

JavaScript. Учебное пособие

Диков Андрей Валентинович

2011



Министерство образования Пензенской области
Государственное бюджетное образовательное
учреждение дополнительного профессионального
образования
«Пензенский институт развития образования»

А.В. Диков

ОСНОВЫ ВЕБ-ДИЗАЙНА:

JavaScript

Учебное пособие

Пенза
2009

УДК 681.142.3(075)

Печатается по решению редакционно-издательского совета государственного бюджетного образовательного учреждения дополнительного профессионального образования «Пензенский институт развития образования» (ПИРО)

Рецензенты:

Н.Б. Тихонова, кандидат педагогических наук;

О.П. Графова, кандидат педагогических наук.

В авторской редакции.

Учебное пособие является логическим продолжением предыдущей работы «Основы веб-дизайна: HTML+CSS» и предназначено для веб-дизайнеров, желающих освоить клиентскую технологию разработки динамических и интерактивных веб-страниц.

© ГБОУ ДПО «Пензенский институт развития образования», 2009

© А.В. Диков, 2009

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ	5
БАЗОВЫЕ СОБЫТИЯ	6
ПЕРЕМЕННЫЕ И ЗНАЧЕНИЯ	8
ФУНКЦИИ ПРЕОБРАЗОВАНИЯ.....	11
УПРАВЛЯЮЩИЕ КОНСТРУКЦИИ.....	14
ПРОЦЕДУРЫ И ФУНКЦИИ	18
МАТЕМАТИКА НА ВЕБ-СТРАНИЦАХ. ОБЪЕКТ MATH.....	23
ПРАКТИЧЕСКАЯ РАБОТА 1	31
МАССИВЫ	36
СТРОКИ. ОБЪЕКТ STRING	39
РЕГУЛЯРНЫЕ ВЫРАЖЕНИЯ	42
ПРАКТИЧЕСКАЯ РАБОТА 2	47
ДАТА И ВРЕМЯ. ОБЪЕКТ DATE.....	49
ПРАКТИЧЕСКАЯ РАБОТА 3	52
ОБЪЕКТНАЯ МОДЕЛЬ БРАУЗЕРА.....	53
КУКИЗ (COOKIES)	76
ПРАКТИЧЕСКАЯ РАБОТА 4	80
ПРАКТИЧЕСКАЯ РАБОТА 5	82
БИБЛИОГРАФИЯ	84

ВВЕДЕНИЕ

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 году фирмой Netscape в сотрудничестве с Sun Microsystems на базе языка Sun's Java*. По инициативе компании Netscape была проведена стандартизация языка ассоциацией ECMA. Стандартизированная версия имеет название ECMAScript, описывается стандартом ECMA-262. Первой версии спецификации соответствовал JavaScript версии 1.1

Операторы JavaScript размещаются в контейнере `<SCRIPT>` и разделяются символом `”;`. Для браузеров, не поддерживающих скрипты, операторы заключают еще в теги комментариев языка HTML, чтобы они не были видны посетителю при просмотре веб-страницы.

* Название Java было дано языку в честь любимой разработчиками марки кофе

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

Приведенный скрипт выводит на веб-страницу текстовую строку «Это JavaScript!». Подробное объяснение метода *write* объекта *document* можно найти на странице 67.

Комментарии JavaScript отличаются от комментариев языка HTML.

// комментарии на одной строке

/*

комментарии на
нескольких строках

*/

Язык чувствителен к регистру при задании значений параметров!

Сценарии с глобальными функциями и переменными рекомендуются размещать в заголовочном разделе HTML-документа (см. пример в конце следующего раздела). Функции могут быть вызваны многократно.

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 другие. 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

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

```

<IMG SRC = "smile.gif" NAME = "wq" onMouseOver =
    "смени_изо ()" onMouseOut = "верни_изо ()">
<SELECT NAME = pict SIZE = 7 onClick = "ch_pict ()">
    <BODY onLoad="ss (); clock ()"
    BACKGROUND=" ../pict/lvb.jpg">
<FORM ACTION="formtest.exe" METHOD="post"
    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 ("Ky-ky")>
```

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 выполняется 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.

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**, после которого следует имя переменной или нескольких переменных, разделенных запятой. Имена переменных, как и все другие элементы языка чувствительны к регистру.

$var \alpha \{ \{, \beta \} \}$

```
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
```

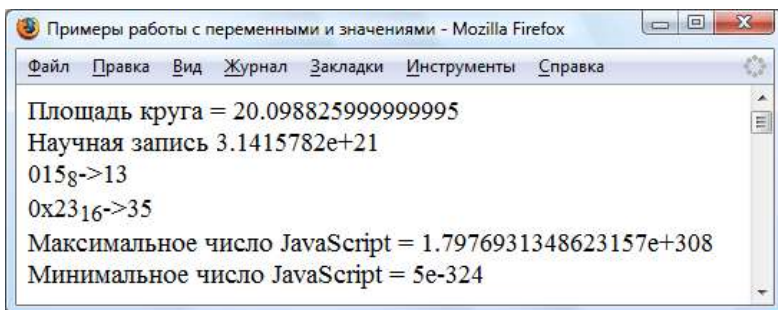
```
document.write ("0x23", "<SUB>16</SUB>", "->", шч, "<br>")
```

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, $5 \cdot x - 7 \% 2 + x / 5$



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

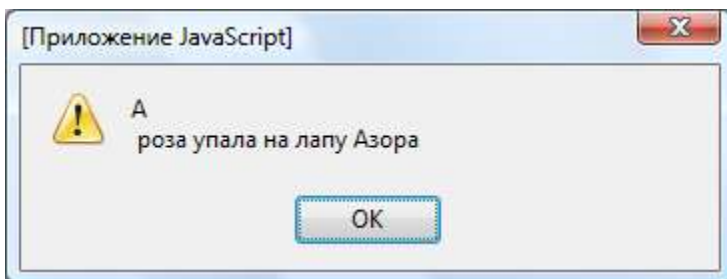
`\\` Обратная косая черта

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

`document.write (строка,"
")`

`var строка2='А \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 означает 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")
```

Про методы *prompt* и *alert* можно почитать на странице 60, про оператор *if* – на странице 15.

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.

$$\alpha.toString ([\beta])$$

где β – основание для представления числовых значений.

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

Функция *eval* преобразует строку в исполняемый код JavaScript.

$$eval (\alpha)$$

```
eval ("alert ('Helo')")
```

```
document.write (eval ('Number.MAX_VALUE'))
```

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

$$\alpha.toFixed (\beta)$$

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

Функция *toExponential* возвращает преобразованное в экспоненциальную форму число α с β знаками после запятой.

$$\alpha.toExponential (\beta)$$

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.

$$\alpha.toExponential (\beta)$$

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

УПРАВЛЯЮЩИЕ КОНСТРУКЦИИ

Цикл

while (α) { β }

где α – условие, являющееся логическим выражением; пока его значение истинно, цикл выполняется;

β – тело цикла, включающее в себя последовательность операторов JavaScript.

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 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

```
<script type="text/javascript">
  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--
  }
</script>
```

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.

$$\textit{for}(\textit{var } \alpha \textit{ in } \beta) \{ \lambda \}$$

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.

Развилка (Условный переход)

```
if ( $\alpha$ )
  {  $\beta$  }
[ else
  {  $\gamma$  } ]
```

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
>	больше
<	меньше

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

Пример

```
<input type = "button" value = "Click me" onClick = "if
(document.form1.tf.value < 1) { document.write ('hello!'); }">
```

Множественный переход

```
switch ( $\alpha$ )
{ case  $\beta_1$ :
     $\gamma_1$ 
    break
  case  $\beta_2$ :
     $\gamma_2$ 
    break
  ...
  default:  $\gamma_d$ 
}
```

где α – переменная или выражение с переменной,
 β_1, β_2, \dots – возможные значения α ,
 $\gamma_1, \gamma_2, \dots, \gamma_d$ – команды JavaScript.

Оператор *switch* заменяет несколько операторов *if*. При каждом предусмотренном значении α будет вызываться на выполнение 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++; break
}
}
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

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.

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

где α – имя функции, которое должно состоять только из одного слова;

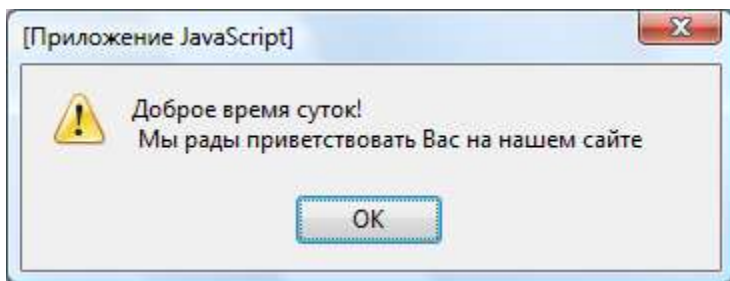
$\beta, \gamma \dots$ – необязательные входные аргументы (параметры) функции;

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 (Пример 5), либо как обработчик события (пример смотри на странице 7).

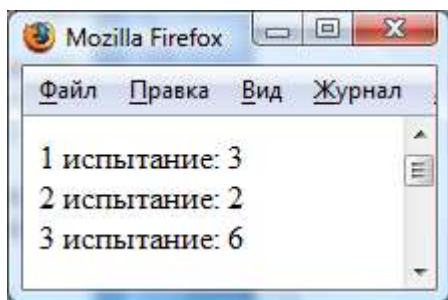
Пример 1. Функция создает оператор *приветствие*. Вызов функции осуществляется в скрипте

```
<script type="text/javascript">
function приветствие () {
    var 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 . О методах объекта *Math* смотри на странице 23.

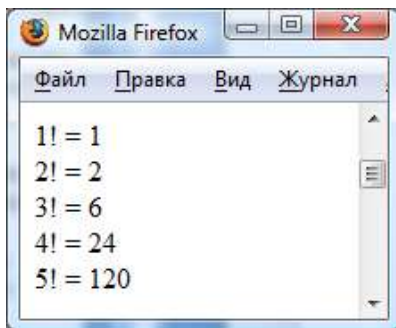
```
<script type="text/javascript">
function случайное_целое () {
    var x = Math.random () * 5 + 1
    return Math.round (x)
}
document.write ("1 испытание: ", случайное_целое(), '<br>')
document.write ("2 испытание: ", случайное_целое(), '<br>')
document.write ("3 испытание: ", случайное_целое(),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



Пример 3. Функция с параметром

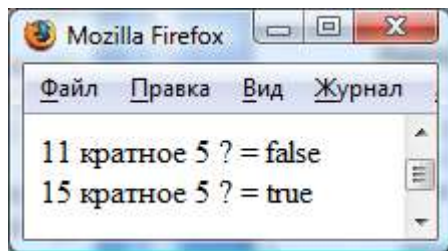
```
function факториал (n) {
    var факт = 1
    for (i = 1; i <= n; i++) факт = факт * i
    return факт
}
```

```
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>")
```



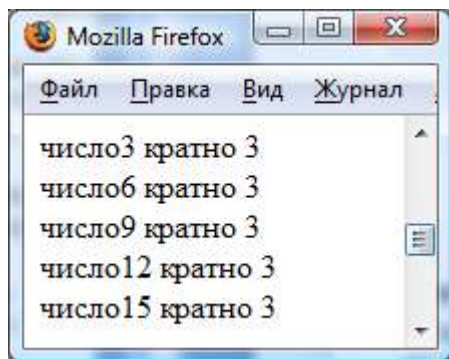
Пример 5. Вызов функции из другой функции

```
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)

```

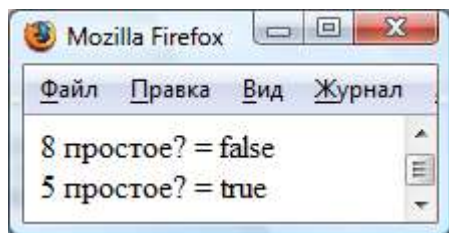


Пример 6. Досрочное прекращение работы функции (досрочный выход из цикла - break)

```

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>" ) }

```

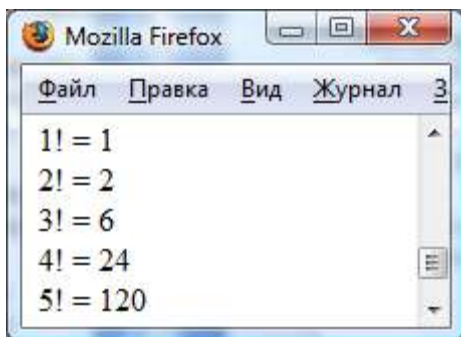


Таблица значений

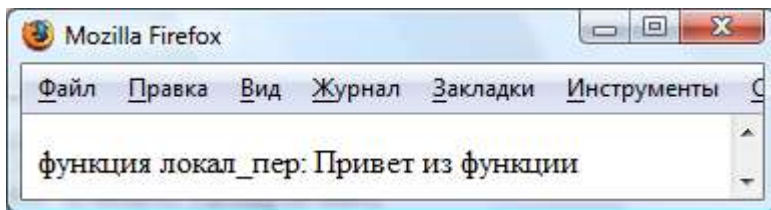
стека		
n		
4	4*3!	24
3	3*2!	6
2	2*1!	2
1	1!	1

Пример 8. Локальная переменная (область видимости)

```

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 процедурой, в которой она определена. Область видимости – это строки кода, где доступно текущее значение переменной.



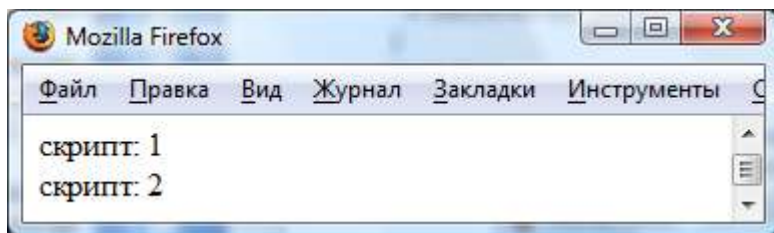
В примере 8 создается локальная переменная *лп*, значение которой может быть выведено только из тела функции. Внешний вызов будет проигнорирован браузером.



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>")
```



МАТЕМАТИКА НА ВЕБ-СТРАНИЦАХ. 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	ln 2	0.6931471805599453
Math.LN10	ln 10	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)	$ \alpha $	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)	$\operatorname{arctg} \alpha$	арктангенс аргумента в диапазоне от $-\frac{\pi}{2}$ до $\frac{\pi}{2}$.
Math.atan2 (α, β)	$\operatorname{arctg} \frac{\alpha}{\beta}$	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	е в степени α
Math.floor (α)		наибольшее целое, меньшее или равное α
Math.log (α)	$\ln \alpha$	натуральный логарифм числа α
Math.max (α, β)		максимальное значение из α и β
Math.min (α, β)		минимальное значение из α и β
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)	$\sqrt{\alpha}$	строковое представление α
Math.tan (α)	$\operatorname{tg} \alpha$	тангенс α

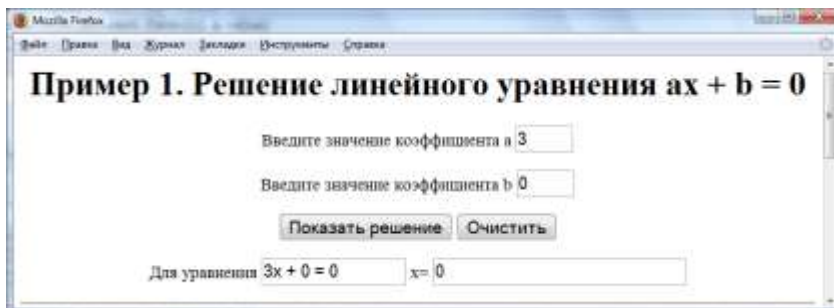
Где α – число, либо числовая переменная или выражение.

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
|
|      |      |       $x := -\frac{a}{b}$ 
|      |      |      вывод «x = », x
|
|      все
```

кон



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

```

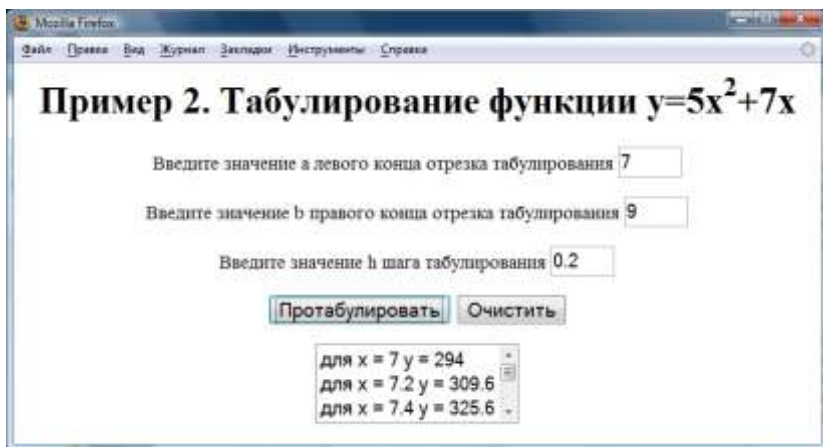
    if ((a==0) && (b==0))
        document.forms[0].x.value= 'решений бесконечно много'
    if ((a==0) && !(b==0))
        document.forms[0].x.value= 'нет решения'
    if (!(a==0)) document.forms[0].x.value=-b/a
    }
</script> </head>
<body style="font-size: 14pt" >
    <H1 align=center>
        Пример 1. Решение линейного уравнения  $ax + b = 0$  </H1>
    <form> <div 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"
            onClick="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 $y = 5x^2 + 7x$
нач цел a, b вещ h
| ввод a, b, h
| нц для x от a до b шаг h
| | $y := 5 * \text{Math.pow}(x, 2) + 7 * x$
| | вывод y
| кц
кон



```

<html> <head>
  <style> INPUT, SELECT {font-size: 14pt} </style>
  <script type="text/JavaScript">
    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[i] = 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 align=center> Пример 2. Табулирование функции
 $y=5x^2+7x$ 
</H1>
<form> <div 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="Тротабулировать"
    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

Динамическое изменение списка

Для добавления нового пункта в размещенный список формы используется конструктор *Option*

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

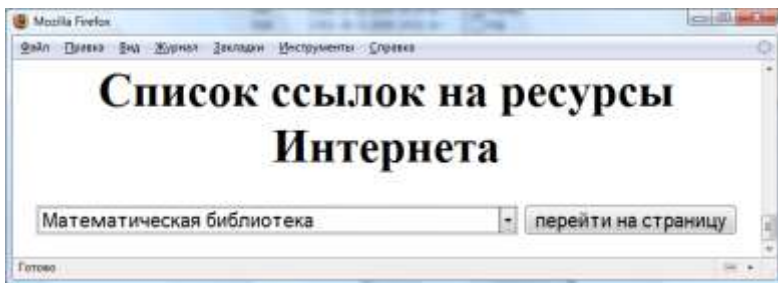
где α – имя элемента *select*,
 β – номер нового пункта списка;
 γ_1 – строка, которая размещается как пункт списка, можно обращаться к этому значению, используя свойство *text*;
 γ_2 – значение, которое передается серверу при выборе данного пункта, можно обращаться к этому значению, используя свойство *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*. Для определения номера выбранного пункта списка используется свойство *selectedIndex* объекта *select*. 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].список_ссылок.selectedInde
x
    return (
        docu-
ment.forms[0].список_ссылок.options[выбранный_пункт_списка].value)
    }
</script> </head>
<body>
<h1>Список ссылок на ресурсы Интернета</h1>
<form>
    <select name='список_ссылок' 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="http://golovolomka.hobby.ru/">Головоломки для
умных людей</option>
        <option value="http://www.etudes.ru/">Математические
этюды</option>
        <option val-
ue="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

Проект 1. «Квадратное уравнение».

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

Решение квадратного уравнения $ax^2 + bx + c = 0$

Дано:

Введите значение коэффициента a : 7

Введите значение коэффициента b : 9

Введите значение коэффициента c : 12

Решение:

$D =$ 0

Для уравнения:

$x_1 =$ 0

$x_2 =$ 0

Показать решение Очистить

Проект 2. «Табулирование любой функции». Усовершенствуйте пример 2 таким образом, чтобы табулирование осуществлялось для любой введенной посетителем функции. Дополнительно предусмотрите, чтобы сценарий определял монотонность функции на заданном отрезке и находил наибольшее и наименьшее значение.

Проект 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).

Проект 4. «Формулы». Разработайте сценарий для веб-страницы, который размещает на веб-странице список с названиями следующих функций: секанс, косеканс, котангенс, арксинус, арккосинус, арксеканс, арккосеканс, арккотангенс, гиперболический синус, гиперболический косинус, гиперболический тангенс, гиперболический секанс, гиперболический косеканс, гиперболический котангенс, гиперболический арксинус. Организуйте ввод числа и вывод в текстовое поле значения указанной посетителем функции.

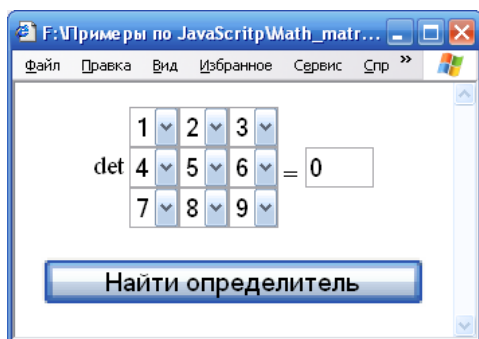
Проект 5. «Приближенные методы решения нелинейного уравнения». Разработайте сценарий для веб-страницы, который находит корень нелинейного уравнения с одной переменной с заданной точностью приближенным методом. Входные данные: уравнение;

концы отрезка, на котором локализован один корень; точность. Выходные данные: корень уравнения.

Проект 6. «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 так далее.

Проект 7. «Определитель матрицы». Разработайте сценарий для веб-страницы, который размещает на странице квадратную матрицу какого-либо размера в виде совокупности списков. Списки предоставляют посетителю удобную возможность быстрого заполнения матрицы элементами. Напишите функцию, которая вычисляет определитель матрицы.

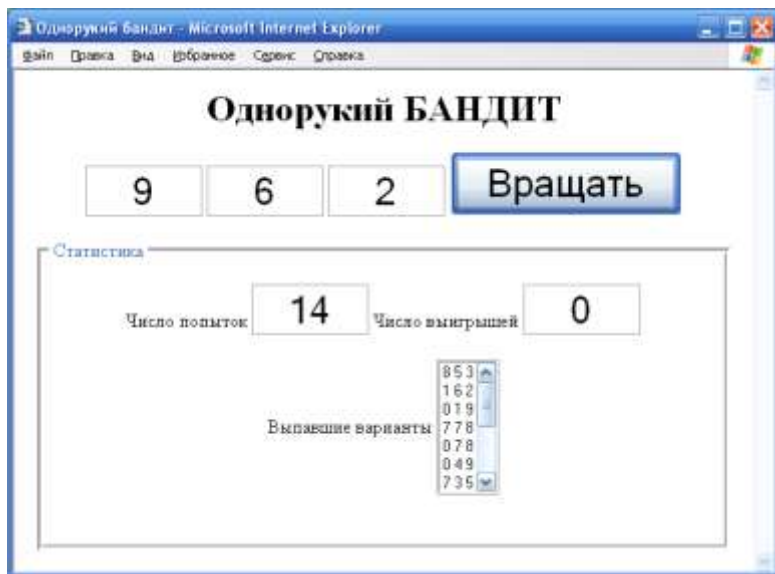
$$\det \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} = a_{11}(a_{22}a_{33} - a_{32}a_{23}) - a_{12}(a_{21}a_{33} - a_{31}a_{23}) + a_{13}(a_{21}a_{32} - a_{31}a_{22})$$



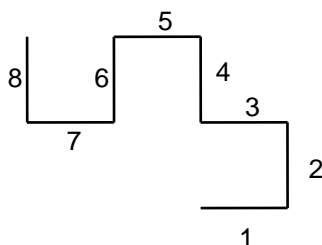
Проект 8. «Числа Фибоначчи». Разработайте сценарий для веб-страницы, который выводит в список последовательность чисел Фибоначчи в диапазоне от 1 до n. Числами Фибоначчи называются элементы числовой последовательности 1, 1, 2, 3, 5, 8, 13, 21, ..., в которой первые два элемента равны 1, а каждый следующий элемент представляет собой сумму двух предыдущих.

Проект 9. «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.



Проект 10. «Кривая дракона». Разработайте сценарий для веб-страницы, который генерирует последовательность из N чисел, отражающую ход кривой дракона, то есть 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.



Проект 11. «Магазин». Разработайте сценарий для веб-страницы, который формирует корзину выбранного посетителем периферийного оборудования для ПК.

Принтеры

HP Color LaserJet 2600n	▲
HP LaserJet 1018	
Lexmark X2350	
Epson Stylus C110	
Ricoh Aficio CL7100D	▼

Мониторы

LG Flatron L226WTQ-WF	▲
Samsung SyncMaster 225BW	
View Sonic VX2255w mb	
Acer X221W	
Envision G22LWK	▼


Добавить в корзину

Добавить в корзину

Интерактивные доски

SMARTboard	▲
Interwrite SchoolBoard	
PolyVision Webster	
Promethean ACTIVboard 50	
ACTIVboard 64	▼

Добавить в корзину



Выбранные товары (корзина)

Удалить выбранный товар

Валюта

Стоимость выбранного товара

Общая стоимость покупки

Проект 12. «Совершенные числа». Разработайте сценарий для веб-страницы, который по заданному числовому диапазону отбирает и выводит в список все встречающиеся в диапазоне совершенные числа. *Совершенным* называется число, равное сумме своих делителей, не считая самого числа.

МАССИВЫ

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.


$$\text{var } \alpha = \text{new Array } ([\beta])$$

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.

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

где α – имя массива, η_1, η_2, \dots – 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"]
```

Некоторые 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 зна-	добавление элементов в конец массива

	чение или имя 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
$\alpha.slice(\beta_n, [\beta_k])$	α – имя массива, β_n – индекс начального элемента фрагмента, β_k – индекс конечного элемента фрагмента	возвращает фрагмент массива, начиная с элемента β_n и заканчивая элементом β_{k-1} включительно.
$\alpha.sort([\beta])$	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
$\alpha.splice(\beta_n, [\delta], \{\gamma\})$	α – имя массива, β_n – номер элемента, δ – число элементов, γ – значение	1. удаляет δ элементов массива, начиная с β_n , когда $\delta > 0$ или неопределено (до конца массива), а также когда не задано γ 2. удаляет и вставляет элементы, начиная с β_n , когда $\delta > 0$ или неопределено и заданы значения γ 3. вставляет элементы перед β_n , когда $\delta = 0$ и заданы значения γ
$\alpha.unshift(\{\beta\})$	α – имя массива, β – какое-либо значение	вставляет элементы β в начало массива

СТРОКИ. Объект String

Определение и примеры строки рассмотрены на странице 10 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 также 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
$\alpha.length$	α – имя строковой переменной	возвращает длину строки, то есть число символов строки
$\alpha.indexOf(\beta[, \gamma])$	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
$\alpha.lastIndexOf(\beta[, \gamma])$	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
$\alpha.charAt(\beta)$	α – имя строковой переменной, β – номер элемента строки α	возвращает символ строки α с номером β
$\alpha.charCodeAt(\beta)$	α – имя строковой переменной,	возвращает код символа строки α с

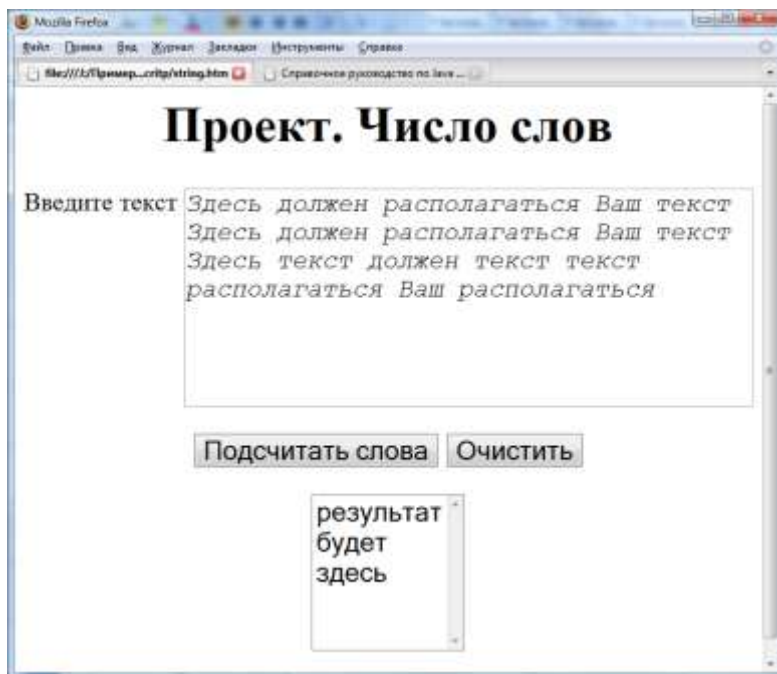
	β – номер элемента строки α	номером β
<i>String.fromCharCode</i> ($\beta_1 [, \beta_2]$)	β_1, β_2, \dots – число, числовая переменная или функция	возвращает строку из символов по кодам β_1, β_2, \dots
<i>α.slice</i> ($\beta_n [, \beta_k]$)	α – имя строковой переменной, β_n – номер начального элемента фрагмента строки α , β_k – номер конечного элемента фрагмента строки α	возвращает подстроку строки α с позиции β_n до позиции β_{k-1} . Если β_k не указан, то до конца строки α . Отрицательное значение β_k ведет отсчет от конца строки
<i>α.split</i> ($\beta [, \gamma]$)	α – имя строковой переменной, β – символ-разделить 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
<i>α.substr</i> ($\beta [, \gamma]$)	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
<i>α.substring</i> ($\beta_n [, \beta_k]$)	α – имя строковой переменной, β_n – номер начального элемента фрагмента строки α , β_k – номер конечного элемента фрагмента строки α	возвращает подстроку строки α от позиции β_n до β_{k-1} . Если β_k не указан, то до конца строки α
<i>α.toLowerCase</i> ()	переводит все символы строки α в нижний регистр	
<i>α.toUpperCase</i> ()	переводит все символы строки α в верхний регистр	
<i>α.replace</i> (β, γ)	возвращает строку α , в которой заменено первое вхождение подстроки β на γ . В качестве β может стоять регулярное выражение (см. следующий параграф)	

Пример 1. Подсчитайте число вхождений заданной буквы в тексте.
количество_a = 0

текст = "Здесь должен располагаться Ваш текст"

```
for (i = 0; i < текст.length; i++) {  
    if (текст.charAt(i) == "a") количество_a++ }
```

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 align=center>Проект. Число слов</H1>
<form>
<div align="center">
<P 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 текст должен текст
текст располагаться Ваш
располагаться</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>

```

РЕГУЛЯРНЫЕ ВЫРАЖЕНИЯ

Регулярное выражение – это еще один тип данных, являющийся маской для вводимых строковых данных. Оно задает структуру данных. Регулярное выражение задается между двумя косыми чертами / / и применяется к строке. Например, для проверки ввода ад-

реса 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.

Например, конструктор `RegExp("JavaScript")` создает регулярное выражение `/JavaScript/`. Конструктор `RegExp("JavaScript","g")` создает регулярное выражение `/JavaScript/g`.

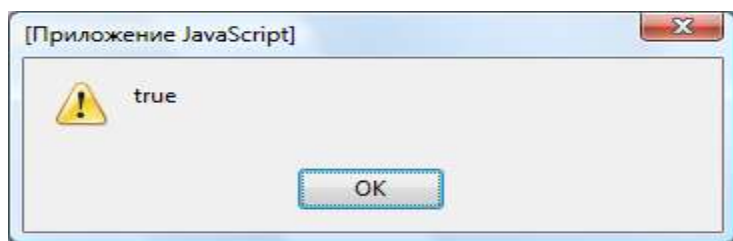
Флаг регулярного выражения	Значение
<code>i</code>	поиск, нечувствительный к регистру
<code>g</code>	глобальный поиск, то есть поиск всех соответствий
<code>m</code>	многострочный поиск

Применить регулярное выражение к строке можно с помощью метода `test`.

$\alpha.test(\beta)$

где α – объект, дающий регулярное выражение, β – объект, возвращающий проверяемую строку. Метод `test` возвращает логическое значение `true` или `false` в зависимости от проверяемой строки. Например,

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



Существуют спецсимволы для построения регулярных выражений.

спецсимвол		регулярное выражение	строка	соответствие
заменитель цифр (d от digital)	\d	^d\d\d/ ^d\d/	"587" "587"	true false
нецифровой заменитель	\D	^D\D\D/ ^D\D\D/ ^D\D\D\D\D/D/	"123" "a2B" "Арбуз!"	false false true
заменитель цифр, букв латинского алфавита и подчеркивания (w от word)	\w	^w\w\w\w\w/ ^w\w\w\w/	"12D_7" "12DШ"	true false
заменитель символов, отличных от цифр, букв латинского алфавита и подчеркивания	\W	^W\W\W\W/ ^W\W\W\W/	"ШОП!" "12DШ"	true false
заменитель символов, отличных от разделителя строки	.	/.../ /.../	"12DШ" "топт"	false true
заменитель неотображаемых символов (которые разделяют данные на слова, абзацы, строки и т.д.)	\s	^D\D\D\s\D\D/ ^D\D\D\s\D\D/	"топ ле" "топ_ле"	true false
заменитель отображаемых символов	\S	^D\D\D\S\D\D/ ^D\D\D\S\D\D/ /T\Sл/	"топ_ле" "топ ле" "т+л"	true false true
заменитель группы символов, один	[]	/[123]\D\W/ /[([123]\D\Щ)]/	"3p." "(3p.)"	true true

из которых должен использоваться при вводе данных		/[1-300]\D\W/	"33p."	true
заменитель группы исключаяющих символов	[^]	/[^1-300]\D\W/	"33p."	false
заменитель начала или конца выражения (искл. для кириллицы)	\b	^\bPenza/ ^\bPenza/	"Penza city" "Russia_Penza"	true false
исключающий заменитель начала или конца выражения (b от begin)	\B	^\BPenza/ ^\BPenza/	"Penza_city" "city_Penza"	false true
заменитель одного символа или указатель на отсутствие символа	?	/o?да/ /o?да/ /o?да/ /o?да/	"ода" "да" "о-да" "а"	true true true false
заменитель нескольких символов или указатель на отсутствие символа	*	/бал*/ /бал*/ /бал*/	"балда" "бурда" "бард"	true false true
заменитель одного или нескольких символов	+	/o+да/ /o+да/ /o+да/ /o+да/	"ода" "да" "о-да" "а"	true false false false
заменитель повторяющихся символов	{n}	^D\D\d{6}\D/	"т.445566."	true
заменитель многократно повторяющихся символов (как минимум n раз)	{n,}	^D\D\d{6,}\D/	"т.44556677."	true
заменитель повторяющихся символов в диапазоне от n до m раз	{n,m}	/Ay{1,6}ч/ /Ay{1,6}ч/ /Ay{1,6}ч/	"Аyyyyyyч" "Ауч" "Аyyyyyyч"	true true false
заменитель начала	^	^M/	"Mozilla"	true

строки		/^М/	"mozilla"	false
заменитель конца строки	\$	/Fox\$/	"FireFox"	true
		/Fox\$/	"FireFoxMoz"	false
заменитель одного из шаблонов		/Fox\$ Moz\$/	"FireMoz2"	false
		/Fox\$ Moz\$/	"FireMoz"	true
		/Fox\$ Moz\$/	"FireFox"	true

Пример. 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)
```

Практическая работа 2

Проект 1. «Число слов». Усовершенствуйте скрипт из примера 2 таким образом, чтобы он учитывал знаки препинания, то есть чтобы знаки препинания не влияли на подсчет одинаковых слов.

Проект 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.

Проект 3. «Шифрограмма». Разработайте сценарий, который сможет зашифровывать многострочный текст, размещенный на веб-странице методом сдвига по алфавиту. Предусмотрите возможность расшифровки сообщения.

Проект 4. «Число словами». Разработайте сценарий, который сможет словами написать заданное число. Например 5 535 -> Пять тысяч пятьсот тридцать пять.

Проект 5. «Автодобавление». Разработайте сценарий, который сможет по нажатию на соответствующую кнопку добавлять часто употребляемые слова, например: добрый день, спасибо, пожалуйста, к сожалению, и так далее.

Проект 6. «Автозамена». Разработайте сценарий, который сможет по введенному при наборе текста шаблону (например, дбр) автоматически заменять его на соответствующее слово (например, добрый).

Проект 7. «Поиск и замена». Разработайте сценарий, который сможет осуществлять поиск по образцу и замену найденного фрагмента на заданный. Предусмотрите подсчет числа замен.

Проект 8. «Лишние пробелы». Разработайте сценарий, который сможет удалять идущие подряд несколько пробелов, оставляя только один.

Проект 9. «Палиндром». Разработайте сценарий, который сможет из введенного посетителем слова построить слово-перевертыш и определить совпадает ли слово-перевертыш с исходным словом.

Проект 10. «Удаление ненужных слов». Разработайте сценарий, который сможет удалять из текста заданные слова. Предусмотрите подсчет числа удалений.

Проект 11. «Заглавная буква». Разработайте сценарий, который переведет в верхний регистр первую букву каждого слова, следующего за точкой.

Проект 12. «Разрядка числа». Разработайте сценарий, который в заданное число добавит пробелы так, чтобы они отделяли по 3 разряда справа налево.

Проект 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).

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  $\alpha$  = new Date ( )
```

где α – имя переменной, которой будет присвоено значение текущей даты и времени.

метод объекта Date	возвращаемое значение
$\alpha.getFullYear()$	возвращает год в полном формате
$\alpha.getYear()$	возвращает год в кратком или полном формате (в зависимости от браузера)
$\alpha.getMonth()$	возвращает номер месяца (от 0 до 11)
$\alpha.getDate()$	возвращает число месяца (от 1 до 31)
$\alpha.getDay()$	возвращает день недели (от 0 до 6)
$\alpha.getHours()$	возвращает час (от 0 до 23)
$\alpha.getMinutes()$	возвращает минуты (от 0 до 59)
$\alpha.getSeconds()$	возвращает секунды (от 0 до 59)
$\alpha.getMilliseconds()$	возвращает миллисекунды (от 0 до 999)
$\alpha.getTime()$	возвращает миллисекунды (от 1 января 1970 года GMT)
$\alpha.setFullYear(\beta)$	устанавливает год в полном формате
$\alpha.setYear(\beta)$	устанавливает год в кратком или полном формате (в зависимости от браузера)
$\alpha.setMonth(\beta)$	устанавливает номер месяца (от 0 до 11)
$\alpha.setDate(\beta)$	устанавливает число месяца (от 1 до 31)
$\alpha.setHours(\beta)$	устанавливает час (от 0 до 23)
$\alpha.setMinutes(\beta)$	устанавливает минуты (от 0 до 59)
$\alpha.setSeconds(\beta)$	устанавливает секунды (от 0 до 59)
$\alpha.setMilliseconds(\beta)$	устанавливает миллисекунды (от 0 до 999)
$\alpha.setTime(\beta)$	устанавливает миллисекунды (от 1 января 1970 года GMT)

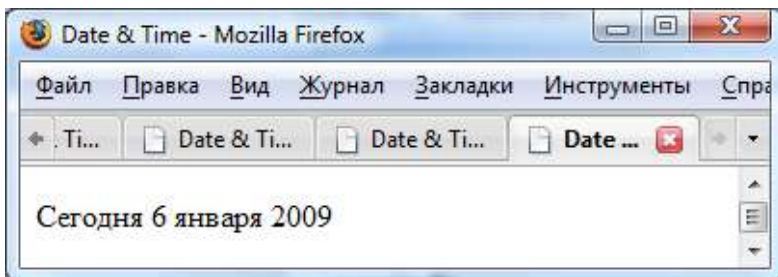
Где α – имя переменной типа Date, β – число.

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", "ок-
тября", "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)
```

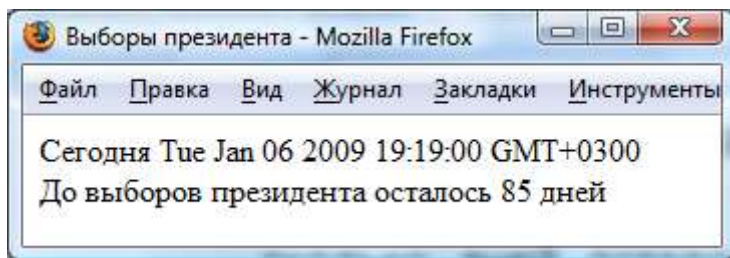
```
сколько_миллисекунд_осталось = выборы_президента -  
сегодня
```

```
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

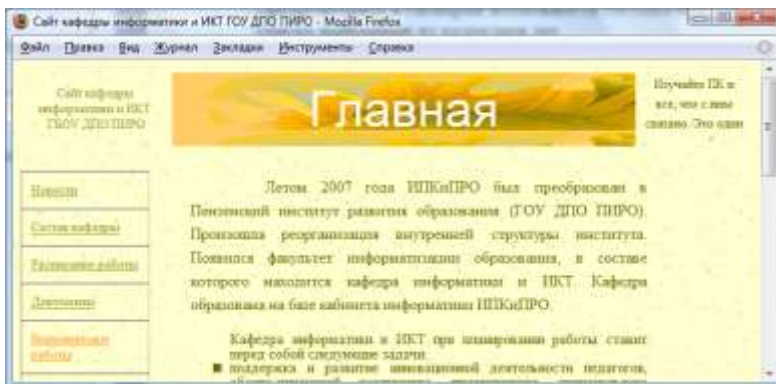
Задание 1. Составьте функцию, которая возвращает текущую дату в формате *день: число месяц: название год: число день недели: название*

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.

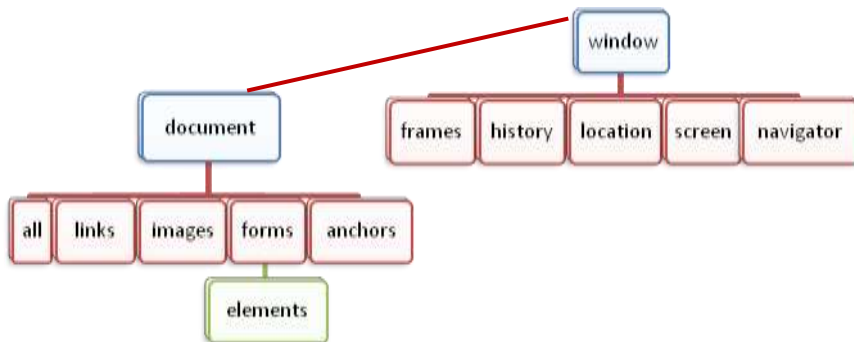
Задание–проект 4. Напишите сценарий, который выводит в таблице календарь на один месяц. Название месяца и год выбирает посетитель из соответствующего размещенного на веб-странице списка. Предусмотрите в сценарии возможность составления посетителем списка выбранных из календаря дат.

Задание–проект 5. Напишите сценарий для всех страниц сайта, который размещает на каждой из них верхний заголовочный баннер и левую навигационную панель. Баннер для каждой странице содержит соответствующий заголовок.



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 параграфе 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.

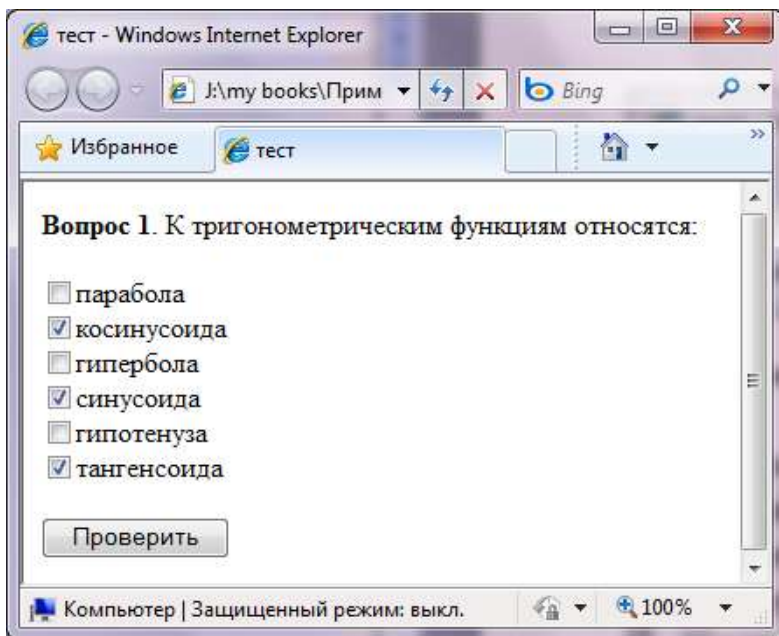
window.document.forms[0].elements[2].checked

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"> гиперболa <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>
```

можно document.forms[0]

можно doc.elements[3].checked



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

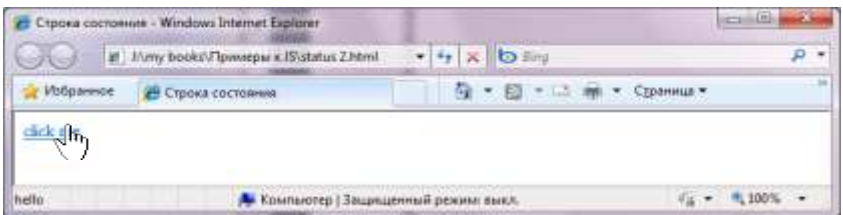
Свойство *closed* возвращает *null*, если окно никогда не открывалось; значение *true*, если окно закрывается; *false* – еще открыто.

Примеры

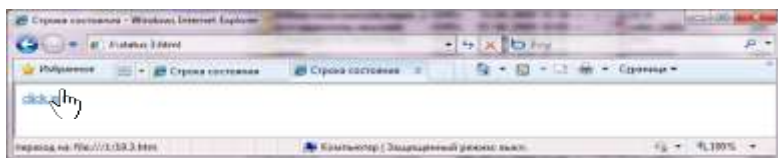
`<body onLoad="status='Добро пожаловать'">`



` click me `




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



Методы

1. open

$$window.open (" \alpha "[, " \beta "[, " \gamma "[])$$

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 не указанные 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, loca-
tion=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; $\gamma...$ – необязательные аргументы.

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

4. clearInterval

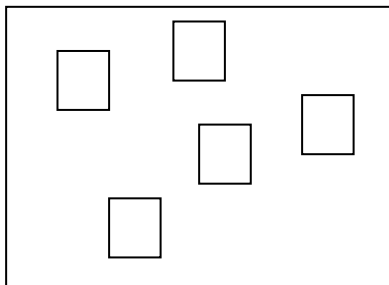
window.clearInterval (ε)

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; $\gamma...$ – необязательные аргументы.

Метод задаёт 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.

$\gamma = \text{window.prompt} (\alpha [, \beta])$

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”.

$\gamma = \text{window.confirm} (\alpha)$

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)">

</a>
```

Объект *Location*

В объекте *location* представлен адрес (URL) загруженного HTML-документа. Существует возможность записывать в *location.href*

свои значения (см. пример на стр. 29). 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

<i>protocol</i>	строка, содержащая начальную часть URL, до двоеточия включительно
<i>hostname</i>	доменное имя или IP адрес
<i>port</i>	часть URL, содержащая номер порта
<i>pathname</i>	часть URL, содержащая путь
<i>search</i>	строка, содержащая любую информацию запроса, присоединенную к URL, начинающаяся с вопросительного знака.
<i>hash</i>	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. Подробнее о методе смотри на странице 40.

Методы

reload ([α])

Повторная загрузка страницы. Где α – необязательный параметр. Принимает значение “False” для загрузки из КЭШа, “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 ("")
  </script>
</html>
```

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([ $\alpha$ ])
```

перемещение вперед по третьей части журнала.

```
window.history.go( $\alpha$ )
```

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>
```

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)

Объект Document

Объект *document* представляет содержимое веб-документа или фрейма, то есть 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)">
```



<code>alinkColor</code>	возвращает цвет активной ссылки
<code>linkColor</code>	возвращает цвет непосещенной ссылки
<code>vlinkColor</code>	возвращает цвет посещенной ссылки
<code>bgColor</code>	возвращает цвет фона документа
<code>fgColor</code>	возвращает цвет текста документа
<code>lastModified</code>	возвращает дату последнего изменения документа
<code>location</code>	возвращает адрес документа
<code>referrer</code>	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
<code>title</code>	возвращает заголовок документа
<code>cookie</code>	возвращает набор кукиз

Пример

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

Коллекции

<code>all</code>	коллекция всех тегов и элементов, расположенных в теле html-документа
<code>anchors</code>	коллекция всех тегов <a name>
<code>forms</code>	коллекция всех форм (теги <form>) документа

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

Элементы коллекции так же как и члены массива имеют общее 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

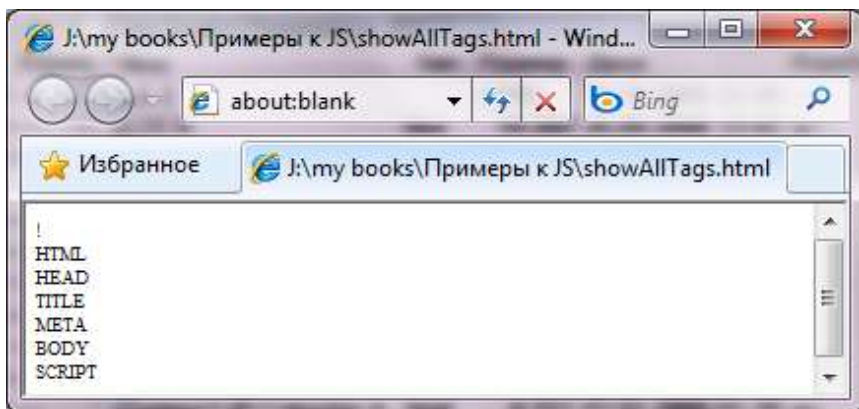
Примеры

```
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

<code>clear ()</code>	очищает документ от текста и дескрипторов
<code>close ()</code>	закрывает все выходные потоки, используемые для записи текста в документ
<code>open ()</code>	открывает все выходные потоки, позволяющие записывать текст в документ
<code>write (α, β, γ, ...)</code>	записывает данные в документ, включая html-теги
<code>writeln (α, β, γ, ...)</code>	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

```

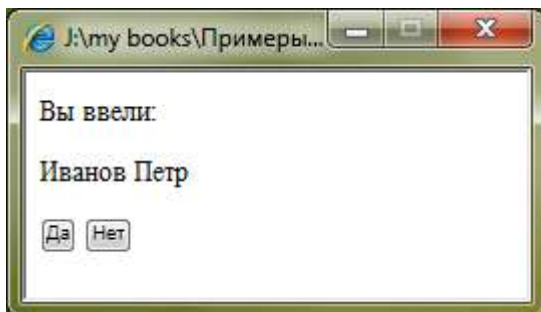
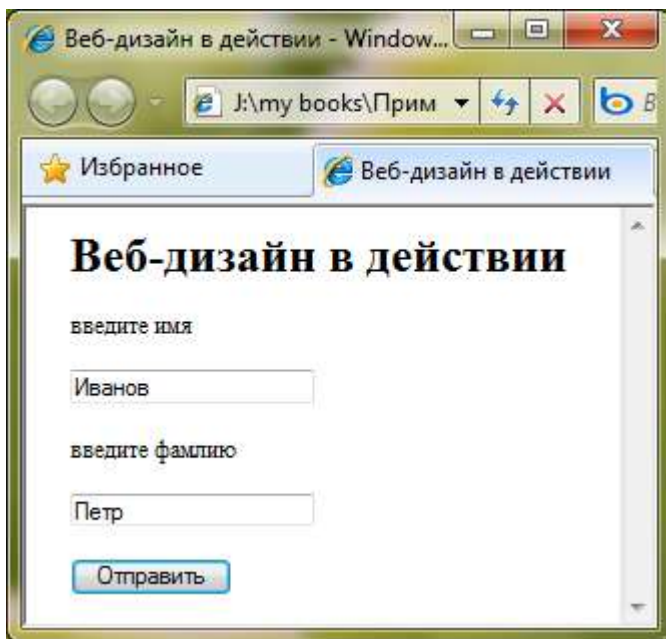
<HTML>
<HEAD>
<TITLE>Веб-дизайн в действии</TITLE>
<SCRIPT type="text/JavaScript">
var w
function send () {
  fio=document.f.n1.value+" "+document.f.n2.value
  if (!window_available())
    w=open("", "k", "width=200,height=150")
  else
    w.focus()
}

```

```

w.document.writeln("<p>Вы ввели:</p>")
w.document.writeln("<p>",fio,"</p>")
w.document.writeln("<input type=button value=Да name=b1
onClick='doSend (this)'>")
w.document.writeln("<input type=button value=Нет name=b2
onClick='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>
<input type=text name=n1>
<p>введите фамлию<\/p>
<input type=text name=n2>
<br><br>
<INPUT TYPE=BUTTON VALUE="Отправить"
onClick="send()">
<\/FORM>
<\/BODY>
<\/HTML>

```



Объект *Image*

Графические изображения на веб-странице являются объектами *image*. Все графические объекты на странице составляют коллекцию графических объектов, то есть 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

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

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("")
        self.resizeTo(w, h)
        self.resizeBy(30, 100)
        self.moveTo(ws, hs)
    }
</script>
```

Объект Navigator или война браузеров

Известно, что для отображения веб-страниц предназначены браузеры. Наиболее популярными «отображателями» веб-страниц являются:





1. Microsoft Internet Explorer (пер. с англ. Explorer – путешественник, исследователь),

2. Mozilla Firefox (пер. с англ. Firefox – огненная лиса, в честь которой, по словам разработчиков, и назван браузер),
3. Opera (пер. с англ. Opera – опера)
4. Apple Safari (пер. с арабск. Safari – поход, путешествие, турпоездка, длительное путешествие в экзотические страны),
5. Konqueror (Конкерор, от англ. *Conqueror* + KDE — завоеватель, победитель),
6. Google Chrome (пер. с 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, решаемых 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 Sa- fari		WebKit	Adobe Integrated Runtime (AIR) — платформо- независимая среда для запуска прило- жений,
Google Chrome			Android — платфор- ма для мобильных телефонов, LeechCraft — кросс- платформенный мо- дульный интернет- клиент с плагином- браузером Poshuku

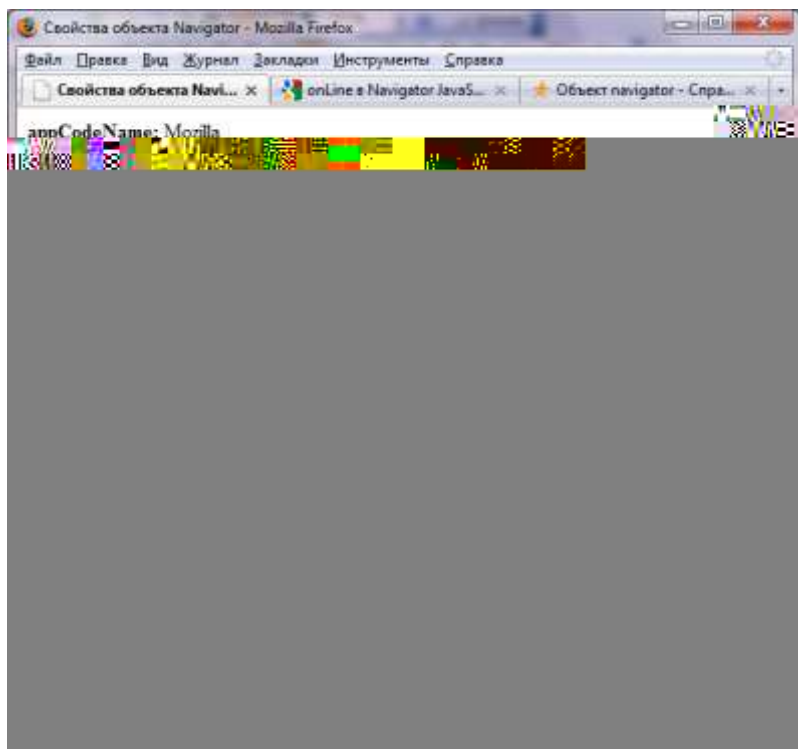
Все браузеры стремятся поддерживать стандарты разработки веб-страниц, такие как HTML, CSS, JavaScript, DOM, форматы отображения веб-графики. Однако, в реальности существует различие в поддержке веб-стандартов различными браузерами. Кроме того, каждый движок «понимает» отличные от стандарта методы, «непонятные»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

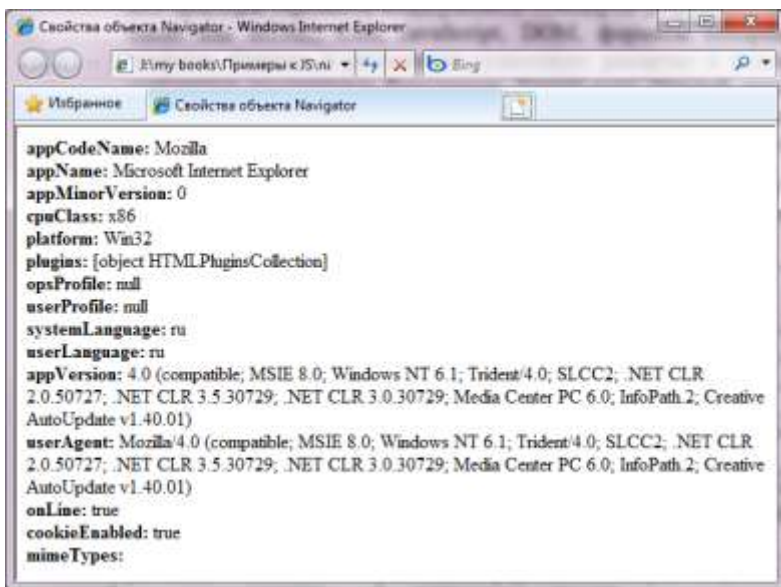
В JavaScript существует объект Navigator, который хранит информацию о браузере посетителя (клиента) текущей веб-страницы. В таблице собраны свойства, общие для нескольких браузеров.

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 на странице 15.





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, scroll-
    bars=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 так далее. 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 поддерживает, а MSIE нет.

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

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, то есть 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 которой 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 куки. По умолчанию имя домена есть имя сервера, сгенерировавшего куки. 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

`docu-`

`ment.cookie="fileN="+document.forms[0].elements[1].value`

`document.cookie="test=comp1"`

`docu-`

`ment.cookie=document.f.n1.name+"="+document.f.n1.value+";`

`expires=Monday, 19-May-2011 23:45:00 GMT"`

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

$\alpha = document.cookie$

где α – имя переменной. В α записываются все пары *имя = значение*, установленные сервером для данной страницы. Для того чтобы прочитать значение какого-либо параметра куки, необходимо писать код JavaScript. В приведенном ниже примере функция `set_cookie ()` записывает в куки все значения элементов формы, а функция `get_cookie (n)` позволяет считывать куки и выделять из всех полей значение заданного в аргументе параметра. Функция `check_cookie()` проверяет существование первого поля куки 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="n1">
        <p>введите фамилию</p> <input type="text" name="n2">
        <p>введите город</p> <input type="text" name="n3">

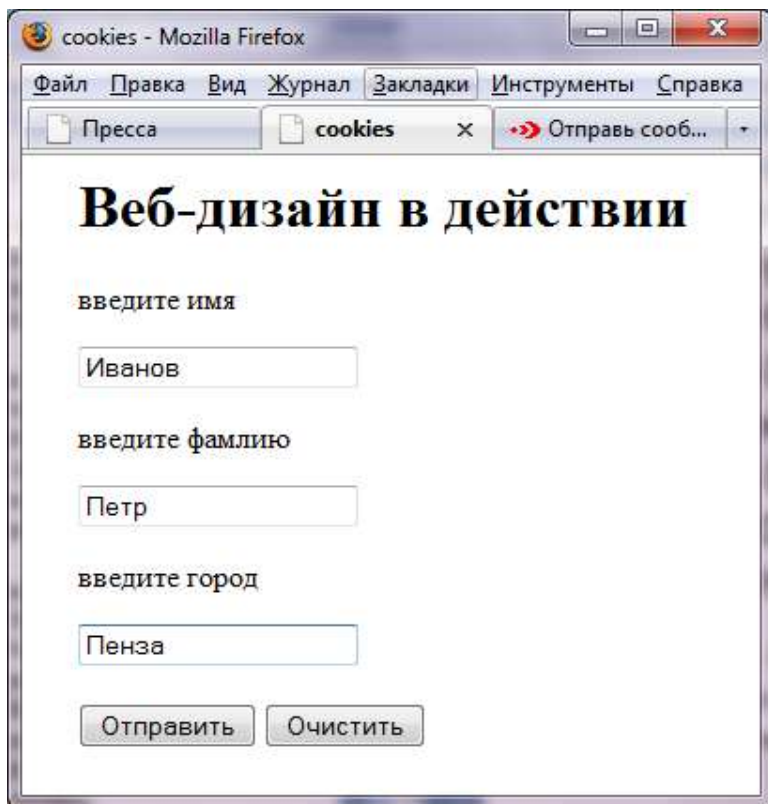
```

```

<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.



Практическая работа 4

Задание 1. Создайте на веб-странице графическую гиперссылку и с помощью скрипта JavaScript сделайте ее интерактивной, то есть когда мышинный курсор будет проходить над ней и при щелчке мыши, изображение должно меняться.

Задание 2. Ролlover. 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. Составьте список ссылок (либо с помощью тега `/`, 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. Изучите способы организации массивов в JavaScript. Создайте массив из 10 изображений и реализуйте их последовательный просмотр на веб-странице с помощью гиперссылок «вперед», «назад»



[вперед](#) [назад](#)

Задание 5. Создайте веб-страницу с большой коллекцией миниатюрных фотографий (5-10 штук). Каждую миниатюру сделайте гиперссылкой, открывающей в центре экрана окно с той же фотографией оригинального размера. Для открытия окна используйте метод `open()` объекта `window`, для передачи окну имени файла оригинальной фотографии – свойство `search` объекта `location`, для размещения окна по центру – свойства `availWidth`, `availHeight` объекта `screen`, хранящие значения ширины и высоты экрана монитора.

Задание 6. Графический индикатор. Создайте на веб-странице работающие электронные часы с графическим индикатором



Задание 7. 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 8. Фотоальбом. С помощью CSS оформите фотографии альбома в красивую графическую рамку. Напишите java-скрипт, который выведет на веб-страницу в несколько строк и столбцов оформленные в красивую рамку фотографии.

Задание 9. Слайдшоу. Напишите скрипт, который будет организовывать просмотр фотографий в виде слайдшоу. В верхней части веб-страницы расположите прокрутку миниатюр, в основной части – оригинальный (или больший) размер выбранной фотографии. С помощью CSS сделайте выделение выбранной фотографии.



Задание 10. 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 5

Задание 1. Создайте бегущую надпись в статусной строке браузера.

Задание 2. Создайте веб-страничку с тестом по математике и напишите скрипт на JavaScript для проверки правильности ответов.

Задание 3. 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. В веб-страницы, созданные ранее, добавьте сценарий JavaScript, выводящий в диалоговом окне тег элемента, по которому щелкает посетитель. Используйте ключевое слово *this*.

Задание 5. Сделайте несколько ячеек таблицы (фон ячейки плюс содержимое) гиперссылками. Используйте свойство *location* объекта *document*.



Задание 6. Для готовой веб-страницы составьте сценарий JavaScript, выводящий в отдельное окно теги всех заголовков html-документа.

Задание 7. Организуйте контролируемый доступ к некоторым страницам сайта. Для этого создайте диалоговое окно для ввода пароля и логина и проверку введенных данных.

Задание 8. Составьте сценарий на JavaScript, выводящий в диалоговом окне текст и внутренние теги элемента, по которому щелкает мышкой посетитель веб-страницы. Веб-страница с именем «17-14.htm» находится на сервере по адресу <http://www.sura.ru/dikov/Mybooks/stuff.htm>






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 кнопки при наведении на нее курсора мыши.

Задание 12. «Градиентная полоска». Создайте сценарий, который создает полоску градиентного цвета. Цвет, ширина и высота полоски может задаваться.

Задание 13. В веб-страницу *X11.htm*, содержащую таблицу с цветовой палитрой веба, добавьте сценарий, 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
	LightPink	#FFB6C1
	Pink	#FFC0CB
	Crimson	#DC143C
	LavenderBlush	#FFF0F5
	PaleVioletRed	#DB7093
	HotPink	#FF69B4
	DeepPink	#FF1493
	MediumVioletRed	#C71585
	Orchid	#DA70D6
	Thistle	#D8BFD8

ТЕКСТ

пробный текст

ТЕКСТ

пробный текст

Задание 14. «Сумашедшее окно». Создайте сценарий, который создает новое окно и перемещает его по рабочему столу таким образом, как будто его трясет.

Задание 15. 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.

Задание 16. Составьте сценарий, который будет осуществлять переход по гиперссылке в зависимости от типа браузера.

Библиография

1. Гудман Д. JavaScript и DHTML. Сборник рецептов. Для профессионалов. СПб: Питер, 2004.
2. Дмитриева М.В. Самоучитель JavaScript. СПб.: БХВ-Петербург, 2001.
3. Дунаев В. Самоучитель JavaScript. СПб: Питер, 2005.
4. Мак-Федрис П. Использование JavaScript. Специальное издание. М.: Вильямс, 2002. 896 с.
5. Мартынов Н.Н. Алгоритмизация и основы объектно-ориентированного программирования на JavaScript. Информатика и ИКТ: профильный уровень. 10-й класс. М.: Бином-Пресс, 2010.
6. Мэрдок К.Л. JavaScript: наглядный курс создания динамических Web-страниц. М.: Вильямс, 2001.
7. Николенко Д.В. Практические занятия по JavaScript. СПб: Наука и техника, 2000.
8. Стандарт ECMA-262, 3я редакция.
<http://www.ecma-international.org/publications/standards/Ecma-262.htm>, 18.06.2009
9. Федоров А. JavaScript для всех. М.: КомпьютерПресс, 1998.
10. Флэнаган Д. JavaScript. Подробное руководство. СПб: Символ-Плюс, 2008.
11. Хольцнер С. Dynamic HTML: руководство разработчика. К.: БХВ, 2000.

Андрей Валентинович Диков (e-mail: *an171@rambler.ru*)
веб-сайт: *http://dikandr.ru*
кандидат педагогических наук

Основы веб-дизайна: JavaScript

Учебное пособие

Сдано в набор 15.01.09. Подписано в печать 29.01.2009.
Формат 60×84 1/16. Печать ризограф. Бумага писчая.
Усл.-печ. л. 4,3. Тираж 100.

Издательство ГБОУ ДПО ПИРО,
440049, Пенза, ул. Попова, 40

Отпечатано с готового оригинал-макета
на ризографе компании ЭЛКОМ

Уважаемые читатели!

Данное пособие охраняется законом об авторском праве. Запрещается распространение копий без разрешения автора.



dikandr.ru

На сайте автора <http://dikandr.ru> в разделе Учебные и методические пособия -> Электронные книги можно приобрести следующие книги

