую информацию о клиенте: тип операционной системы, название браузера и его версию, доступность cookie, разрешающую способность и глубину цвета монитора.

¶

7 Document Object Model (DOM)

7.1 Общие сведения о DOM

DOM раскрывается как Document Object Model, что в переводе на русский означает Объектная модель документа. Данная модель представляет собой стандарт, одобренный консорциумом Всемирной паутины W3C, и может поддерживаться различными технологиями веб-дизайна, в том числе и JavaScript.

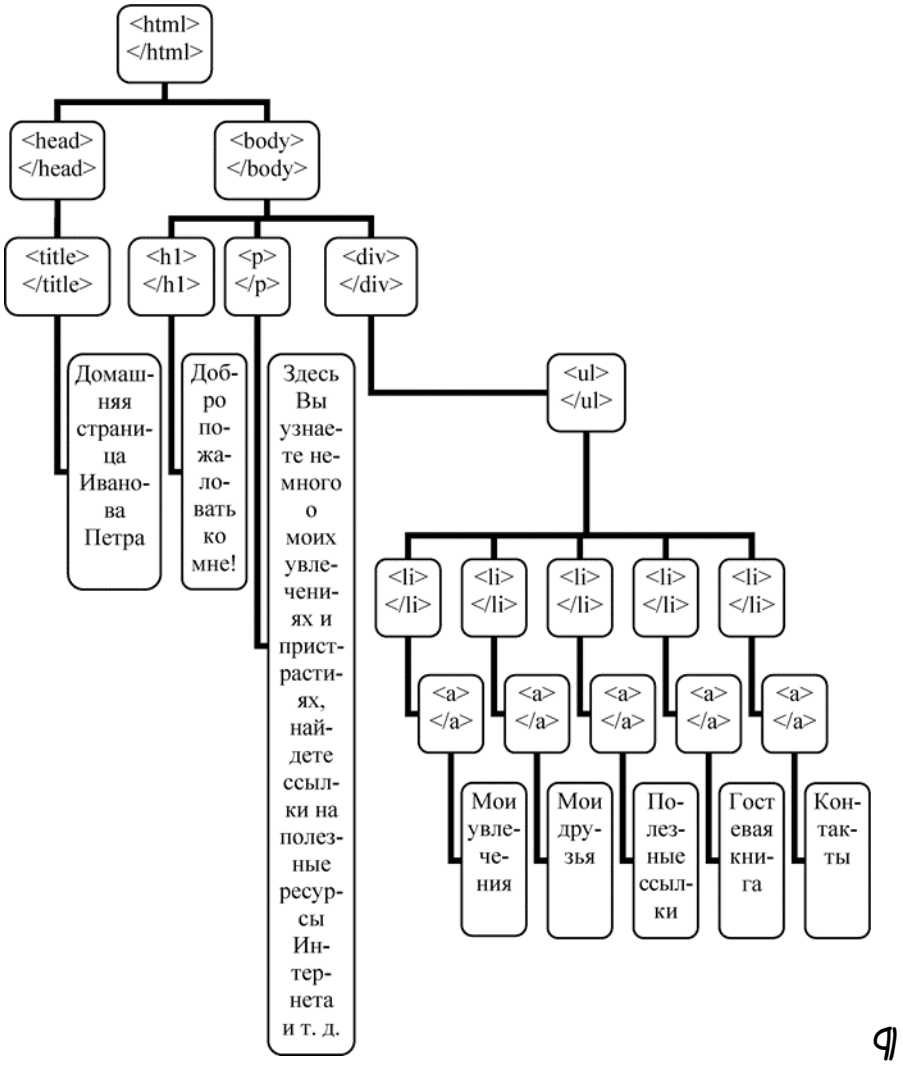
DOM представляет собой иерархическую структуру узлов — объектов html-документа. Различают следующие типы узлов:

- *Элемент* — это узел, представляющий собой html-тег.
- *Текстовый узел* (на *англ*. text node) — это текст, содержащийся между открывающимся и закрывающимся html-тегом.
- *Родительский узел* (на *англ*. parent node) — html-тег, содержащий другие теги или текстовые узлы.
- *Дочерний узел* (на *англ*. child node) — html-тег, входящий в другой тег или текстовый узел. Последний всегда является только дочерним. В иерархии узлов дочерний узел находится на более низком уровне, чем родительский.
- *Сестринский узел* (на *англ*. sibling node) — html-тег или текстовый узел, находящий на одном уровне иерархии с каким-либо другим узлом. Построим объектную модель для следующего html-документа.

Пример 1.1

```
<html>
  <head>
    <title> Домашняя страница Иванова Петра </title>
  </head>
  <body>
    <h1>Добро пожаловать ко мне!</h1>
    <p>Здесь Вы узнаете немного о моих увлечениях и пристрастиях, найдете ссылки на полезные ресурсы Интернета и так далее</p>
    <div id = "navBar">
      <ul>
      <li> <a href="hobbis.html"> Мои увлечения </a>
      <li> <a href="friends.html"> Мои друзья </a>
      <li> <a href="resourses.html"> Полезные ссылки </a>
      <li> <a href="guestbook.html"> Гостевая книга </a>
      <li> <a href="callme.html"> Контакты </a>
    </ul>
  </div>
  </body>
</html>
```

¶


<html> </html>
<head> </head>
<body> </body>
<title> </title>
<h1> </h1>

Домашняя страница Иванова Петра
Добро пожаловать ко мне!
Здесь Вы узнаете немного о моих увлечениях и пристрастиях, найдете ссылки на полезные ресурсы Интернета и т. д.

<a>
<a>
<a>
<a>
<a>
Мои увлечения
Мои друзья
Полезные ссылки
Гостевая книга
Контакты

Теги <head></head> и <body></body> являются дочерними узлами для контейнера <html></html>, а между собой являются сестринскими узлами. Теги <title></title> — дочерние для <head></head>. Контейнеры <h1></h1>, , сестринские между собой и дочерние относительно узла <body></body>. Узел дочерний для и не является сестринским никакому узлу и т. д.

¶

7.2 Узлы

Самый верхний узел dom-дерева (узел <HTML>) можно прочитать свойством *documentElement* объекта *document*.

document.documentElement

Узел <BODY> доступен через одноименное свойство объекта *document*.

document.body

Однако к узлам, отличным от <HTML> и <BODY>, обращение происходит через универсальное свойство *childNodes*, которое возвращает ссылку на дочерний элемент текущего узла. Так как дочерних элементов может быть несколько, то свойство *childNodes* представляет собой массив. И, как и для любого массива, для него предусмотрено свойство *length*, хранящее число элементов массива.

childNodes [α]

где α — номер дочернего элемента (нумерация осуществляется с нуля).

Добавим в приведенный пример в конец html-кода скрипт, использующий *childNodes*.

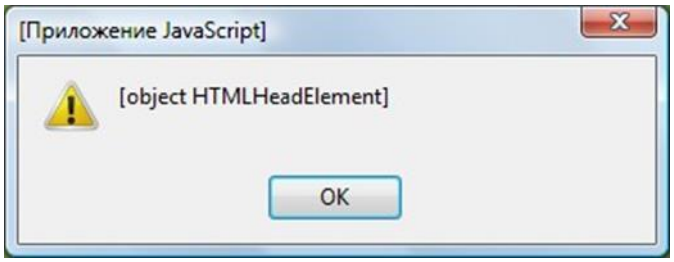
```
<script type = "text/javascript">
alert (document.childNodes[0].childNodes[0])
</script>
```

Для ссылки на элемент <HTML> вместо фразы

```
document.documentElement
```

можно воспользоваться фразой

```
document.childNodes[0].
```



Как видно из рисунка, браузер *Mozilla Firefox* возвращает ссылку на узел <head></head>, так как он является дочерним узлу <html></html>, который, в свою очередь, является дочерним для объекта *document*.

Браузер *Internet Explorer* выдает окно с несколько иным содержанием.

Как видно из рисунка, *Explorer* выдает ссылку на объект, не уточняя его при этом.

В DOM существует метод *hasChildNodes*, определяющий наличие дочерних узлов у текущего элемента α.

$$\alpha.hasChildNodes\ ()$$

В случае, если у тестируемого узла α есть дочерний узел, то метод возвращает значение *true*, в противном случае — *false*. Например, если добавить в предыдущий скрипт еще две следующие строки кода JavaScript, то браузер выдаст диалоговое окно со словом *true*.

```
html_узел = document.childNodes [0]
alert (html_узел.hasChildNodes ())
```

Кроме *childNodes* для ссылки на дочерние узлы в *JavaScript* предусмотрены еще два свойства — *firstChild* и *lastChild*. Первое свойство возвращает ссылку на первый узел в массиве дочерних узлов, а второе на последний. Так, например, следующая строка ссылается на узел <head></head>.

```
document.childNodes[0].firstChild
```

$$\alpha.firstChild$$
$$\alpha.lastChild$$

где α — ссылка на узел.

Для получения подробных данных об узле предусмотрены свойства *nodeName, nodeType* и *nodeValue,* первое из которых возвращает в JavaScript либо название тега, либо значение *#text* в случае текстового узла. Свойство *nodeValue* возвращает значение текстового узла, а в случае узлов-тегов дает *null*. Свойство *nodeType* определяет тип проверяемого элемента. Если элемент α — тег, то *nodeType* возвращает 1, если α — текстовый узел, то дает 3 (в модели DOM имеется 12 типов элементов, но в JavaScript поддерживаются только два). Свойство *tagName*, так же, как и свойство *nodeName*, возвращает имя тега, если α является узлом, но в противном случае — *undefined*.

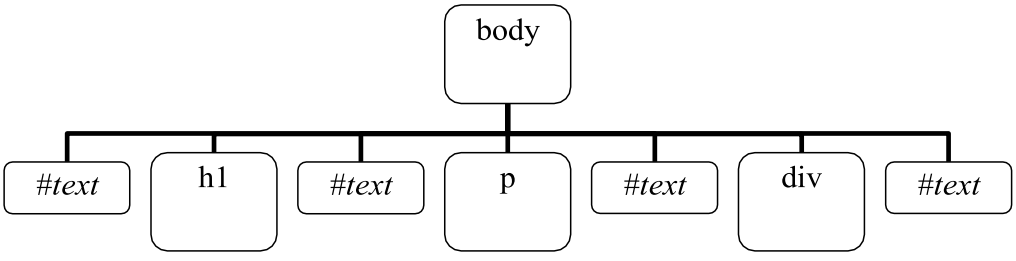
α*.nodeName*
α*.nodeValue*
α*.nodeType*
α*.tagName*

где α — ссылка на узел.

Атрибуты тоже считаются узлами в dom-модели, родителем которых является элемент DOM, у которого они указаны. Однако на практике атрибуты так и используют как свойства узла, значения которых можно устанавливать и изменять.

Если мы проверим число дочерних узлов у <BODY> для вышеприведенного html-документа (пример 1.1), то обнаружим, что в браузере *Internet Explorer* будет выдано значение 3, а в *Mozilla Firefox* — значение 7. Дальнейшее исследование показывает, что *Firefox* «видит» между написанными нами тегами еще и текстовые узлы.

```
function покажи_узлы () {
for (i=0; i<=document.body.childNodes.length; i++)
   alert (document.body.childNodes[i].nodeName)
}
```



Изменим скрипт для примера 1 на следующий.

```
<script type = "text/javascript">
  html_узел=document.childNodes[0]
  body_узел=html_узел.childNodes[1]
  p_узел=body_узел.childNodes[1]
  alert (p_узел.nodeName)
</script>
```

На рисунке видно, что в результате работы скрипта появилось диалоговое окно с названием узла *p*. К сожалению, по непонятным причинам *Mozilla Firefox* вместо узла *p* выдает ссылку на узел *h1*. К счастью, в *DOM* есть еще свойство *getElementById* объекта *document*, которое возвращает ссылку на узел по идентификатору узла, т. е. по значению атрибута *id*, присущему любому html-тегу. Это свойство работает одинаково в обоих браузерах и очень удобно при разработке проекта.

getElementById (α)

где α — значение атрибута *id* html-тега.

Например, ссылка на узел *div* с атрибутом *navBar* в предыдущем примере будет выглядеть следующим образом:

```
document.getElementById ("navBar")
```

Ссылаться можно сразу и на целый массив однотипных элементов с помощью свойства *getElementsByTagName*. Последовательность элементов массива выстраивается в соответствии с расположением элементов на веб-странице (слева направо, сверху вниз).

getElementsByTagName (α)

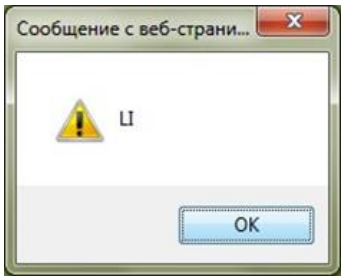
где α — имя элемента.

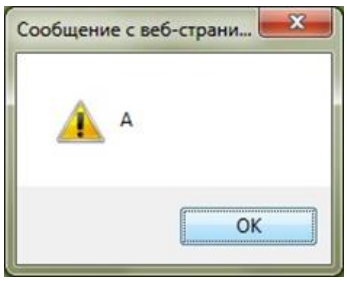
α.getElementsByTagName ('')*

Метод *getElementsByTagName ('*')* возвращает массив всех детей узла α в порядке их обхода. Примеры скриптов для вышеприведенного html-документа:

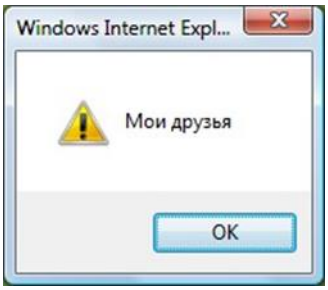
```
<script type = "text/javascript">
  узел_список=document.getElementById ("список")
  массив=узел_список.getElementsByTagName ('*')
  for (var i=0; i<массив.length; i++) alert (массив[i].nodeName)
</script>
```

Данный скрипт выводит диалоговые окна с тегами LI и A столько раз, сколько пунктов содержит список.






```
¶<script type = "text/javascript">
  lis=document.getElementsByTagName ("li")
  alert (lis[1].childNodes[0].childNodes[0].nodeValue)
</script>
```



Переменной *lis* присваивается массив узлов . Во второй строчке скрипта формируется обращение к значению текстового узла, прародителем которого является второй узел в массиве *lis.*

Свойство *getElementsByName* возвращает все элементы, у которых имя (атрибут *name*) равно α.

getElementsByName (α)

где α — строка.

В DOM можно обращаться не только к элементам-потомкам, но и к

элементам-родителям. Свойство *parentNode* существует для ссылки на родительский узел текущего элемента.

$$\alpha.parentNode$$

где α — ссылка на узел.

На сестринский узел можно ссылаться через родительский. Сестринским называется узел, находящийся на одном уровне иерархической структуры документа. Следующий скрипт в примере иллюстрирует организацию ссылок на все пункты списка, являющиеся сестринскими узлами.

Пример 1.2

```
<html>
<head><title>домашняя страница Иванова Петра</title></head>
<body>
<h1>Добро пожаловать ко мне!</h1>
<p>Здесь Вы узнаете немного о моих увлечениях и пристрастиях, найдете ссылки на
полезные ресурсы Интернета и так далее</p>
<div id="navBar">
  <ul id="список">
    <li><a href="hobbis.html">Мои увлечения</a>
    <li><a href="friends.html">Мои друзья</a>
    <li><a href="resourses.html">Полезные ссылки</a>
    <li><a href="guestbook.html">Гостевая книга</a>
    <li><a href="callme.html">Контактная информация</a>
  </ul>
</div>
<script type = "text/javascript">
  узел_список = document.getElementById ("список")
  var пункт_списка = new Array ()
  пункт_списка [1] = узел_список.firstChild
  alert (пункт_списка[1].nodeName)
  for (i=2; i<=5; i++) {
  пункт_списка[i]=пункт_списка[i-1].parentNode.childNodes[i-1]
  alert (пункт_списка[i].nodeName)
  }
</script>
</body>
</html>
```

Однако приведенный пример правильно работает в браузере Internet Explorer и неправильно в Mozilla Firefox.

Для сокращенных ссылок на сестринские узлы в DOM существуют еще два свойства.

α.previousSibling

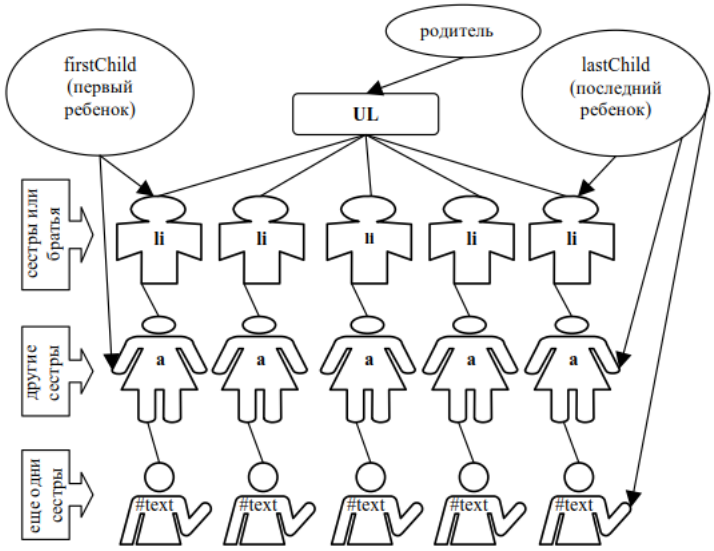
α.nextSibling

где *α* — ссылка на текущий сестринский узел (sibling в пер. с *англ.* — брат или сестра вершины дерева, имеющие общего родителя).

Первое свойство используется для ссылки на предыдущий сестринский узел, второе — на следующий сестринский узел.

В скрипте предыдущего примера можно заменить строку в цикле на *пункт_списка[i]=пункт_списка[i-1].nextSibling*, и результат останется без изменения.

```
<script type = "text/javascript">
  узел_список = document.getElementById ("список")
  var пункт_списка = new Array ()
  пункт_списка[1] = узел_список.firstChild
  alert (пункт_списка[1].nodeName)
  for (i=2; i<=5; i++) {
    пункт_списка[i]=пункт_списка[i-1].nextSibling
    alert (пункт_списка[i].nodeName) }
</script>
```



7.3 Атрибуты и свойства

В структуре DOM все стандартные (HTML 4.0) атрибуты тегов доступны для чтения и изменения. Для проверки существования атрибута имеется метод *hasAttribute*, возвращающий логическое значение *true* или *false*.

hasAttribute(*α*)

где *α* — имя атрибута.

Однако этот метод все еще не поддерживается браузером Internet Explorer.

Прочитать из скрипта значение атрибута можно методом *getAttribute*.

getAttribute(α)

где *α* — имя атрибута.

Напишем скрипт, возвращающий значение атрибута *bgcolor*, для примера 1.2.

¶

Пример 1.3

```
<script type = "text/javascript">
  узел_body=document.body
  значение_bgcolor=узел_body.getAttribute ("bgcolor")
  alert (значение_bgcolor)
</script>
```

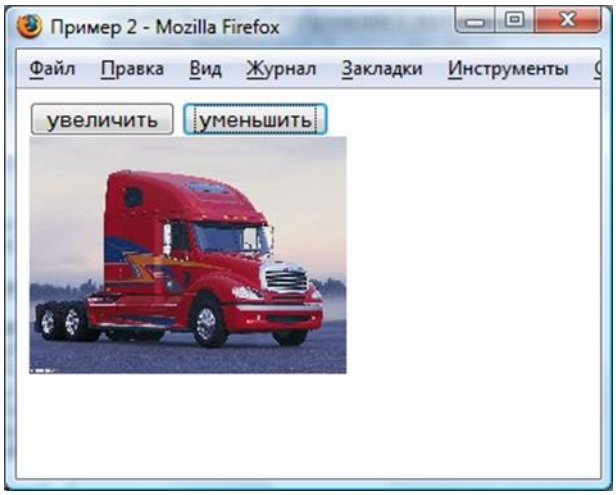
Часто атрибуты тегов являются свойствами соответствующих объектов-узлов DOM. Например, графический элемент *<img src = car161.jpg id = "авто" width = “100”>* имеет атрибут *width*, который можно изменить в скрипте традиционным путем как изменение значения свойства объекта.

```
авто_узел = document.getElementById ("авто")
авто_узел.width = 500
```

Пример 2.1. Зум картинки.

```
<html>
  <head>
    <title> Пример 2 </title>
  </head>
  <body>
    <input type=”button” value="увеличить" onClick="зум1 ()">
    <input type=”button” value="уменьшить" onClick="зум2 ()">
    <br>
    <img src=”car161.jpg” id="авто" width=”100”>
```

```
    <script type = "text/javascript">
      function зум1() {
        авто_узел=document.getElementById ("авто")
        авто_узел.width = авто_узел.width + 50
      }
      function зум2() {
        авто_узел=document.getElementById("авто")
        авто_узел.width = авто_узел.width - 50
      }
    </script>
  </body>
</html>
```



В функциях примера 2.1 мы видим ссылку на узел <img> с идентификатором «авто» (*авто_узел=document.getElementById ("авто")*). Следующая строка функции обращается к свойству width данного узла и либо увеличивает, либо уменьшает его значение.

Изменять или устанавливать значение атрибута можно и через метод *setAttribute*.

$$\alpha.setAttribute\ (в,\gamma)$$

где α — ссылка на узел веб-страницы; $в$ — имя атрибута тега; γ — значение атрибута.

Параметр $в$ заключается в кавычки. Например,

```
авто_узел.setAttribute ("width", 50)
авто_узел.setAttribute ("width", "50")
авто_узел.setAttribute ("width", авто_узел.getAttribute ("width") - 50)
```

Удалить значение атрибута или сам атрибут можно посредством метода

removeAttribute.

α.removeAttribute (в)

где *α* — ссылка на узел веб-страницы; *в* — имя атрибута тега.

Добавим в предыдущий пример кнопку «исходное изображение» и соответствующую функцию, удаляющую параметр *width*, отвечающий за изменение размера картинки.

¶

Пример 2.2

```
<input type = “button” value = "исходное изображение" onClick = "оригинал ()">
function оригинал () {
  авто_узел=document.getElementById ("авто")
  авто_узел.removeAttribute ("width")
}
```

Браузер автоматически синхронизирует значения атрибутов и свойств. И все же они не всегда идентичны!

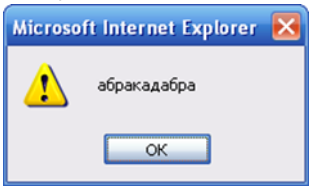
Например, изменим функцию *зум2*. Добавим строку, которая будет присваивать недопустимое значение параметру *width*. Недопустимой для *width* является любая строка, так как он отвечает за размер изображения, задаваемый числом.

Пример 2.3

```
function зум2 () {
  авто_узел=document.getElementById ("авто")
  авто_узел.setAttribute ("width", "абракадабра")
  alert (авто_узел.getAttribute ("width"))
  alert (авто_узел.width) }
```

Выполняя команды, браузер выведет в первом диалоговом окне значение «абракадабра», а в следующем окне числовое значение, отражающее текущий размер изображения.

Атрибут может быть любой строкой, он лишь показывает, что написано в исходном коде документа. Свойства же отображают текущее состояние тега, они не могут содержать недопустимых значений!



Универсальный атрибут *style* также доступен в DOM для чтения и изменения его значения.

α.style.в

где *α* — ссылка на узел веб-страницы; *в* — несколько измененное название свойства из таблицы стилей (CSS).

¶

```
var узел_table=document.getElementById ("таблица1")
узел_table.style.display='none'
```

Все дефисы в имени свойства должны быть удалены, а идущая после дефиса буква должна стать прописной, например fontSize вместо font-size.

```
ячейка = document.getElementById (“ячейка_столбца”)
ячейка.style.borderWidth = ’medium’
```

Свойство style содержит коллекцию всех css-свойств html-элемента. Но значения будут возвращены только те, которые заданы в теге html-элемента. Все значения, заданные в разделе <style type=”text/css”> или в отдельном файле, будут в скрипте проигнорированы (т. е. будут выдаваться пустые строки). Чтение css-свойств, определенных в разделе style, можно осуществить оператором getComputedStyle.

getComputedStyle(α,null).в

где *α* — ссылка на элемент; *в* — свойство элемента.

Например:

```
alert (getComputedStyle(узел_элемента, null).left)
```

Не всегда у атрибута и связанного с ним свойства одинаковые названия. Например, исключением является атрибут *class*, которому соответствует свойство *className*.

α.className = в

где *α* — ссылка на узел веб-страницы; *в* — имя класса стиля, заключенное в кавычки.

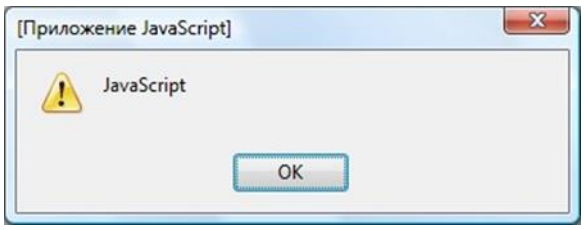
```
<style type = "text/css">
  img.sp {cursor: url(images/scale_plus.cur), pointer;}
  img.sm {cursor: url(images/scale_minus.cur), pointer;}
</style>
<body>
<img src=" name='оригинал' id="исходное" class="sp"
onClick="размер_изменить (this.src)">
<script type="text/javascript">
function размер_изменить (x) {
  ссылка=document.getElementById("исходное")
  if (x.indexOf("s640") > 0) {
    x=x.replace ("s640","s800")
    ссылка.className = "sm"
    document.оригинал.src=x
    }
  else {
    x=x.replace ("s800","s640")
    ссылка.className = "sp"
    document.оригинал.src=x
    }
}
</script>
</body>
```

7.4 Текстовый узел

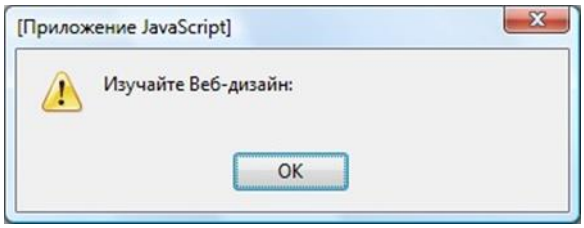
Текстовый узел представляет собой набор символов, заключенный в тег-контейнер или находящийся между открывающимся тегом и следующим дочерним узлом.

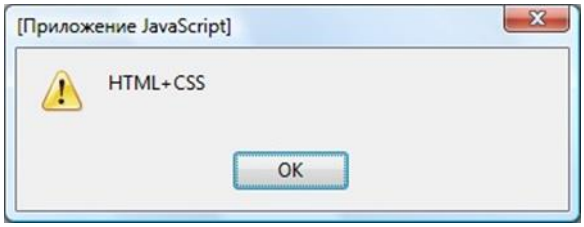
Пример 3.1

```
<body>
  <h1 id="заголовок1">JavaScript</h1>
  <p id="абзац1"> Изучайте Веб-дизайн: <strong> HTML+CSS
  </strong> </p>
</body>
<script type = "text/javascript">
  h1_узел = document.getElementById ("заголовок1")
  alert (h1_узел.firstChild.nodeValue)
</script>
```

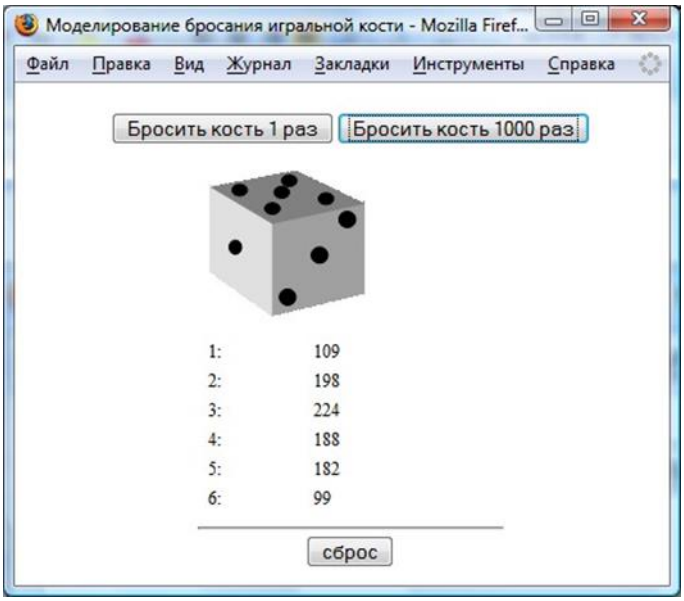


```
<script type = "text/javascript">
  p_узел = document.getElementById ("абзац1")
  alert (p_узел.firstChild.nodeValue)
  strong_узел = p_узел.childNodes [1]
  alert (strong_узел.firstChild.nodeValue)
</script>
```





Следующий пример демонстрирует, как можно менять текстовую информацию в ячейках таблицы с помощью ссылок на узлы — ячейки таблицы. Скрипт моделирует подбрасывание игральной кости с шестью гранями и подсчитывает число выпадений каждой грани, что отражается в таблице.



Пример 3.2. Моделирование бросания игральной кости.

```
<html>
```

```
<head>
<title> Моделирование бросания игральной кости </title>
<script type = "text/javascript">
var ad = new Array (7)
var d = new Array (7)
  for (var j = 1; j <= 7; j++) {
    ad [j] = new Image (50,50)
    d [j] = new Image ()
    fn = "animated_dice_" + j + ".gif"
    fnd = "d" + j + ".gif"
    ad [j].src = fn
    d [j].src = fnd
  }
var interval_id
var frame_number = 1
var k = 0
function start_animation () {
  interval_id = setInterval ("pk ()", 50)
}
function pk () {
  document.images["dice"].src = ad[frame_number].src
  frame_number ++
  if (frame_number == ad.length) {
    frame_number = 1
    k++
  }
  if (k == 5) {
    k = 0
    clearInterval (interval_id)
    подбросить_1раз ()
  }
}
function случайное_число () {
  var x = Math.random () * 5 + 1
  return Math.round (x)
}
var total1 = 0, total2 = 0, total3 = 0, total4 = 0, total5 = 0, total6 = 0
function подбросить_1раз () {
sch = случайное_число ()
  switch (sch) {
    case 1:
      total1++
      document.images["dice"].src = d[1].src
      td_узел=document.getElementById ("грань1")
      td_узел.firstChild.nodeValue = total1
      break
    case 2:
      total2++
      document.images["dice"].src = d[2].src
      td_узел = document.getElementById ("грань2")
      td_узел.firstChild.nodeValue = total2
      break
```

Предва-рительная загрузка изобра-жений

```
    case 3:
      total3++
      document.images["dice"].src = d[3].src
      td_узел = document.getElementById ("грань3")
      td_узел.firstChild.nodeValue = total3
      break
    case 4:
      total4++
      document.images["dice"].src = d[4].src
      td_узел=document.getElementById ("грань4")
      td_узел.firstChild.nodeValue = total4
      break
    case 5:
      total5++
      document.images["dice"].src = d[5].src
      td_узел=document.getElementById ("грань5")
      td_узел.firstChild.nodeValue = total5
      break
    case 6:
      total6++
      document.images["dice"].src = d[6].src
      td_узел=document.getElementById ("грань6")
      td_узел.firstChild.nodeValue = total6
      break
  }
}
function подбросить_1000раз () {
  for (var i = 1; i <= 1000; i++) {
    подбросить_1раз ()
  }
}
function сбросить () {
  for (i=1; i<=6; i++) {
    текущая_грань = "грань" + i
    td_узел=document.getElementById (текущая_грань)
    td_узел.firstChild.nodeValue = ""
  }
  total1 = 0
  total2 = 0
  total3 = 0
  total4 = 0
  total5 = 0
  total6 = 0
}
</script>
</head>
<body topmargin=20 leftmargin=20>
<div align=center>
<form name="f1">
<p>
<input type=button value="Бросить кость 1 раз" onClick="start_animation ()" style="font-size:20
pt">
```

```
<input type = button value = "Бросить кость 1000 раз" onClick = "подбросить_1000раз ()" style
= "font-size:20 pt">
</p>
<table style = "font-size: 20 pt" id = "таблица" width = 200>
  <tr height=120>
  <td colspan=2> <img src=animated_dice_1.gif name="dice"> </td>
    </tr>
    <tr> <td>1:</td> <td id="грань1">   </td> </tr>
    <tr> <td>2:</td> <td id="грань2">   </td> </tr>
    <tr> <td>3:</td> <td id="грань3">   </td> </tr>
    <tr> <td>4:</td> <td id="грань4">   </td> </tr>
    <tr> <td>5:</td> <td id="грань5">   </td> </tr>
    <tr> <td>6:</td> <td id="грань6">   </td> </tr>
</table>
<p></p>
<hr width=50%>
<input type=button style="font-size: 20 pt" value="сброс" on- Click="сбросить ()">
</form>
</div>
</body>
</html>
```

Первые строки скрипта до первой функции осуществляют предварительную загрузку изображений игральной кости.

Первый набор из 7 картинок предназначен для организации анимации бросания игральной кости. Второй набор из 6 изображений — для демонстрации выпавшей грани.

7.5 Операции с узлами

На html-узлы можно не только ссылаться и менять их свойства, но и удалять их, добавлять новые узлы в существующую структуру документа, заменять отдельные элементы веб-страницы на другие.

Добавить новый элемент в текущий документ можно с помощью двух шагов. Сначала инициируется метод создания узла, который просто резервирует данный объект. Затем применяется метод добавления зарезервированного узла как дочернего элемента к существующему узлу.

document.createElement (α)

где α — имя создаваемого узла-тега. Например:

```
узел_li = document.createElement ("li")
```

α.appendChild (β)

где α — имя узла, к которому добавляется созданный элемент β. Например:

```
узел_ul.appendChild (узел_li)
```

¶

Добавление текстового узла осуществляется методом *createTextNode*.

document.createTextNode (α)

где α — текстовая строка. Например:

```
узел_текст = document.createTextNode (“изучайте веб-дизайн”)
узел_li. appendChild (узел_текст)
```

Удалить дочерний узел можно методом *removeChild*.

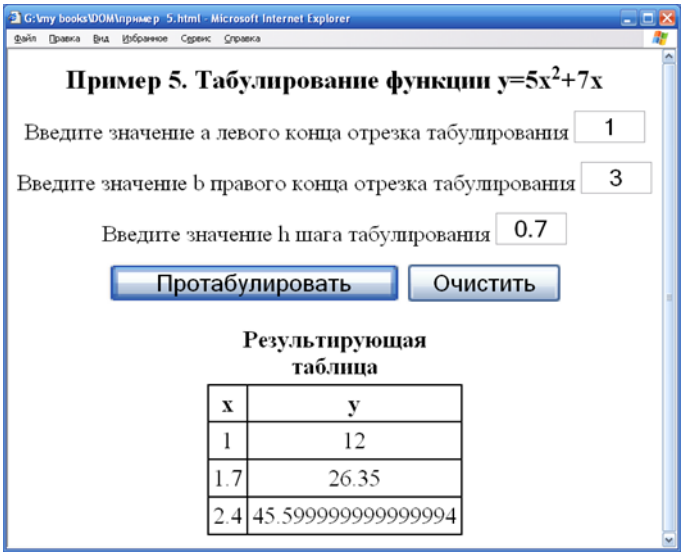
α.removeChild (β)

где α — родительский узел; β — дочерний узел. Например:

```
узел_родитель = узел_tr.parentNode
узел_родитель.removeChild (узел_tr)
```

Пример 3.3. Табулирование функции.

Применим методы изменения узлов для создания, добавления и удаления строк и ячеек таблицы, содержащих информацию о значениях функции в заданных точках.




```
<html>
<head>
<style type=”text/css”>
  input, select, table {
    font-size: 21pt;
    text-align: center;
}
</style>
<script type = "text/JavaScript">
var i
function создать_таблицу () {
  a=parseInt(document.forms[0].a.value)
  b=parseInt(document.forms[0].b.value)
  h=parseFloat(document.forms[0].h.value)
  var узел_table = document.getElementById ("таблица1")
  узел_table.style.display = ''
  i=0
  for (x=a;x<=b;x=x+h) {
    y=5 * Math.pow (x, 2) + 7 * x
    узел_tbody=document.createElement ("tbody")
    узел_table.appendChild (узел_tbody)
    узел_tr=document.createElement ("tr")
    узел_tr.id="строка"+i
    узел_tbody.appendChild (узел_tr)
    узел_td=document.createElement ("td")
    узел_tr.appendChild (узел_td)
    узел_x=document.createTextNode(x)
    узел_td.appendChild (узел_x)
    узел_td=document.createElement ("td")
    узел_tr.appendChild (узел_td)
    узел_y=document.createTextNode(y)
    узел_td.appendChild (узел_y)
    i++
  }
}
```

```
function очистить () {
  узел_table = document.getElementById ("таблица1")
  document.forms[0].a.value = ""
  document.forms[0].b.value = ""
  document.forms[0].h.value = ""
  for (j=0; j<i; j++) {
    Id_td="строка" + j
    узел_tr = document.getElementById (Id_td)
    узел_родитель = узел_tr.parentNode
    узел_родитель.removeChild (узел_tr)
    }
  узел_table.style.display="none"
}
</script>
</head>
<body style="font-size: 20pt">
<h1 style= “text-align: center;” onCopy="return false;"> Пример 5. Табулирование функции
y=5x <sup>2</sup>+7x </h1>
<form>
<div style= “text-align: center;”>
<P>Введите значение a левого конца отрезка табулирования <input name="a" type="text"
size="3" value="7" > </p>
<p>Введите значение b правого конца отрезка табулирования <input name="b" type="text"
size="3" value="9"> </p>
<p>Введите значение h шага табулирования
<input name="h" type="text" size="3" value="0.2"> </p>
<p> <input type="button" value="Протабулировать" style="font-size: 21pt"
onClick="создать_таблицу ()">
<input type="button" value="Очистить" style="font-size: 21pt" on-Click="очистить ()"> </p>
<table border="1" width=35% cellpadding="5"
cellspacing="5" id="таблица1" style="display: none; text-align: center;">
<caption> Результирующая таблица </caption>
  <tr>
    <th>x</th>
    <th>y</th>
  </tr>
</table>
</div>
</form>
</body>
</html>
```

Следует обратить внимание на то, что строки таблицы добавляются к узлу <TBODY>. Это позволяет работать скрипту не только в *Mozilla FireFox*, но и в *Internet Explorer*.

¶

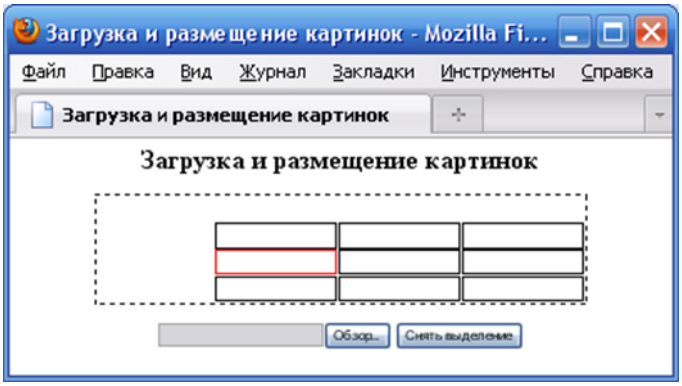
Пример 3.4. Загрузка и размещение картинок (для браузера Mozilla Firefox).

Рассмотрим более сложный пример. Средствами HTML разместим на веб-

странице таблицу, средствами CSS будем видоизменять курсор мыши для выделения строк и столбцов таблицы. Создадим скрипт, который будет в зависимости от выделенной части таблицы выводить на страницу кнопки с названиями соответствующих действий над таблицей, а при щелчке на кнопке осуществлять эти действия.

Создадим функции для выделения частей таблицы (ячейки, строк, столбцов) и снятия выделения, добавления и удаления строки и столбца, вставки изображения (из той же папки, где находится html-документ) в выделенную ячейку.

Исходный код примера смотри на прилагаемом компакт-диске.



¶

Практическая работа №7

Задание 1

Постройте дерево тегов для следующего html-документа.

```
<html>
<head>
</head>
<body style="font-size: 20pt">
  <h1 style= “text-align: center;”>
    Проект. Число слов
  </h1>
  <form>
  <div style= “text-align: center;”>
  <p style= “text-align: top;”>Введите текст
  <textarea name="txt" rows="7" cols=35 style="display:inline; vertical-align:top; font-size:20pt;
font-style:italic;">
  Здесь должен располагаться Ваш текст
  </textarea>
  </p>
  <input type="button" value="Подсчитать слова" style="font- size:21pt"
onClick="подсчитать_слова ()">
  <input type="button" value="Очистить" style="font-size:21pt" onClick="чистить ()">
  <p>
  <select name='список_оригинальных_слов' size=5 style="font- size:21pt">
    <option>результат
    <option>будет
    <option>здесь
  </select>
  </p>
</div>
</form>
<hr>
</body>
</html>
```

¶

Задание 2

Напишите скрипт, который будет выводить в диалоговом окне:

1) все пункты списка из примера 1;
2) все гиперссылки (значения атрибута HREF) из примера 1.

Задание 3

Разработайте сценарий, который предлагает посетителю веб-страницы возможность открытия гиперссылки в новом окне браузера.

Задание 4. Веб-палитра

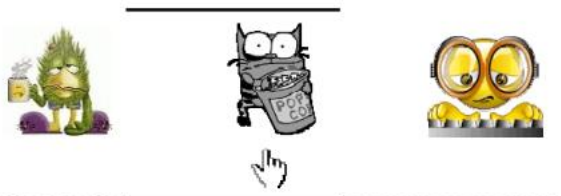
В веб-страницу «веб-палитра.html» (файл находится на компакт-диске в

папке «Материал к упражнениям»), содержащую таблицу с цветовой палитрой веба, добавьте сценарий с технологией DOM, который должен реагировать на щелчки мышкой по какому-либо цвету из палитры таким образом, чтобы посетитель мог видеть в отдельной достаточно большой области страницы выбранный цвет фона или текста.

Цвет	Название цвета	RGB	ТЕКСТ
	LightPink	#FFB6C1	пробный текст
	Pink	#FFC0CB	
	Crimson	#DC143C	
	LavenderBlush	#FFF0F5	ТЕКСТ
	Pale ioletRed	#DB7093	
	HotPink	#FF69B4	пробный текст
	DeepPink	#FF1493	
	MediumVioletRed	#C71585	
	Orchid	#DA70D6	
	Thistle	#D8BFD8	

Задание 5

Создайте html-документ, на котором разместите несколько картинок. Используя CSS, сделайте эффект появления верхней и нижней границ у картинки при наведении на нее курсора мыши. Используя JavaScript и DOM, разработайте сценарий, который будет при щелчке мышкой на картинке выводить в отдельную область информацию, связанную по смыслу с выбранной картинкой.



Коты бывают разные: мартовские, …

¶

Задание 6

Составьте сценарий, который после щелчка по кнопке будет осуществлять сортировку табличных данных или элементов списка.

ФИО	Группа	Результат	Дата	Таймер
Дмитриева М.	12	75	11/02/2007	10:56
Акишкина С.	бх-21 егф	82	6/26/2008	10:37
Дашкина Д.	Х-31	64	1/15/2009	12:21

Шокоров А.	ФИ-31	53	6/18/2009	12:58
Вдовина Е.С.	Фи-31	42	6/18/2009	14:21
Конаков М.	Фи-31	89	6/18/2009	14:40
Вдовина Е.	Фи-31	57	6/19/2009	10:21

Упорядочить по ФИО | Упорядочить по результату | Фильтр по группе | Фильтр по ФИО

Задание 7

Составьте сценарий, который после щелчка по кнопке будет осуществлять фильтрацию табличных данных или элементов списка.

Задание 8

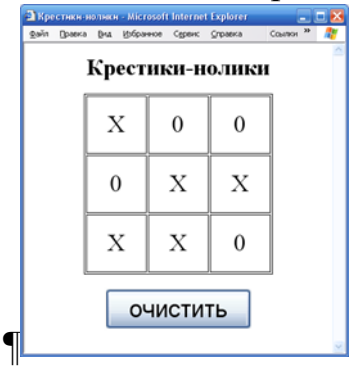
Составьте сценарий, который по введенным посетителем данным строит таблицу.

Задание 9

Составьте сценарий, который по введенным посетителем данным разместит список.

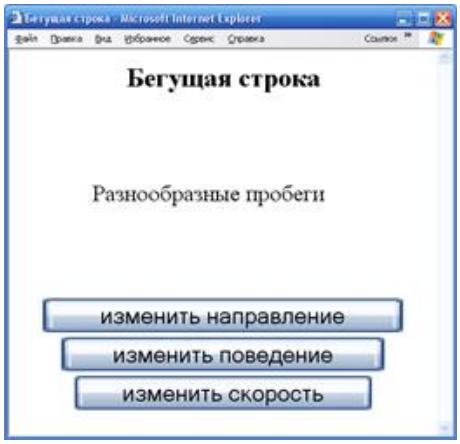
Задание 10. Крестики-нолики

Составьте сценарий, который сможет поддерживать игру в крестики-нолики. Посетитель выбирает щелчком ячейку, сценарий в нее заносит либо крестик, либо нолик, в зависимости от предыдущего хода.



Задание 11. Бегущая строка

Составьте сценарий, который сможет изменять различные параметры бегущей строки.



Задание 12. Облако тегов

Напишите скрипт, который будет подсчитывать одинаковые слова, введенные посетителем в текстовую область и формировать блок, в котором эти же слова будут отображаться размером шрифта, пропорционально количеству их повторений в тексте.

Литература

1. Пол Вилтон, Джереми МакПик. JavaScript. Руководство программиста. СПб: Питер, 2009-720 с.
2. Стоян Стефанов. JavaScript. Шаблоны. СПб: Символ-плюс, 2011-272 с.
3. Диков А. В. Клиентские технологии веб программирования: JavaScript и DOM : учебное пособие / А. В. Диков. — Санкт Петербург: Лань, 2020. — 124 с. ил
4. Дунаев В. HTML, скрипты и стили. СПб: БХВ-Петербург, 2011- 816 с.
5. Дронов В. HTML 5, CSS 3 и Web 2.0. Разработка современных Webсайтов. СПб: БХВ-Петербург, 2011- 416 с.
6. Джон Поллок. JavaScript. Руководство разработчика. СПб: Питер, 2011- 544 с.
7. Дэвид Макфарланд. JavaScript. Подробное руководство. Эксмо, 2009- 608 с.
8. Климов А. JavaScript на примерах. СПб: БХВ-Петербург, 2009-336 с.
9. Шафер С., HTML, XHTMLи CSS. Библия пользователя. М.: Вильямс, 2010 – 656 с.
10. Лабберс К., Олберс Н., Салим К.. HTML5 для профессионалов: мощные инструменты для разработки современных веб-приложений. М.: Вильямс, 2011 – 272 с.

Петр Георгиевич Асалханов
Надежда Владимировна Бендик

Web-программирование: JavaScript

Учебное пособие

Лицензия на издательскую деятельность
ЛР № 070444 от 11.03.98 г.
Подписано в печать 28.11.2020 г.
Тираж 30 экз.

Издательство Иркутского государственного аграрного
университета имени А.А. Ежевского
664038, Иркутская обл., Иркутский р-н, пос. Молодежный