ТЕМА 3. ОСНОВЫ ПРОГРАММИРОВАНИЯ (НА ЯЗЫКЕ JAVASCRIPT)

В результате освоения этой самой большой темы изучаются основы программирования с использованием языка **JavaScript**. Рассматривается только лишь программирование, без HTML и CSS, так как HTML-дескрипторы и CSS-стили, необходимые для выполнения примеров и заданий, должны быть освоены при построении страниц в предыдущем разделе. Мы вернемся к совместному использованию языка HTML и языка JavaScript при прохождении темы 4, когда будем подготовлены к их объединению на основе объектной модели документа (DOM) в так называемом динамическом HTML (DHTML).

Упражнения, контрольные вопросы, список литературы и веб-источников помещены в конце каждого подраздела.

3.1. Основные понятия.

В этом небольшом подразделе кратко рассмотрим некоторые базовые понятия языка JavaScript. Итак, **JavaScript состоит из следующих основных элементов:**

- литералов и данных;
- переменных;
- подпрограмм-функций;
- объектов и их свойств и методов.
- выражений и операций;
- операторов;
- программ.

Простые элементы, содержащиеся в коде программы, которыми оперируют остальные, более сложные элементы, называют **литералами.** ***Литерал*** — это некоторая последовательность символов (литер), имеющая самостоятельное значение. То есть, символы (литеры) можно сгруппировать так, что они будут литералами (**данными)** какого-либо известного нам типа. В языке JavaScript предусмотрены следующие типы литералов: целого типа; вещественные;

логического типа; строковые литералы и другие.

Данные (литералы) можно сохранять в оперативной памяти, используя **переменные**. **Переменная** должна иметь идентификатор (имя), которое создаётся программистом. После создания переменной за ней автоматически резервируются байты памяти, в которые можно заносить данные путём присвоения значений переменным. При этом в левой части стоит имя переменной, а в правой — **выражение**.

Выражения – это манипуляции (действия или операции) над данными, переменными, функциями, свойствами и методами объектов, в результате которых получается одно-единственное значение. Манипуляции создаются с помощью **операций (или знаков операций)**, например, выражение **`x*10`** увеличивает предыдущее значение **`x`** в **10** раз и для этого используется арифметическая операция умножения *****, целое данное **10** и переменная **x**. В переменную **x** предварительно должно быть занесено любое данное целого или вещественного типа. Выражение должно стоять в правой части **оператора** присваивания.

Операторы – это некоторая синтаксически законченная конструкция языка, аналогичная предложению в русском языке, состоящая из последовательности всех выше перечисленных элементов. Простейший (базовый) оператор — **оператор присваивания**, когда в левой части от **операции присваивания** (=) стоит имя переменной, а в правой — выражение. Пример двух операторов присваивания, приводящих к вычислению целого значения **x**, равного **80**: **x=8; x=x*10;**. Ещё один пример с тремя операторами, в которых используются вещественные значения: **x=8.5; y=x*10+3.5; z=(y-x)*2;** .

Имеются также другие операторы, с помощью которых осуществляется управление вычислительным процессом: оператора ветвления или if (если) и операторов организации циклов.

Программа представляет собой последовательность операторов. Если несколько операторов располагаются на одной строке, то между ними ставится точка с запятой (;) – разделитель операторов. Если оператор размещен на отдельной строке, то разделитель можно не ставить.

Подпрограммы-функции (или просто **— функции)**. - это некоторая группа операторов, создаваемая по определённым правилам и приводящая к вычислению некоторых данных с помощью операторов, сгруппированных в этой функции. Функции присваивается идентификатор (имя), а в скобках указывается входное данное. Функцию можно вызывать для выполнения по её имени. Например, если имеется функция **sin()**, вычисляющая значение синуса угла, заданного в градусах, то с её помощью можно вычислять значения синусов: **x=30; y=sin(x)-sin(30);**.

Объекты можно представить в виде хранилищ некоторых данных (свойств) и функций (методов), создаваемых по определённым правилам. Например, в языке Javascript имеется объект

Math, в котором хранятся данные - математические константы, и функции, вычисляющие наиболее употребимые математические функции. Константы называются свойствами, а функции — методами объекта **Math**. Кроме уже имеющихся объектов, программист может создавать свои собственные объекты, либо дополнять и изменять имеющиеся.

Свойства объектов - это значения, принадлежащие объектам, например, `Math.PI` – значение математической константы π объекта `Math,` представленное с большой точностью.

Методы объектов выполняют некоторые манипуляции над данными. Метод — это функция, «прикреплённая» к объекту. Например, в языке Javascript имеется метод **`Math.sin()`**, вычисляющий значение синуса угла, задаваемого в радианах. Тогда с помощью оператора **`x=Math.sin(3.14)`** переменной **`x`** будет присвоено значение синуса угла, равного 3.14 (угол в радианах, равный приблизительно 180-ти градусам). Используя свойство **`Math.PI`** можно вычислить более точное значение синуса 180-ти градусов: **`y=Math.sin(Math.PI)`**.

Функции, свойства и методы объектов можно использовать при построении выражений. Например: **y=(sin(30)+Math.sin(Math.PI/6))-1.0**. В этом примере предполагается, что функция **sin()** предварительно создана или присоединена из доступной библиотеки. О присоединении объекта **Math** заботиться не нужно – он относится к числу стандартных объектов, встроенных в язык программирования.

Таким образом, функции и объекты с их свойствами и методами позволяют постоянно пополнять библиотеки алгоритмов, созданных к настоящему времени мировым сообществом людей, владеющих программированием.

3.2. Первые программы.

Внимание! При выполнении упражнений настоящей этого и всех последующих подразделов настоящей темы:

• все файлы с выполненными заданиями должны быть помещены в папку exercises и иметь имена exercise_НомерПодразделаНомерПримера.htm, например: exercise_3_2_1.html, exercise_3_2_2.html и так далее;

• Ваша страничка со ссылками на выполненные задания (файл exercises.html), которую Вы

подготовили при прохождении предыдущего раздела, должна быть дополнена ссылками на все страницы с выполненными заданиями настоящего подраздела с краткими описаниями упражнений. Ссылки и описания должны быть вставлены в html-таблицу с двумя колонками: в первой колонке — ссылки, во второй — описание.

3.2.1. Внедрение программ в документ.

Блок `<script>...</script>` - это блок языка HTML, специально предназначенный для указания браузеру, что содержимое блока является программным кодом, которой требуется предварительно интерпретировать.

Существует несколько способов вставки кода языка `JavaScript` в HTML-документ. Рассмотрим два наиболее часто используемых способа.

По первому способу код программы вставляется непосредственно в html-документ. Операторы языка записываются между тэгами `<script>...</script>` и весь этот блок должен находиться в блоке `<head>...</head>` и (или) в блоке `<body> .. </body>`.

Второй способ — это когда межу тэгами `<script>...</script>` ничего не пишется, а в открывающем тэге указывается путь к файлу с программным кодом, например, так:
`<script type="text/javascript" src="./scripts/lib_Petrov.js"></script>`
Такой блок с указанием пути к файлу с операторами языка обычно вставляется в блок `<head>...</head>` перед закрывающим тэгом `</head>`. Файл с операторами языка `Javascript` обычно должен иметь расширение «`js`». Операторы в файле пишутся без каких либо html-тэгов, по правилам написания операторов.

Вначале научимся использовать первый метод вставки программного кода. Например, имеется html-файл, содержащий следующий код:

Листинг 3.2.1.

```
<html><head>
  <title>Студент Петров. Пример 3.2.1.</title>
  <script type="text/javascript" language="javascript">
    //внедрение программы в документ (в контейнер <head>...</head>)
    document.write("<h2>Hello, World! This is Petrov Ivan!</h2>");
  </script>
</head>
<body>
  <h1>Студент Петров. Упражнение 3.2.1</h1>
</body>
</html>
```

Если открыть этот файл в браузере, то на экране должно появиться окно, похожее на рис.3.2.1.

Рис.3.2.1. Результат вставки программы в блок `<head>...</head>`.

Повторите этот пример. Если у вас всё получилось, переходите к следующему примеру. Если не получается, попросите помощи у преподавателя, добейтесь необходимого результата и только после этого идите дальше.

Пример кода для получения того же результата с размещением `Javascript`-кода в блоке `<body> .. </body>` может быть таким:

Листинг 3.2.2.

```
<html><head>
  <title>Студент Петров. Пример 3.2.2.</title>
</head>
<body>
  <script type="text/javascript" language="javascript">
    /*внедрение программы в документ
    (в начале контейнера <body>...</body>)*/
    document.write("<h2>Здравствуй, Мир! Это Петров Иван!</h2>");
  </script>
  <h1>Студент Петров. Упражнение 3.2.2</h1>
</body>
</html>
```

или таким:

Листинг 3.2.3.

```
<html><head>
  <title>Студент Петров. Пример 3.2.3.</title>
</head>
<body>
  <h1>Студент Петров. Пример 3.2.3</h1>
  <script type="text/javascript" language="javascript">
    //внедрение программы в документ
    //в конце контейнера <body>...</body>)
    document.write('<h2>Здравствуй, Мир! Это "программист" Петров Иван!
</h2>');
  </script>
</body>
</html>
```

В первом случае в браузере отобразится страница типа той, которая показана на рис.3.2.2.

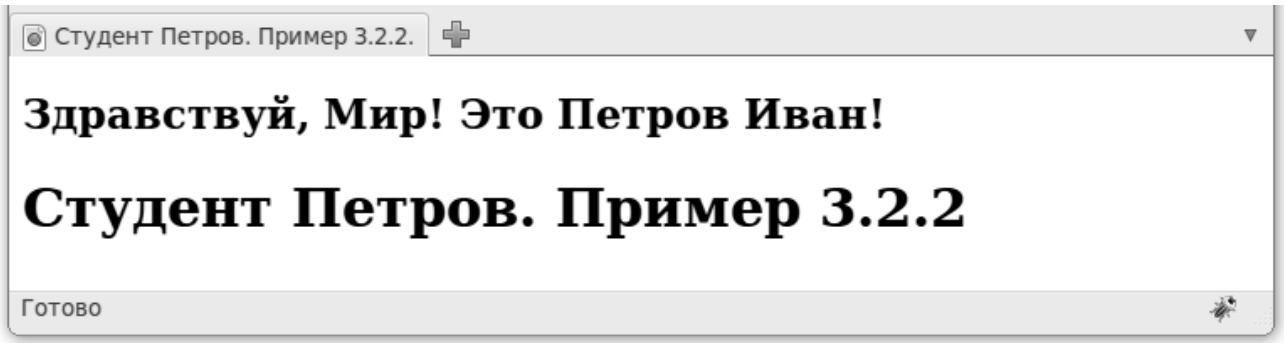

Рис.3.2.2. Результат вставки программы в блок `<body>...</body>` (в начале блока).

Во втором случае в браузере отобразится что-то типа такого:

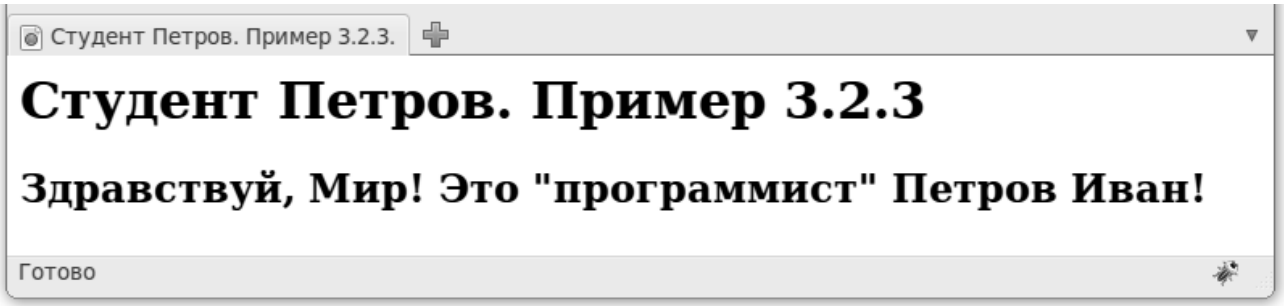

Рис.3.2.3. Результат вставки программы в блок `<body>...</body>` (в конце блока).

Повторите все три примера, но с использованием файла с таблицами стилей, которые были созданы Вами при выполнении упражнений предыдущего раздела. Посмотрите, - в чём разница отображения каждой страницы?

Из примеров видно, что одним из атрибутов тега `<script>` является атрибут `language`, с помощью которого браузер определяет используемый язык сценариев. Для языка JavaScript значение атрибута равно `“JavaScript”`, для `JScript – “Jscript”`, а для `VBScript – “VBScript”` и так далее. При использовании языка `Javascript` атрибут `language` можно опускать, так как этот язык используется браузерами по умолчанию.

По действующему стандарту атрибут `language` вообще отменён. Вместо него в новых версиях браузеров поддерживается атрибут `type=”text/javascript”`. Для совместимости новых и старых версий браузеров рекомендуется использовать оба атрибута, хотя при использовании языка `Javascript` оба атрибута можно не писать.

Документ может содержать сколько угодно блоков `<script>...</script>`. В этом случае все они последовательно интерпретируются браузером.

3.2.2. Ввод-вывод информации.

В рассмотренных выше примерах для формирования вывода в HTML-страницу используется метод `write()` объекта `document`. В скобках в двойных или одинарных кавычках

помещаются выводимые в документ строки. Строки могут включать в себя тэги HTML, которые не отображаются, а управляют разметкой и форматированием, как обычные HTML-тэги.

При использовании метода `document.write()` в выводимой строке можно использовать тэги, которые не отображаются, а управляют форматированием текста. Такой метод формирования выводимой информации (вручную) трудоёмок и ничем не отличается от вывода непосредственно с использованием тэгов, без использования программирования.

Вывод с использованием программирования более эффективен при формировании выводимой информации программным способом. Это станет понятно в последующих подразделах после изучения основ программирования.

Итак, Вы опробовали один из основных методов вывода информации в документ.

Освоим ещё три более простых метода вывода и ввода информации: **`alert()`, `confirm()`** и **`prompt()`**.

`Alert (предупреждение)` – этот метод служит для вывода информации в окне предупреждения. Рассмотрим пример, расположенный ниже.

Листинг 3.2.4

```
<html>
<head>
  <title>Студент Петров. Пример 3.2.4.</title>
</head>
<body>
   <h3>Пример 3.2.4. Вывод информации с помощью метода alert()</h3>
   <script type="text/javascript">
          alert("Это alert (предупреждение), метод языка JavaScript, с которым
экспериментирует Иван Петров");
   </script>
  <h4>
   Ответы на контрольные вопросы
  </h4>
  <ol>
   <li>Вопрос. Когда на экране отобразится текст, расположенный после вызова метода alert()?
<br/>
      Ответ. ???
   </li>
   <li>Вопрос. Что означают строка, введённая после символов "//"?<br/>
         Ответ. ???
   </li>
  </ol>
</body>
</html>
```

На рис.3.2.4 показано отображение в браузере страницы, имеющей html-код из листинга 3.2.4.

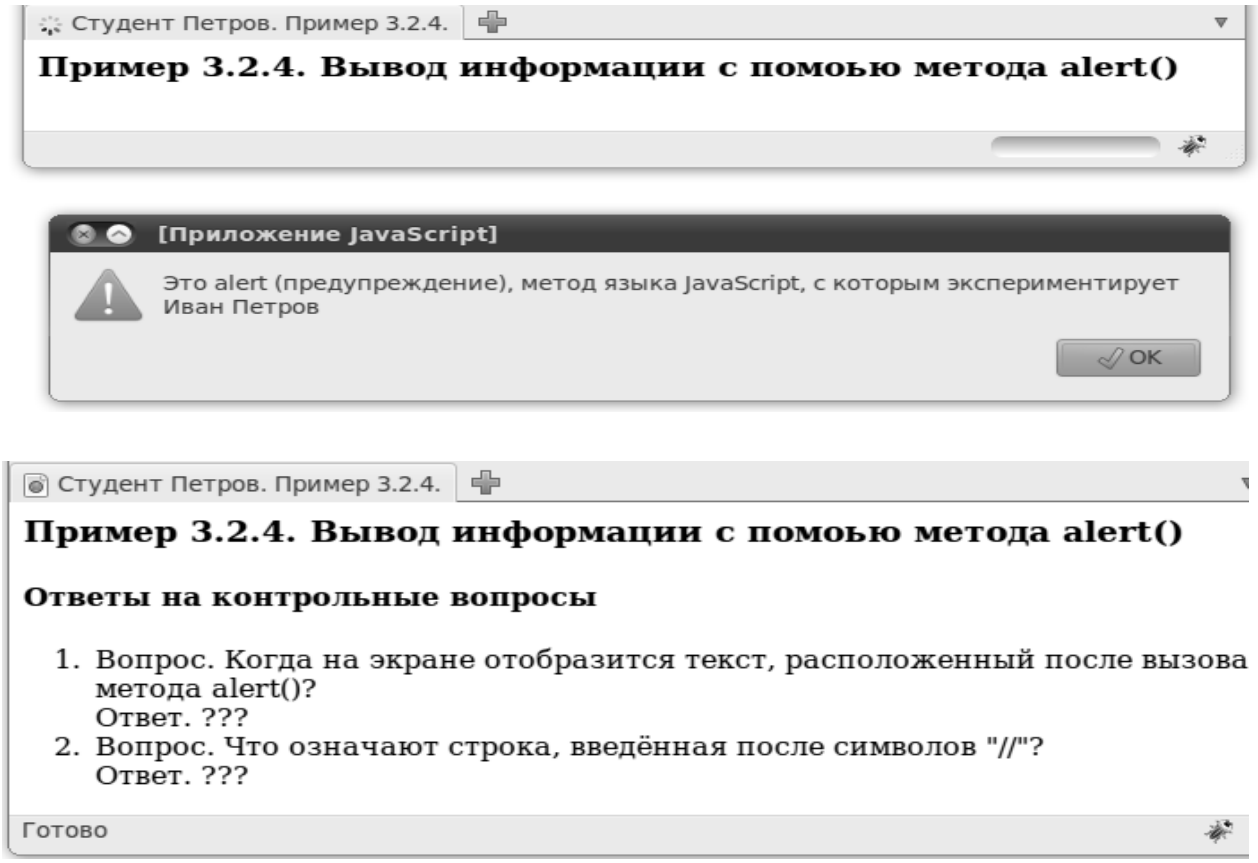

Рис. 3.2.4. Метод `alert()` вывел окно предупреждения в соответствии с кодом листинга 2.2.4.

В самом начале, при открытии страницы, в браузере появляется лишь часть документа, соответствующая html-коду, расположенному до вызова метода alert(). Затем срабатывает оператор вызова метода и появляется окно предупреждения. После появления окна предупреждения загрузка html-кода приостанавливается. После нажатия кнопки “OK”, работа метода `alert()` завершается, окно предупреждения исчезает, страница дозагружается и отображается в виде, показанном в нижней части рис.3.2.4.

`Confirm (подтверждение)` – этот метод предназначен не только для вывода информации, но также позволяет пользователю сделать выбор в форме ответа Да/Нет. В результате выбора метод выдаст логическое значение `true` или `false` и, таким образом пользователь вводит эти данные в программу. Впоследствии будем использовать такой ввод для выбора хода вычислительного процесса.

Листинг 3.2.5

```
<html>
<head><title>Простейшая страница с методом confirm()</title>
<script>
confirm("Это confirm-подтверждение. Сделайте выбор: да/нет")
/* Эксперименты студента Петрова с методом confirm() */
</script>
</head>
<body>
```

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

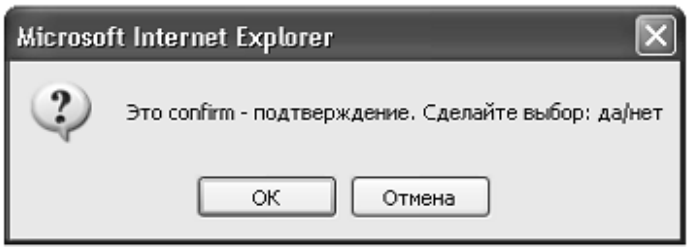

Рис. 3.2.5. Метод confirm() вывел окно подтверждения в соответствии с кодом листинга 3.2.5.

`Prompt (запрос/ответ)` – этот метод позволяет пользователю ввести данные в поле ввода текста. При вызове метода в скобках указываются два строковых параметра: первый — строка, которая будет отображена в диалоговом окне (обычно эта строка используется для пояснения вводимой информации). Вторая строка — значение по умолчанию, которое будет выведено в поле ввода текста. Возвращаемое методом значение зависит от действий пользователя. Если пользователь нажал на **`OK`**, то возвращаемое методом значение равно строке, содержащей текст, введённый пользователем, либо значению по умолчанию, если ничего не вводилось. Если пользователь закрыл окно, либо нажал кнопку **`Отмена`** (**`Cancel`**), то возвращается специальный тип **`null`** (об этом типе данных см. в следующем подразделе).

Листинг 3.2.6.

```
<html> <!-- это - комментарий в языке html. Он может быть и многострочным -->
<head>
<title>Простейшая страница с использованием метда prompt()</title>
<script>
// это - однострочный комментарий в языке JavaScript
prompt("Как вас зовут? (введите свои данные вместо данных по умолчанию)",
"Петров Иван Петрович");
/* а это - многострочный комментарий
в языке JavaScript */
</script>
</head>
</html>
```

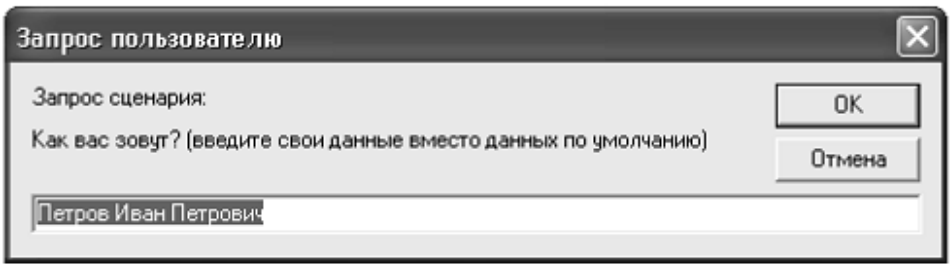

Рис. 3.2.6. Метод `prompt()` вывел окно запрос/ответ в соответствии с кодом примера 3.2.6.

Поэкспериментируйте с примерами, приведенными в листингах, демонстрирующих примеры использования методов ввода-вывода информации.

3.2.3. Комментарии.

Самая простая конструкция любого языка – **комментарии**, которые используются разработчиками программ для того, чтобы облегчить чтение собственного кода при повторном обращении к нему, вставив строки с описанием выполняемых действий. В `JavaScript` есть возможность использовать однострочные и многострочные комментарии.

Однострочный комментарий создаётся вставкой двух подряд идущих прямых слэджа (`//`) в начале строки, которую надо убрать из программы, но так, чтобы эта строка осталась среди операторов программы. Таким образом скрывают от показа в окне браузера следующий за этими символами текст.

Чтобы сделать комментарием сразу несколько строк текста, нужно ограничить его символами `/*` … `*/`. Такой комментарий называют **многострочным комментарием**.

Образцы комментариев приведены в листингах с примерами. Поэкспериментируйте с комментариями обоих типов. Вспомните, что комментарии используются и в языке HTML, но там для этого используются другие символы. Какие? А как задаются комментарии в таблицах стилей?

3.2.4. Упражнения.

В качестве упражнений повторите все 6 примеров, приведённые в листингах 3.2.1- 3.2.6, и при этом:

1) вместо «Петров Иван» должны отображаться Ваши фамилия и имя;

2) к каждому html-документу с упражнением должен быть подключён css-файл с таблицей стилей, разработанной при освоении предыдущего раздела;

3) на каждой странице-упражнении должна быть строка меню со ссылками для возврата на страницу со списком упражнений и на «домашнюю»;

4) дополните каждую веб-страницу с упражнениями нумерованным списком с ответами на контрольные вопросы: первую страницу — ответами на первые два вопроса; вторую — на третий и четвёртый вопрос и так далее, ответив в последнем упражнении на последние два вопроса. Вопрос также необходимо показать и он должен стоять перед ответом.

4) файлы с упражнениями должны быть помещены в папку `exercises` и иметь имена: `exercise_3_2_1.html` - `exercise_3_2_6.html`;

5) таблица со списком-ссылками (файл `exercises.html`) должна быть дополнена строками со ссылками на эти шесть упражнений;

Пример выполнения упражнений смотрите на сайте студента Петрова.

3.2.5. Контрольные вопросы.

1. Когда при загрузке html-страницы в браузере на экране отобразится текст, расположенный после вызова метода `alert()`?

2. Что означает строка, введённая после символов "`//`"?

3. Выполнится ли оператор `document.write("Петров Иван")`, если он будет помещён

внутри тэгов `<h1><script>...</script></h1>`?

4. Что первым появится на экране, если посреди текста html-документа имеется скрипт с оператором `confirm("Петров Иван")`: 1) первая половина текста; 2) весь тест; 3) вторая половина текста; 4) текст, выводимый с помощью метода `confirm()`?

5. Где появится текст, выводимый с помощью оператора `document.write("Петров Иван")`, если этот оператор введён в блоке `<head>...</head>`?

6. Что означают строки, введённые между символами "/* и */"?

7. Выполнится ли оператор `prompt("Петров Иван")`, и если да, то что появится по умолчанию в поле ввода данных?

8. Какая строка появится в поле ввода при выполнении оператора:
`prompt("Как вас зовут?","Введите свои данные")`?

9. Что появится на web-странице, если html-код этой страницы содержит скрипт со строкой:
`//document.write("<h2>Петров Иван</h2>");` ?

10. Что появится на web-странице, если html-код этой страницы содержит скрипт с оператором:
`document.write("<!--<h1>Здравствуй, Мир! Это Петров!</h1>-->");` ?

11. Что появится по умолчанию в поле ввода данных при выполнении оператора
`prompt("Петров Иван","")`?

12. Что появится на web-странице при срабатывании оператора `document.write(<h3>Петров Иван</h3>)` ?

3.2.6. Литература и веб-источники к подразделу.

[2] Дронов В.А. Javascript в Web-дизайне. – СПб.: БХВ-Петербург, 2004. - 880 с.

[3] Кингсли-Хью Э., Кингсли-Хью К. JavaScript 1.5: учебный курс. – СПб.: Питер, 2001. – 272 с.

[4] Полупанов В.Н. Программирование на стороне клиента. Конспект лекций. Керчь, КГМТУ, 2006. - 280 с.

Веб-источники

[6] http://pvn.ho.ua/pvn.org.ua/ – лекции на сайте преподавателя.

[8] http://analog.com.ua - HTML - справочник (наиболее простой).

[9] http://css.manual.ru/ - CSS-справочник (наиболее простой).

3.3. Данные, переменные и константы.

Внимание! При выполнении упражнений, содержащихся в настоящем разделе:

- все файлы с выполненными заданиями должны быть помещены в папку exercises и иметь имена exercise_НомерПодразделаНомерПримера.htm, например: exercise_3_3_1.html, exercise_3_3_2.html и так далее;
- Ваша страничка со ссылками на выполненные задания (файл exercises.html), которую Вы подготовили при прохождении предыдущего раздела, должна быть дополнена ссылками на все страницы с выполненными заданиями настоящего подраздела с краткими описаниями упражнений. Ссылки и описания должны быть вставлены в html-таблицу с двумя колонками: в первой колонке — ссылки, во второй — описание.

3.3.1. Типы данных.

В JavaScript используются следующие основные типы данных:

- строка;
- число;
- логическое или булево значение;
- специальные типы данных.

Строковые литералы (строки) заключаются в двойные или одинарные кавычки. Строка – это связанный набор символов, таких как буквы, знаки препинания и цифры. Чаще всего строки представляют собой какой-либо текст, например: `'Республика Крым'.`

Числа, в программах на языке JavaScript могут быть двух типов: целые числа и числа с плавающей точкой (вещественные).

Хотя чаще всего вы будете использовать числа, записанные в десятичной системе счисления, но могут применяться также числа в восьмеричной и шестнадцатиричной системе счисления.

Вещественные числа могут представляться с дробной десятичной частью (3.141592653589), либо в экспоненциальном виде (3.14e2 или 3.14E2). При экспоненциальной записи числа символ `“e”` (или `“E”`) означает «10 в степени». Числа в JavaScript могут быть от 10^{308} до 10^{-308} (нуль целых с тремястами семью нулями и единицей после запятой).

Существуют также специальные значения, которые используются в некоторых случаях вместо чисел:

- **`NaN`** (несуществующее число), свидетельствует об обращении к несуществующему числу и

при выполнении недопустимых арифметических операций (например, при делении 0 на 0);

- **`Infinity`** — такое значение присваивается, при превышении максимально допустимых значений числового типа при выполнении арифметических операций (например, при делении числа на 0).

Булевы значения могут принимать лишь два значения: `true` (истина) и `false` (ложь). К булевым выражениям мы вернемся в подразделе 3.5. Здесь же вам нужно запомнить, что такой тип данных существует в `JavaScript`, а также, что наименование этот тип данных получил в честь английского математика Джорджа Буля (1815-1864).

Специальные типы данных:

- **`null`** (неопределенный тип), свидетельствует о полном отсутствии каких-либо данных;
- **`undefined`** (неопределяемый тип), чаще всего свидетельствует об ошибке в вашей программе.

Следует обратить особое внимание на различия в представлении числовых и строковых данных. Строковые данные заключаются в кавычки, а числа записываются без кавычек. Например, запись `“-3.14”` - это данное является текстом, а запись –3.14 представляет вещественное число. Это данные различных типов, которые нельзя смешивать в операциях, а иначе могут появиться трудно обнаруживаемые ошибки. С точки зрения языка программирования числа (точнее, данные числового типа) можно корректно использовать в арифметических операциях, а строки (данные строкового типа) – в строковых операциях.

3.3.2. Переменные и константы.

Переменные имеют огромное значение в любом языке программирования. Переменная – это имя (идентификатор), присваиваемое ячейке памяти компьютера, которая хранит определенные литералы во время исполнения программы на JavaScript. Имена переменных не могут начинаться с цифры и содержать пробелы.

Чтобы объявить (создать) переменную в языке JavaScript используется оператор **`var`**, вслед за которым указывается имя, которое вы хотите присвоить переменной, например: `var a1, a_1, b;`. В этом примере созданы три переменные.

Переменной при создании можно присвоить начальное значение: `var a1=314, a_1=3.14, b='Crimea';`. В этом примере созданы три переменные и им присвоены значения: первым двум переменным — числа, переменной `b` — строка.

Знак «**=**» называется **операцией присваивания**. С помощью операции присваивания создаются **операторы присваивания**: в левой части — переменная, в правой — то, что заносится в эту переменную.

Переменную можно образовать и без ключевого слова **`var`**, простым присваиванием:

`a1=314;.`

Область действия переменных — это область, где могут использоваться эти переменные в программе. В `JavaScript` существует две области действия переменных:

- **глобальная**, когда переменную можно использовать в любом месте внутри программы, в том числе и внутри всех функций, используемых в программе;
- **локальная**, когда переменную можно использовать только внутри той функции, в которой эта переменная создана (объявлена).

Чтобы объявить локальную переменную внутри функции, необходимо использовать ключевое слово `var`.

Чтобы объявить глобальную переменную, объявите переменную просто присваиванием (или в функции или вне функции).

Рекомендуется объявлять переменные в одном месте, лучше - в начале программы и функции, и потом их использовать, так, чтобы легко было определить, какие переменные используются в программе.

Следует знать о переменных также следующее:

- в именах переменных можно использовать символы как верхнего и нижнего регистра, так и их сочетание;
- до объявления переменной ее значение имеет неопределенный тип (undefined);
- имя переменой не может начинаться с цифры;
- в именах переменных недопустимы пробелы; если необходим разделитель, используйте символ подчеркивания «_»;
- в качестве имен переменных нельзя использовать зарезервированные слова `JavaScript` (например: `var, const`).

Константы определяются при помощи ключевого слова `const`, например: `const pi=3.141592653589, e=2.718281828459045;`

Отличие константы от переменной состоит в том, что переменной можно присваивать данные много раз и каждый раз прежнее значение в переменной будет заменяться новым, а значение константы изменить невозможно.

3.3.3. Преобразование типов.

Для преобразования строк в числа в `JavaScript` предусмотрены встроенные функции **`parseInt()`** и **`parseFloat()`**. Можно также использовать встроенную функцию **`Number()`**. Вы еще не знакомы с понятием «функция», поэтому пока что можно считать, что это выражение с круглыми скобками, в которых в качестве параметров указываются преобразуемые данные.

Функция **parseInt(строка, основание)** преобразует строку в целое число в системе счисления по указанному основанию (8, 10, 16). Основание можно не указывать, тогда предполагается десятеричная система счисления (основание 10).

Функция **parseFloat(строка)** преобразует строку в число с плавающей точкой.

Функция **Number(строка)** преобразует строку в число с плавающей точкой или целое, в зависимости от значения строки.

Обратная задача преобразования строк в числа осуществляется с помощью встроенной функции **String(n),** которая преобразует число n в строку.

Функция **IsNaN(значение)** может использоваться для определения того, является ли значение числом. Если указанное значение не является числом, функция возвращает логическое значение true, а иначе – false. При этом эта функция считает числом и строку, содержащую только числа, а также логические значения.

Функция **typeof(значение)** предназначена для определения типа переменной. В зависимости от задаваемого значения функция выдаёт:

- **number** — если значение — число;
- **string** — если строка;
- **boolean** — если логическое значение;
- **undefined** — если значения нет или не определено;
- **object** — если значение является объектом.

Учитывая важность понятия типов данных приведем следующие примеры:

```
Number(“314”)          // равно 314
Number(“-3.14”)        // равно -3.14
String (314)           // равно “314”
String (-3.14)         // равно “-3.14”
parseInt(“3.14”)       // равно 3
parseInt(“-3.14”)      // равно -3
parseInt(“Крым”)       // равно NaN
parseInt(“0xFF”,16)    // шестнадцатиричное число преобразовано в 255
parseFloat(“3.14”)     // равно 3.14
parseFloat(“-3.14”)    // равно -3.14
typeof(“Crimea”)       // равно string
typeof(3.14)           // равно number
typeof(“3.14”)         // равно string
isNaN(314)             // равно false
isNaN(“314”)           // равно false
isNaN(“Крым-2005”)     // равно true
var a=“314”; alert(typeof(a));  // выдаст string
alert(typeof(Number(a)));       // выдаст number
a=314; alert(typeof(a));        // выдаст number
alert(typeof(String(a)));       // выдаст string
```

```
a=3.14; b=String(a);
alert(typeof(a));                // выдаст number
alert(typeof(b));                // выдаст string
```

Как следует из показанных выше примеров, преобразование типов данных и преобразование типов переменных выполняется одинаково. Поэкспериментируйте с каждым из этих примеров. Для того, чтобы проверить правильность срабатывания каждой строки, используйте метод `alert()`, например: `alert(Number("314"))` — выдаст результат для первого примера.

Ниже приведены примеры html-документов, в которые вставлены простые скрипты по определению числового и строкового типов: данных (листинг 3.3.1) и переменных (листинг 3.3.2).

Листинг 3.3.1.

```
<html><head>
<title>Студент Петров. Пример 3.3.1.</title>
</head>
<body>
<h2>Пример 3.3.1.Эксперименты по определению типов данных</h2>
<h3>Вывод окна предупреждения с типом данного, равного 3.14</h3>
<script>
alert(typeof(3.14)); // должно появиться number;
</script>
<h3>Вывод окна предупреждения с типом данного, равного "3.14"</h3>
<script>
alert(typeof("3.14")); // должно появиться string;
</script>
</body></html>
```

Листинг 3.3.2.

```
<html><head>
<title>Студент Петров. Пример 3.3.2.</title>
</head>
<body>
<h2>Пример 3.3.2. Студент Петров. Эксперименты по определению типов
переменных</h2>
<h3>Вывод окна предупреждения с типом переменной, которой присвоено число
3.14</h3>
<script type="text/javascript">
var a;
a=3.14;
alert(typeof(a)); // должно появиться number;
</script>
<h3>Вывод окна предупреждения с типом переменной, которой присвоена строка
"3.14"</h3>
<script>
var b="3.14";
alert(typeof(b)); // должно появиться string;
</script>
</body></html>
```

В листинге 3.3.3 приведён ещё один пример html-документа со скриптами, в которых показано образование, преобразование и определение типов переменных. Вывод результатов

осуществляется с использованием метода document.write().

Листинг 3.3.3.

```
<html><head>
<title>Студент Петров. Пример 3.3.3.</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
</head>
<body>
<h2>Пример 3.3.3. Образование, преобразование и определение типов
переменных</h2>
<h3>Образуем и выводим числовую и строковую переменные:</h3>
<script type="text/javascript">
var num_pi=3.142, str_pi="3.1415"; //образуем числовую и строковую переменные
document.write('<ul>');
document.write('<li>n_pi = '); document.write(num_pi);
document.write('</li>');
document.write('<li>s_pi = '); document.write(str_pi);
document.write('</li>');
document.write('</ul>');
</script>
<h3>Определяем и выводим типы образованных переменных:</h3>
<script>
var type_num_pi, type_str_pi;
type_num_pi=typeof(num_pi);
type_str_pi=typeof(str_pi);
document.write('<ul>');
document.write('<li>тип num_pi = '); document.write(type_num_pi);
document.write('</li>');
document.write('<li>тип str_pi = '); document.write(type_str_pi);
document.write('</li>');
document.write('</ul>');
</script>
<h3>Образуем и выводим две новые переменные, преобразуя две первые
переменные:</h3>
<script>
var i_pi = parseInt(num_pi), r_pi = parseFloat(str_pi); // числовые: целую и
вещественную
document.write('<ul>');
document.write('<li>i_pi = '); document.write(i_pi); document.write('</li>');
document.write('<li>r_pi = '); document.write(r_pi); document.write('</li>');
document.write('</ul>');
</script>
<h3>Определяем и выводим типы вновь образованных переменных:</h3>
<script>
var type_i_pi, type_r_pi;
type_i_pi=typeof(i_pi);
type_r_pi=typeof(r_pi);
document.write('<ul>');
document.write('<li>тип i_pi = '); document.write(type_i_pi);
document.write('</li>');
document.write('<li>тип r_pi = '); document.write(type_r_pi);
document.write('</li>');
document.write('</ul>');
</script>
</body></html>
```

HTML-документ со скриптами, в которых даны примеры образования булевых переменных

и специальных значений, показан в листинге 3.3.4.

Листинг 3.3.4.

```
<html><head>
<title>Студент Петров. Пример 3.3.4.</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
</head>
<body>
<h2>Пример 3.3.4. Образование булевых данных и специальных зачений</h2>
<h3>Образуем и выводим переменные логического типа с логическими
данными:</h3>
<script type="text/javascript">
var x1_logical=true, x2_logical=false; //образуем две логические
переменные
document.write('<ul>');
document.write('<li>x1_logical = '); document.write(x1_logical);
document.write('</li>');
document.write('<li>x2_logical = '); document.write(x2_logical);
document.write('</li>');
document.write('</ul>');
</script>
<h3>Определяем и выводим типы образованных переменных:</h3>
<script>
document.write('<ul>');
document.write('<li>тип x1_logical = ');
document.write(typeof(x1_logical)); document.write('</li>');
document.write('<li>тип x2_logical = ');
document.write(typeof(x2_logical)); document.write('</li>');
document.write('</ul>');
</script>
<h3>Образуем и выводим специальные значения:</h3>
<script>
var x1 = 3.14/0, x2 = 0/0; // математически некорректные операции
document.write('<ul>');
document.write('<li>x1 = 3.14/0 = '); document.write(x1);
document.write('</li>');
document.write('<li>x2 = 0/0 = '); document.write(x2);
document.write('</li>');
document.write('</ul>');
</script>
</body></html>
```

3.3.4. Упражнения.

Упражнение 3.3.1. Повторите пример из листинга 3.3.1, дополнив ещё одним скриптом с определением типа логических данных. Кроме этого, дополните веб-страницу нумерованным списком с ответами на 1-й, 2-й и 3-й контрольные вопросы.

Упражнение 3.3.2. Повторите пример из листинга 3.3.2, дополнив ещё одним скриптом с созданием и определением типа логических данных. Кроме этого, дополните веб-страницу

нумерованным списком с ответами на 4-й, 5-й и 6-й контрольные вопросы.

Упражнение 3.3.3. Повторите пример из листинга 3.3.3 «как есть». Кроме этого, дополните веб-страницу нумерованным списком с ответами на 7-й, 8-й и 9-й контрольные вопросы.

Упражнение 3.3.4. Повторите пример из листинга 3.3.4, дополнив ещё одним скриптом с созданием и выводом констант. Кроме этого, дополните веб-страницу нумерованным списком с ответами на 10-й, 11-й и 12-й контрольные вопросы.

Внимание! Как и прежде Вы должны соответствующим образом оформить выполненные упражнения:

1) вместо «Петров Иван» должны отображаться Ваши фамилия и имя;

2) к каждому html-документу с упражнением должен быть подключён css-файл с таблицей стилей, разработанной при освоении предыдущего раздела;

3) на каждой странице-упражнении должна быть строка меню со ссылками для возврата на страницу со списком упражнений и на «домашнюю»;

4) файлы с упражнениями должны быть помещены в папку `exercises` и иметь имена: `exercise_3_3_1.html` - `exercise_3_3_4.html;`

5) таблица со списком-ссылками (файл `exercises.html`) должна быть дополнена строками со ссылками на эти четыре упражнения.

Пример выполнения упражнений смотрите на сайте студента Петрова.

3.3.5. Контрольные вопросы.

1. Типы данных в языке Javascript?
2. Создание констант?
3. Способы создания переменных?
4. Ключевое слово языка Javascript, предназначенное для образования переменных?
5. Функции для преобразования переменной строкового типа в переменную числового типа?
6. Функции для преобразования переменной числового типа в переменную строкового типа?
7. Функция для определения типа переменной?
8. Функция для преобразования переменной вещественного типа в целочисленную переменную?
9. Специальное значение, которое присваивается переменной при делении числа на ноль?
10. Специальное значение, которое присваивается переменной при делении ноля на ноль?
11. Функция для определения того, является ли значение числом?
12. Можно ли изменить значение константы присваиванием ей нового значения?

3.3.6. Литература и веб-источники к подразделу.

[2] Дронов В.А. Javascript в Web-дизайне. – СПб.: БХВ-Петербург, 2004. - 880 с.

[3] Кингсли-Хью Э., Кингсли-Хью К. JavaScript 1.5: учебный курс. – СПб.: Питер, 2001. – 272 с.

[4] Полупанов В.Н. Программирование на стороне клиента. Конспект лекций. Керчь, КГМТУ, 2006. - 280 с.

Веб-источники

[6] http://pvn.ho.ua/pvn.org.ua/ – лекции на сайте преподавателя.

[8] http://analog.com.ua - HTML - справочник (наиболее простой).

[9] http://css.manual.ru/ - CSS-справочник (наиболее простой).

3.4. Массивы

3.4.1. Понятие массива

В предыдущем подразделе вы научились пользоваться переменными, в которых можно было хранить по одному значению различных типов: строка, число, булево значение, специальное значение.

В этом подразделе вы узнаете о новой разновидности переменных, способных хранить в себе много значений – **о массивах**.

Главное преимущество массивов заключается в том, что правильное их использование значительно упрощает код и помогает избежать создания множества переменных с похожими именами.

Массив - это тип переменной, содержащей не одно, а много значений, называемых **элементами массива**. К любому элементу можно обратиться по имени массива с указанием порядкового номера элемента, называемого **индексом**. Синтаксис обращения к элементу массива имеет вид:

```
имя_массива[индекс]
```

Нумерация элементов массива начинается с нуля. Пусть, к примеру, в качестве имени массива мы придумали `name_arr`. Тогда: первый элемент - это `name_arr[0]`; второй элемент - `name_arr[1]`; **…**; последний элемент - `name_arr[name_arr.length-1]`.

Для вычисления индекса последнего элемента массива в выше приведённом примере использовалась переменная `name_arr.length`, в которой хранится размер массива: **размер массива - это число элементов массива**.

Дело в том, что как только создаётся массив, он становится экземпляром объекта **Array**. Вновь создаваемые массивы наследуют все свойства и методы этого объекта. Если проще, то

свойства объекта - это переменные, а **методы объекта** - это функции.

И свойства и методы вычисляются лишь при указании имени экземпляра объекта, затем ставится точка и имя свойства или метода (при вызове метода следует указывать также параметры в круглых скобках. Вы это уже знаете, так как пользовались методами объекта Math). Таким образом, чтобы получить размер или число элементов массива нужно ввести **`name_arr.length`**.

Индекс может быть задан целым числом, целочисленной переменной или выражением, возвращающим целое число. Ну, целое в качестве индекса это и так ясно. О том, что для вычисления индексов можно использовать переменные и выражения вы уже тоже догадались (name_arr.length-1 - это ведь простое, но выражение с использованием переменной).

Пусть была создана переменная типа массив (далее просто массив) с четырьмя элементами:

```
var arr1=[8,2008,"март","год"]
```

Тогда значение второго элемента массива вы можете извлечь, указав его имя и индекс в квадратных скобках:

```
a=arr1[1].
```

В результате переменной **`a`** будет присвоено значение **2008**. В том, что вы обратились к нужному элементу массива можно убедиться, набрав **`alert(a)`**. Конечно, можно обойтись и без промежуточной переменной: **`alert(arr1[1])`**.

3.4.2. Создание и использование массивов

Существует несколько способов создания массивов. Один способ уже применялся в предыдущем подразделе: **`var arr1=[8,2008,"март","год"]`**.

Рассмотрим ещё два способа создания массивов, которые будут использоваться в заданиях настоящего раздела. Это способы создания нового массива, как объекта, с использованием ключевого слова **`new`**.

При создании любых новых объектов используется следующий синтаксис: **`new Array(параметры)`**.

При этом действуют следующие правила:

• если в качестве параметра передаётся одно число (переменая) целочисленного типа, то создаётся массив, содержащий соответствующее количество пустых (неопределённого типа) элементов;

• если передаётся несколько параметров или один параметр нечислового типа, то создаётся соответствующее количество элементов и эти параметры присваиваются в качестве значений этим элементам массива;

• если параметры не заданы, то создаётся массив нулевой длины, то есть массив пока ещё без

элементов.

Рассмотрим способ создания массива, работу с ним и вывод на экран на примере массива дней недели:

Листинг 3.4.1

```
<html>
<head><title>Пример 3.4.1. Способы создания массивов</title>
</head>
<body>
<h1>Пример 3.4.1. Способы создания массивов</h1>
<script>
//
// первый способ создания массива
//
var days_of_week_russ = new Array(7);
days_of_week_russ[0]="Понедельник";
days_of_week_russ[1]="Вторник";
// остальные дни недели введите самостоятельно
// …
//
// второй способ создания массива
//
var days_of_week_engl = new
Array("Sunday","Monday","Tuesday","Wednesday","Thursday","Friday","Saturday");
//
// далее - несколько примеров вывода значений элементов массива
//
alert(days_of_week_russ[0]); // вывод первого элемента русскоязычного массива
alert(days_of_week_russ[1]); // ... второго ...
i=2;
alert(days_of_week_russ[i]); // … третьего …
alert(days_of_week_russ[2*i-1]); // … шестого …
alert(days_of_week_russ); // вывод всех элементов русскоязычного массива
alert(days_of_week_engl); // вывод всех элементов англоязычного массива
alert("Выводим количество элементов массива days_of_week_russ");
alert(days_of_week_russ.length);
alert("Выводим последний элемент массива days_of_week_russ");
alert(days_of_week_russ[days_of_week_russ.length-1]);
</script>
</body>
</html>
```

Глядя на код строк программы в листинге 3.4.1, запомните то, что необходимо вам для создания переменной типа массив:

• создать переменные, указав с помощью **`new Array()`**, что новые переменные являются массивами. В примере - это две переменные `days_of_week_russ` и `days_of_week_engl`;

• задать размер массива. В примере – размер массивов равен 7-ми): для первого массива размер задан явно (`new Array(7)`), а размер второго массива определяется автоматически по числу значений, присвоенных его элементам.

Таким образом, для первого массива вначале созданы семь пустых элементов, которым впоследствии можно присвоить значения, а для второго массива поочередно для каждого элемента выполнены два действия: создается очередной элемент массива и ему присваивается значение.

Обратите внимание, на то, что JavaScript индексирует элементы (присваивает индексы), начиная с номера **0** (а не с 1).

В примере также используется скрытое свойство **`length`** объекта `Array` для определения длины массива.

Чтобы закрепить усвоение правил создания и использования одномерных массивов вам необходимо выполнить упражнение 3.4.1, которое вы найдёте в пункте 3.4.5. Упражнения.

3.4.3. Двумерные массивы

Массивы, рассмотренные в предыдущем подразделе, являются одномерными. Их можно представить в виде вектора (математический образ) или в виде одной колонки (графический образ).

Каждому элементу массива можно присваивать значение, равное массиву, и таким образом создать двумерный массив. Двумерный массив можно представить в виде матрицы (математический образ) или в виде таблицы (графический образ).

В свою очередь, каждому элементу двумерного массива можно присвоить значение одномерного массива и таким образом получить трёхмерный массив, который можно представить в виде параллелепипеда или куба и т.д. Вам для усвоения последующего материала достаточно научиться работать с двумерными массивами.

Если представить двумерный массив в виде таблицы, то обращение к элементам двумерного массива происходит в соответствии со следующим синтаксисом: **`имя_массива[индекс_строки][индекс_столбца]`**.

Пример создания и использования двумерного массива приведён в листинге 3.4.2.

Листинг 3.4.2

```
<html><head>
<title>Пример 3.4.2. Работа с двумерными массивами</title>
<META http-equiv=Content-Type content="text/html; charset=UTF-8">
<link rel="stylesheet" type="text/css" href="Petrov.css">
</head>
<body>
<h4>Пример 3.4.2. Создание двумерного массива размерностью (4x3) и заполнение его
элементов значениями целых  случайных чисел на заданном интервале</h4>
<script>
var a=-50; b=50;
var arr_2 = new Array(4);
arr_2[0]=new Array(3)
arr_2[1]=new Array(3)
arr_2[2]=new Array(3)
arr_2[3]=new Array(3)

arr_2[0][0]=a+Math.round((b-a)*Math.random());
arr_2[0][1]=a+Math.round((b-a)*Math.random());
arr_2[0][2]=a+Math.round((b-a)*Math.random());

arr_2[1][0]=a+Math.round((b-a)*Math.random());
arr_2[1][1]=a+Math.round((b-a)*Math.random());
```

```
arr_2[1][2]=a+Math.round((b-a)*Math.random());

arr_2[2][0]=a+Math.round((b-a)*Math.random());
arr_2[2][1]=a+Math.round((b-a)*Math.random());
arr_2[2][2]=a+Math.round((b-a)*Math.random());

arr_2[3][0]=a+Math.round((b-a)*Math.random());
arr_2[3][1]=a+Math.round((b-a)*Math.random());
arr_2[3][2]=a+Math.round((b-a)*Math.random());

document.write("<table>");
document.write("<tr><th>№ строки</th>");
document.write("<th>Столбец1</th>");
document.write("<th>Столбец2</th>");
document.write("<th>Столбец3</th></tr>");

document.write("<tr><td>Строка1</td><td>",arr_2[0][0],"</td><td>",arr_2[0]
[1],"</td><td>",arr_2[0][2],"</td></tr>");
document.write("<tr><td>Строка2</td><td>",arr_2[1][0],"</td><td>",arr_2[1]
[1],"</td><td>",arr_2[1][2],"</td></tr>");
document.write("<tr><td>Строка3</td><td>",arr_2[2][0],"</td><td>",arr_2[2]
[1],"</td><td>",arr_2[2][2],"</td></tr>");
document.write("<tr><td>Строка4</td><td>",arr_2[3][0],"</td><td>",arr_2[3]
[1],"</td><td>",arr_2[3][2],"</td></tr>");

document.write("</table>");
</script>
</body>
</html>
```

В браузере код листинга 3.4.2 будет отображаться в виде страницы, показанной на рис.3.4.1. При обновлении страницы числа будут меняться.

Рис. 3.4.1. Вывод в документ значений двумерного массива в виде таблицы. Значения округлены до целых с использованием метода Math.round().

В примере из листинга 3.4.2 вы познакомились ещё с одним полезным приёмом - для округления чисел до целых использован метод **`Math.round()`** Синтаксис метода: **`Math.round(округляемое_число)`**.

В листинге 3.4.2 создаётся и выводится двумерный массив размерностью (4x3). Значения элементов массива создаются с помощью датчика случайных чисел **`Math,random()`**. **`Math,random()`** выдаёт случайное вещественное число, равномерно распределённое на интервале [0,1]. После выполнения выражения `a+(b-a)*Math.random()` может быть получено вещественное случайное число на интервале

[a,b]. Но поскольку нам нужно получить целые числа, было использовано выражение **a+Math.round((b-a)*Math.random())**. В результате мы получаем таблицу случайных целых чисел, равномерно распределённых на интервале [-50,50].

Для округления вещественных чисел с точностью до нужного числа знаков после запятой можно использовать метод **toFixed()**. Синтаксис метода - **имя_переменной.toFixed(точность**), где **точность** — параметр, определяющий число знаков после десятичной точки с округлением последней цифры по правилам округления.

В листинге 3.4.3 числа вычисляются с точностью до сотых. Точность указывается в качестве параметра метода.

Листинг 3.4.3.

```
<html><head>
<title>Пример 3.4.3. Работа с двумерными массивами. Округление чисел методом
toFixed().</title>
<META http-equiv=Content-Type content="text/html; charset=UTF-8">
<link rel="stylesheet" type="text/css" href="Petrov.css" />
</head>
<body>
<h4>Пример 3.4.2. Работа с двумерными массивами. Округление чисел с использованием
метода toFixed() объекта Number</h4>
<script language="Javascript">
var a=-50.00; b=50.00;
var arr_2 = new Array(3);
arr_2[0]=new Array(3)
arr_2[1]=new Array(3)
arr_2[2]=new Array(3)

arr_2[0][0]=a+(b-a)*Math.random();
arr_2[0][1]=a+(b-a)*Math.random();

arr_2[1][0]=a+(b-a)*Math.random();
arr_2[1][1]=a+(b-a)*Math.random();

arr_2[2][0]=a+(b-a)*Math.random();
arr_2[2][1]=a+(b-a)*Math.random();

document.write("<table>");
document.write("<tr><th>№ строки</th>");
document.write("<th>Столбец1</th>");
document.write("<th>Столбец2</th></tr>");
document.write("<tr><td>Строка1</td><td>",
arr_2[0][0].toFixed(2),"</td><td>",arr_2[0][1].toFixed(2),"</td></tr>");
document.write("<tr><td>Строка2</td><td>",
arr_2[1][0].toFixed(2),"</td><td>",arr_2[1][1].toFixed(2),"</td></tr>");
document.write("<tr><td>Строка3</td><td>",
arr_2[2][0].toFixed(2),"</td><td>",arr_2[2][1].toFixed(2),"</td></tr>");
document.write("</table>");
</script>
</body>
</html>
```

В браузере код примера 3.4.3 будет отображаться в виде страницы, показанной на рис.3.4.3. При обновлении страницы числа будут меняться.

Рис. 3.4.3. Вывод в документ значений двумерного массива в виде таблицы. Значения округлены до сотых с использованием метода toFixed().

В завершение усвоения правил создания и использования двумерных массивов необходимо выполнить упражнение 3.4.2, которое вы найдёте в пункте **3.4.5. Упражнения**.

3.4.4. Ассоциативные массивы

В ассоциативном массиве используются не целочисленные, а строковые индексы. Во многих случаях использовать ассоциативные массивы более удобно, чем обычные, с целочисленными индексами. Логично, например, присвоить элементам ассоциативного массива развёрнутые наименования университетов. Для вашего университета это присвоение может быть таким:

```
var arr_univers=new Array();
arr_univers["KMGTU"] = "Kerch state maritime technological university";
```

Тогда наименование университета можно легко получить, поскольку такой индекс легче запомнить, чем целое число. Например, для вывода в текущем документе это можно сделать так:

```
document.write(arr_univers["KMGTU"]);
```

3.4.5. Упражнения

Упражнение 3.4.1. Используя код листинга 3.4.1 создайте веб-страницу, изменив код так, чтобы вывод осуществлялся с использованием метода `document.write()`, а не метода `alert()`. При этом не забудьте полностью заполнить русскоязычный массив (читайте комментарии в коде примера).

Задание рекомендуется выполнять в такой последовательности:

- вначале воспроизведите полностью пример из листинга 3.4.1 и поэкспериментируйте с ним;
- затем закомментируйте первый оператор с методом `alert()` и вставьте вместо него оператор с методом `document.write()`. Посмотрите, что происходит, убедитесь в правильности того, что сделали и только после этого замените второй `alert()`, и т.д.

Вам необходимо также ответить на первые семь контрольных вопросов. Поместите ответы на контрольные вопросы на страницу с настоящим упражнением.

Не забудьте также про ссылки для возвращения и стилевое оформление, оформив эту страницу также, как вы это делали при выполнении предыдущих заданий. Файл с упражнением должен иметь имя exercise_3_4_1.html.

В таблице на странице со списком выполненных упражнений (файл exercises.html) вставьте очередную строку: в левой ячейке должно быть наименование упражнения (упражнение 3.4.1), в правой — краткая формулировка того, что делается в упражнении.

Выполнив упражнение 3.4.1 вы должны получить примерно такой результат, который можно посмотреть на сайте студента Петрова вашей группы.

Упражнение 3.4.2. Используя код листинга 3.4.3 создайте веб-страницу изменив код таким образом, чтобы в таблице было не 3, а 6 строк и не 2, а 3 столбца со случайными числами. Кроме этого, случайные числа в каждом столбце должны различаться точностью и принадлежать различным интервалам:

- 1-й столбец: вещественные числа; интервал значений - [-300.0;300.0];
- 2-й столбец: вещественные числа; интервал значений — [0.00;10.50];
- 3-й столбец: целые числа; интервал — [0;1020] .

Необходимо также ответить на последние семь контрольных вопроса. Поместите ответы на контрольные вопросы на страницу с настоящим упражнением.

Упражнение необходимо оформить также, как и предыдущее задание.

Выполнив упражнение 3.4.2 вы должны получить примерно такой результат, который можно посмотреть на сайте студента Петрова вашей группы.

3.4.6. Контрольные вопросы

1. Понятие “массив”?

2. Способы создания одномерного массива?

3. Как определить длину одномерного массива?

4. Индекс первого по счёту элемента одномерного массива?

5. Индекс последнего по счёту элемента одномерного массива?

6. Напишите оператор присваивания какого-либо значения первому по счёту элементу одномерного массива.

7. Напишите оператор присваивания какого-либо значения последнему по счёту элементу одномерного массива.

8. Правила создания двумерного массива?

9. Напишите оператор присваивания какого-либо значения элементу двумерного массива, находящемуся во второй строке и в третьем столбце (используя аналогию с таблицей).

10. Напишите операторы присваивания каких-либо значений элементам двумерного массива, находящимся на главной диагонали матрицы размером 4x4 (используя аналогию с двумерной матрицей).

11. Метод в языке Javascript предусмотренный для генерации случайного числа?

12. Метод в языке Javascript предусмотренный для округления вещественного числа до целого?

13. Метод в языке Javascript предусмотренный для округления вещественного числа до вещественного с заданной точностью?

14. Генерация случайного числа на занном интервале?

3.4.7. Литература и веб-источники к подразделу.

[2] Дронов В.А. Javascript в Web-дизайне. – СПб.: БХВ-Петербург, 2004. - 880 с.

[3] Кингсли-Хью Э., Кингсли-Хью К. JavaScript 1.5: учебный курс. – СПб.: Питер, 2001. – 272 с.

[4] Полупанов В.Н. Программирование на стороне клиента. Конспект лекций. Керчь, КГМТУ, 2006. - 280 с.

Веб-источники

[6] http://pvn.ho.ua/pvn.org.ua/ – лекции на сайте преподавателя.

[8] http://analog.com.ua - HTML - справочник (наиболее простой).

[9] http://css.manual.ru/ - CSS-справочник (наиболее простой).

3.5. Операции, операторы и выражения

3.5.1. Оператор присваивания.

Вы научились пользоваться переменными (обычными и массивами) для сохранения их в памяти.

Сохранение в памяти осуществляется с помощью **оператора присваивания**, который конструируется с помощью **операции присваивания** (=). В левой части от операции присваивания должна находиться переменная, а в правой — то, что присваивается (запоминается). В правой части может находиться константа или переменная, которой уже что-то присвоено, а

также выражение. Выражения будут рассматриваться далее в этом подразделе.

Что такое **операторы**? По своему смыслу **оператор** подобен завершенному предложению в английском языке (и других языках тоже). Вы их уже вовсю используете. Все программы на языке JavaScript, независимо от их размеров и сложности, представляют собой **последовательность операторов**. Например, когда вы вызываете в сценарии метод `alert(x)` для вывода на экран значения переменной `x` то это – оператор. Пример другого оператора - строка **`x=x+10`,** в которой значение **`x`** увеличивается на **10** с помощью выражения в правой части и это значение с помощью операции присваивания помещается в переменную **`x`**, в которой до этого хранилось предыдущее значение. Как уже упоминалось, в конце каждого оператора необходимо ставить точку с запятой (**`;`**), разделяющую операторы. Если оператор размещается в отдельной строке, то разделитель можно опустить.

Итак, ещё раз о переменных, константах и операторе присваивания.

Переменная – это имя, присваиваемое группе ячеек памяти компьютера, в которой могут содержаться данные во время исполнения программы. Имена переменных не могут начинаться с цифры, содержать пробелы, точки, слэджи и ещё некоторые символы. В данном пособии мы будем использовать **лишь латиницу и символ подчёркивания**.

Чтобы создать переменную в языке JavaScript используется оператор **`var`,** вслед за которым указывается имя, которое вы хотите присвоить переменной, например: `var a1, a_1, b;`. В этом примере созданы три переменные без присвоения значений.

Переменной при создании можно присвоить начальное значение: `var a1=314, a_1=3.14, b='Crimea'`. В этом примере созданы три переменные и им присвоены значения: первым двум переменным — числа, переменной `b` — строка.

Знак «**`=`**» называется **операцией присваивания**. С помощью операции присваивания создаются **операторы присваивания**: в левой части — переменная, в правой — то, что заносится в эту переменную.

Переменную можно образовать и без ключевого слова **`var`**, простым присваиванием: `a1=314;`.

На этом этапе вам следует знать о переменных также следующее:

- в именах переменных можно использовать символы верхнего и нижнего регистра, при этом «регистр имеет значение»;
- до объявления переменной ее значение имеет неопределенный тип (undefined);
- имя переменой не может начинаться с цифры;
- в именах переменных недопустимы пробелы; если необходим разделитель, используйте символ подчеркивания «_» (`n_var`) или разные регистры (`nVar`);
- в качестве имен переменных нельзя использовать зарезервированные слова `JavaScript`

(например: `var, const`).

Константы определяются при помощи ключевого слова `const`, например: `const pi=3.141592653589, e=2.718281828459045;`

Отличие константы от переменной состоит в том, что переменной можно присваивать данные много раз и каждый раз прежнее значение в переменной будет заменяться новым, а значение константы изменить невозможно.

3.5.2. Арифметические операции и выражения.

Арифметические операции – это элементарные математические действия: сложение **(+)**, вычитание **(-)**, деление **(/)**, умножение **(*)**, остаток от деления **(%)**.

Арифметические выражения строятся с использованием арифметических операций, переменных, констант и математических функций.

Выше перечисленные операции служат для построения выражений с двумя с двумя операндами. Операции с двумя операндами называют бинарными операциями. Есть ещё арифметические операции с одним операндом — унарные. С унарными операциями вы сможете разобраться самостоятельно по справочнику, например этому - http://javascript.ru/manual .

Все арифметические операции известны вам из арифметики и алгебры. Единственная операция, нуждающаяся в пояснении, - остаток от деления нацело. Например, 9 делится на 5 с остатком 4. Иногда эту операцию также называют **взятием по модулю**. При использовании в качестве операндов целых чисел – результат также будет представлять целое число. Но если операндами будут не целые, а вещественные числа (с плавающей точкой), в результате также получится вещественное число, например: 5.5 % 2.2 = 1.1.

Выше упоминались **математические функции**, которые можно использовать в арифметических выражениях. Далее мы научимся самостоятельно создавать необходимые функции. Однако в языке `Javascript` имеются наиболее употребимые математические функции, принадлежащие объекту **`Math`**. Функции, встроенные в объект, называются **методами**. Обращение к методам любого объекта осуществляется с использованием следующего синтаксиса:

`имя_объекта.имя_метода(параметры)`

Обращение к методам объекта Math выполняется, соответственно, так:

`Math.имя_метода(параметры)`

Например, **`a1=Math.sqrt(2)`** приведёт к вычислению корня квадратного из двух. Если параметров несколько, то они разделяются запятой, например, выполнение оператора **`a1=Math.pow(a1,5)`**, приведёт к возведению в пятую степень, числа, находящегося до этого в переменной `a1`.

Свойства объекта `Math` — это константы, в которых хранятся наиболее употребимые математические постоянные, например, в **`Math.PI`** хранится число **π**, а свойство **`Math.EXP`** — это число **e** (неперово число или основание натурального логарифма).

Принципы использования объектов и свойств объекта `Math` очень просты. Обо всех объектах и методах этого объекта, предусмотренных в языке Javascript, можно узнать в справочниках, например в этом - http://javascript.ru/manual.

Чтобы убедиться в том, что вы владеете арифметическими операциями и умеете строить арифметические выражения, повторите следующий пример:

Листинг 3.5.1

```
<html><head>
<title>Пример 3.5.1.</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
</head>
<body>
<h2>Пример 3.5.1. Арифметические операции и выражения</h2>
<h3>Вычисление значений синусоиды на интервале от 0 до 90 градусов с шагом 45 градусов)</h3>
<script type="text/javascript">
h_grad=45.0; // шаг в градусах
h_rad=h_grad*Math.PI/180.0; // шаг в радианах
//
//вычисляем первое значение аргумента и функции
//
i=0;
x_grad=0.0+i*h_grad;;
x_rad=x_grad*Math.PI/180.0;;
y=Math.sin(x_rad);
document.write("<p>угол в градусах = "+x_grad+"; угол в радианах = "+x_rad+"; sin("+x_grad+") = "+y+"</p>")
//
//вычисляем второе значение аргумента и функции
//
i=1;
x_grad=0.0+i*h_grad;;
x_rad=x_grad*Math.PI/180.0;;
y=Math.sin(x_rad);
document.write("<p>угол в градусах = "+x_grad+"; угол в радианах = "+x_rad+"; sin("+x_grad+") = "+y+"</p>")
//
//вычисляем третье значение аргумента и функции
//
i=2;
```

```
x_grad=0.0+i*h_grad;;
x_rad=x_grad*Math.PI/180.0;;
y=Math.sin(x_rad);
document.write("<p>угол в градусах = "+x_grad+"; угол в радианах = "+x_rad+";
sin("+x_grad+") = "+y+"</p>")
</script>
</body></html>
```

3.5.3. Операции сравнения

Операции сравнения используются для построения логических выражений. В Javascript предусмотрены следующие операции сравнения: меньше **(<)**, меньше или равно **(<=)**, больше **(>)**, больше или равно **(>=)**, равно **(==)**, не равно **(!=)**. Это – операции с двумя операндами.

Сравниваться могут как числовые данные, так и строки. Единственное условие состоит в том, что нужно сопоставлять значения, относящиеся к одному и тому же типу. В противном случае JavaScript попытается перевести данные из одного типа в другой, что не всегда удается. Чтобы избежать ошибок сравнивайте данные только одного типа.

Результатом выполнения выражения с использованием операции сравнения является данное логического типа. Например, после выполнения оператора a = 2<3 в переменную a будет занесено значение, равное **`true,`** а при выполнении оператора b = 2>3 в переменную b занесётся **`false`**.

3.5.4. Логические операции и выражения.

В Javascript предусмотрены три логические операции: логическое **И (&&)**, логическое **ИЛИ (||)**, логическое **НЕ (!)**. Первые две - операции с двумя операндами, а логическое НЕ – операция с одним операндом.

Логическое **И** означает, что обе части выражения должны быть истинны.

Логическое **ИЛИ** означает, что, по крайней мере, одна часть выражения должна быть истинной.

Логическое **НЕ** изменяет значение истина/ложь на обратное.

Логические операции позволяют свести воедино результаты сравнений нескольких переменных. Например, после выполнения оператора **`a = (2<3)&&(3<5)`** в переменную a будет занесено **`true`**, а после выполнения **a = (2<3)&&(3>5)** — **`false`**.

Чтобы убедиться в том, что вы владеете построением логических выражений с помощью операций сравнения и логических операций, повторите следующий пример:

Листинг 3.5.2.

```
<html><head>
<title>Пример 3.5.2.</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
</head>
```

```
<body>
<h2>Пример 3.5.2. Логические операции и выражения</h2>
<h3>Вычисление значений функции y=(x>=0)&&(x%2==0)<br>
    для значений аргумента x на интервале от -2 до +2<br>
    с шагом h = 1</h3>
<script type="text/javascript">
var x = new Array(); y = new Array()
x_0=-2;
h=1;
//
//вычисляем первое значение аргумента и функции
//
i=0;
x[i]=x_0+i*h;
y[i]=(x[i]>=0)&&(x[i]%2.0==0)
document.write("<p>x = "+x[i]+"; y = "+y[i]+"</p>")
//
//вычисляем второе значение аргумента и функции
//
i=1;
x[i]=x_0+i*h;
y[i]=(x[i]>=0)&&(x[i]%2.0==0)
document.write("<p>x = "+x[i]+"; y = "+y[i]+"</p>")
//
//вычисляем третье значение аргумента и функции
//
i=2;
x[i]=x_0+i*h;
y[i]=(x[i]>=0)&&(x[i]%2.0==0)
document.write("<p>x = "+x[i]+"; y = "+y[i]+"</p>")
//
//вычисляем четвёртое значение аргумента и функции
//
i=3;
x[i]=x_0+i*h;
y[i]=(x[i]>=0)&&(x[i]%2.0==0)
document.write("<p>x = "+x[i]+"; y = "+y[i]+"</p>")
//
//вычисляем пятое значение аргумента и функции
//
i=4;
x[i]=x_0+i*h;
y[i]=(x[i]>=0)&&(x[i]%2.0==0)
document.write("<p>x = "+x[i]+"; y = "+y[i]+"</p>")
</script>
</body></html>
```

3.5.5. Операции над строками

В приведённых выше примерах использовался метод `write()` объекта `document` для вывода на экран результатов, получаемых в программе. В качестве входных данных метода (в круглых скобках) формировались строки из переменных и строковых констант (то, что в кавычках). Для формирования нужного текста использовалась **операция конкатенации**, которая выглядит

также, как и арифметическая операция суммирования - «**+**».

В Javascript предусмотрено много средств для работы со строками. Под строками понимаются строковые переменные и константы. Набор методов и свойств для работы со строками хранится в объекте **String**. Так, для определения длины строки (количества символов, составляющих строку) можно использовать свойство **length**. Например, в результате срабатывания операторов: str="АР Крым"; document.write(str.length) на экран будет выведено число 7 (учитывая, что пробел также является символом).

Описания и примеры использования всех методов объекта **String** вы можете взять в справочниках, например в этом - http://javascript.ru/manual.

Далее кратко опишем несложные правила применения и примеры использования **операции конкатенации**, которым вам обязательно необходимо уметь пользоваться. Рассмотрим следующий пример.

Листинг 3.5.3.

```
<html><head>
<title>Пример 3.5.3.</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
</head>
<body>
<h2>Пример 3.5.3. Операции со строками: операция конкатенации</h2>
<script type="text/javascript">
stroka_1="студент";
stroka_2=prompt("Введите имя","Иван");
stroka_3=prompt("Введите фамилию","Петров");
stroka_1=stroka_1+" "+stroka_2+"_"+stroka_3;
document.write("<h2>"+stroka_1+"</h2>");
</script>
</body></html>
```

При запуске в браузере скрипт из листинга 3.5.3 сначала занесёт в переменную stroka_1 текст студент. Затем в переменную stroka_2 пользователю необходимо ввести имя, иначе будет введено имя по умолчанию (Иван). Тем же способом вводится фамилия. С помощью оператора конкатенации формируется строка и присваивается переменной stroka_1. Для того, чтобы слова не сливались, при формировании строки в качестве разделителей вставлены пробелы. Далее сформированная строка выводится на экран с помощью метода write(). Входной параметр метода сформирован также с помощью оператора конкатенации — слева и справа от переменной, содержащей сформированную строку, добавлены html-тэги заголовка.

Необходимо учесть, что при использовании операции конкатенации происходит автоматическое изменение типа переменной. Например, оператор x=2+"3" приведёт к тому, что переменной x будет присвоено строковое значение «23», так как если один из операндов является

строкой, даже если эта строка содержит цифры, то второй операнд автоматически преобразуется в строку и выполняется конкатенация строк. Все остальные арифметические операции (-, *, /, %), наоборот: пытаются преобразовать строку в число.

Для закрепления материала настоящего подраздела вам необходимо перейти к пункту 3.5.6 и выполнить два упражнения (упражнение 3.5.1 и 3.5.2).

3.5.6. Упражнения

Упражнение 3.5.1. Повторите пример из листинга 3.5.1, изменив его таким образом, чтобы рассчитывались значения тангенса для значений угла от 0 до 90 градусов с шагом 30 градусов.

В этом же html-документе поместите ответы на первые пять контрольных вопроса (с 1-го по 5-й).

Упражнение необходимо сохранить в файле exercise_3_5_1.html и оформить его отображение на своём сайте так, как это вы делали с предыдущими упражнениями.

Упражнение 3.5.2. Повторите пример из листинга 3.5.2, изменив его таким образом, чтобы рассчитывались значения функции для аргумента на интервале от -3 до +3, причём значения функции были бы истинными лишь для положительных и нечётных значений аргумента.

В этом же html-документе поместите ответы на последние пять контрольных вопроса (с 6-го по 10-й).

Упражнение необходимо сохранить в файле exercise_3_5_2.html и оформить его отображение на своём сайте так, как это вы делали с предыдущими упражнениями.

3.5.7. Контрольные вопросы

1. Оператор для вычисления котангенса?
2. Значение последнего тангенса в упражнении 3.5.1?
3. Значение тангенса для угла 45 градусов в упражнении 3.5.1?
4. Значение котангенса для угла 90 градусов?
5. Значение тангенса для угла в 135 градусов?
6. Число истинных значений, полученных при выполнении упражнения 3.5.2?
7. Число ложных значений, полученных при выполнении упражнения 3.5.2?
8. Число истинных значений, которое могло бы быть получено при замене логического И на логическое ИЛИ в упражнении 3.5.2?
9. Число ложных значений, которое могло бы быть получено при замене логического ИЛИ на логическое И в упражнении 3.5.2?
10. Число истинных значений, которое могло бы получиться при выполнении упражнения 3.5.2, в котором для получения значения функции использовался бы оператор y[i]=!((x[i]>=0)||(x[i]%2.0==0))?

3.5.8. Литература и веб-источники к подразделу.

[2] Дронов В.А. Javascript в Web-дизайне. – СПб.: БХВ-Петербург, 2004. - 880 с.

[3] Кингсли-Хью Э., Кингсли-Хью К. JavaScript 1.5: учебный курс. – СПб.: Питер, 2001. – 272 с.

[4] Полупанов В.Н. Программирование на стороне клиента. Конспект лекций. Керчь, КГМТУ, 2006. - 280 с.

Веб-источники

[6] http://pvn.ho.ua/pvn.org.ua/ – лекции на сайте преподавателя.

[8] http://analog.com.ua - HTML - справочник (наиболее простой).

[9] http://css.manual.ru/ - CSS-справочник (наиболее простой).

3.6. Операторы управления вычислительным процессом.

3.6.1. Оператор if

Оператор **`if`** или **оператор организации ветвлений**, позволяет выбрать и запустить на выполнение одну из альтернативных групп операторов. Выбор осуществляется с помощью булевых значений (`true` или `false`). Рассмотрим, следующий пример использования оператора `if`.

Листинг 3.6.1

```
<html> <head><title>Пример 3.6.1</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
</head>
<body>
<h2>Пример 3.6.1. Оператор if с использованием метода confirm()</h2>
<script type="text/javascript">
var response =
confirm("Вы знаете Javascript? - Нажмите OK. А если нет - Отмена.");
if ( response == true)
  {
```

```
    var str = "Отлично! Знание Javascripta – то, что необходимо"
    alert(str)
  }
  else
  {
    var str = "Жаль! Javascript надо бы знать."
    alert(str)
  }
if (response) document.write("<h3>Результат опроса: " + str + "</h3>")
if (!response) document.write("<h3>Результат опроса: " + str + "</h3>")
</script>
</body> </html>
```

В этом примере, когда вы щелкаете на кнопке OK, метод `confirm()` создаёт булево значение `true`, которое присваивается переменной `response.` Переменная `response` в операторе `if` проверяется на равенство значению `true.` Проверка на равенство выдаёт `true`, поэтому выполняются два оператора , стоящие после условия в фигурных скобках и в результате на экран с помощью метода `alert()`выводится `Отлично! … .` Если же вы нажимаете на `Отмена`, создается и запоминается в переменной `response` булево значение `false`. В этом случае срабатывают операторы, стоящие в фигурных скобках после `else.`

Фигурные скобки «**{**» и «**}**» называются **операторными скобками.** Операторные скобки используются **для объединения операторов в группы**. Легко усвоить смысл фигурных скобок, если вспомнить использование скобок для выделения многочленов в школьной алгебре.

Синтаксис оператора **`if`** можно записать следующим образом:

`if (условие) {операторы1;}else{операторы2;}.`

Работу оператора **`if`** можно описать так: **если `условие истинно (true)`, то выполняются `операторы1`, если `условие ложно (false)`, то выполняются `операторы2`.**

Можно использовать и сокращённый вариант оператора if:

`if (условие) {операторы1;}`.

Работу сокращённого оператора **`if`** можно описать так: **если `условие истинно (true)`, то выполняются `операторы1`, если `условие ложно (false)`, то начинают выполняться операторы, стоящие после оператора `if`**.

В примере 3.6.1 есть два сокращённых оператора **`if`**, выводящих результат опроса с помощью метода `document.write()`. В качестве условия в этих операторах используется непосредственно переменная `response` без использования операции сравнения, поскольку в этой переменной и без того находится логическое значение `true` или `false`. На примере этих двух операторов можно также понять, что, если после условия стоит один оператор, то операторные

скобки можно не использовать. То же самое справедливо для оператора, стоящего после `else`. Например, два усечённых оператора в листинге 3.6.1 можно заменить одним:

```
if (response) document.write("<h3>Результат опроса: "+str+"</h3>")
else document.write("<h3>Результат опроса: " + str + "</h3>")
```

3.6.2. Условный оператор

Условный оператор может заменить оператор if в наиболее простых случаях, когда в качестве альтернатив используется по одному оператору . Например, операторы:

```
x=prompt('ведите целое число',1);
if (x>0) y=Math.log(x); else y=Math.log(Math.abs(x));
```

можно заменить операторами:

```
x=prompt('ведите целое число',1);
y=(x>0) ? Math.log(x) : Math.log(Math.abs(x));
```

Можете убедиться, что результаты будут одинаковыми.

Синтаксис условного оператора можно записать в виде:

`условие ? оператор1 : оператор2`

В результате выполнения условного оператора срабатывает один из двух вложенных в него операторов в зависимости от значения условия: если условие истинно, то выполняется первый оператор, если ложно — второй.

Синтаксис второго варианта условного оператора можно записать так:

`переменная = условие ? выражение1 : выражение2`

В этом варианте записи условный оператор возвращает результат вычисления одного из выражений в зависимости от значения условия: если условие истинно, то вычисляется и присваивается переменной первое выражение, если ложно — второе.

В листинге 3.6.2 приводятся три примера использования условного оператора, которые заменяют один и тот же оператор `if`.

Листинг 3.6.2.

```
<html> <head><title>Пример 3.6.2</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
</head>
<body>
<h2>Пример 3.6.2. Оператор <b>if</b> можно заменить условным оператором</h2>
<script type="text/javascript">
var str;
var str1 = "Отлично! Знание Javascripta – то, что необходимо"
var str2 = "Жаль! Javascript надо бы знать."
var response = confirm("Вы знаете Javascript? - Нажмите OK. А если нет -
Отмена.");
// оператор if
```

```
if (response)
document.write("<h3>Результат опроса (оператор if) : " + str1 + "</h3>")
else
document.write("<h3>Результат опроса (оператор if): " + str2 + "</h3>");
// условный оператор (пример 1)
response ?
document.write
("<h3>Результат опроса (условный оператор1):"+str1+"</h3>") :
document.write
("<h3>Результат опроса (условный оператор1): "+str2+ "</h3>");
// условный оператор (пример 2)
response ? str=str1 : str=str2;
document.write("<h3>Результат опроса (условный оператор2): "+str+"</h3>")
// условный оператор (пример 3)
str = response ? str1 : str2;
document.write("<h3>Результат опроса (условный оператор3): "+str+"</h3>")
</script>
</body>
</html>
```

Повторите этот пример, чтобы убедиться, что вы освоили использование условного оператора.

3.6.3. Операторы организации цикла

Цикл – это повторение последовательности операторов некоторое количество раз до тех пор, пока выполняется некоторое условие. В языке JavaScript существует четыре оператора цикла:

- **`do … while`**
- **`while`**
- **`for`**
- **`for … in`**

Рассмотрим первые два оператора – операторы: **`do … while`** и **`while`**. Оператор цикла **`while`** называют оператором цикла с предусловием, а оператор организации цикла **`do...while`** - оператором цикла с постусловием. Их различие следует из названий.

Оператор `do…while` запускает на выполнение оператор или группу операторов и повторяет эту процедуру до тех пор, пока условие не перестанет выполняться. Синтаксис этого оператора можно записать так:

```
do {группа операторов} while (условие)
```

Группа операторов заключается в **операторные скобки** (в языке Javascript – это фигурные скобки). Если в группу входит лишь один оператор, то **операторные скобки** можно не использовать. Эти операторы, заключённые в операторные скобки (или не заключённые, если лишь один оператор), часто называют **телом цикла**. Понятие “тело цикла” будем использовать во всех операторах цикла. Это определение поясним на следующем примере.

Листинг 3.6.3.

```
<html>
<head><title>Пример 3.6.3</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
</head>
<body>
<h2>Пример 3.6.3. Оператор <b>do...while</b> - оператор цикла с
постусловием</h2>
<script type="text/javascript">
var x = 0
do
  {
    x=x+1;
    alert(x);
  }
while (x < 4);
</script>
</body>
</html>
```

В данном сценарии имеется переменная **x**, которой до цикла присваивается значение, равное нулю. Затем в теле цикла (после **do)** первый оператор увеличивает значение **x** на **1,** а второй выводит результат в окне предупредительных сообщений. В условии (после **while)** указывается, что операторы тела цикла должны повторяться до тех пор, пока значение **x** меньше **4**.

Необходимо заметить, что, если условие с самого начала не выполняется, то операторы, стоящие в теле цикла всё равно один раз выполнятся, поскольку условие проверяется после того, как выполнятся операторы тела цикла (поэтому и название оператора — оператор цикла с постусловием).

В качестве одного из операторов в теле цикла обязательно должен быть оператор, меняющий переменную, входящую в условие так, чтобы рано или поздно условие перестало выполняться (в примере это `x=x+1`). Иначе цикл никогда не завершится и программа «зависнет».

Оператор `while` подобен оператору `do … while` и действует похожим образом, только условие, при котором выполнятся (или не выполнятся) операторы тела цикла, стоит в самом начале (поэтому — оператор с предусловием). В операторе `while`, в отличие от оператора `do … while`, операторы тела цикла могут вовсе не запускаться на выполнение, если условие с самого начала не выполняется. Синтаксис оператора while имеет вид:

`while (условие) {группа операторов}`

Это определение реализовано в следующем примере.

Листинг 3.6.4.

```
<html>
<head><title>Пример 3.6.4</title>
```

```
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
</head>
<body>
<h2>Пример 3.6.4. Оператор <b>while</b> - оператор цикла с предусловием</h2>
<script type="text/javascript">
var x = 0
while (x < 4)
  {
    x=x+1;
    alert(x);
  }
</script>
</body>
</html>
```

Оператор for называется оператором цикла с переменной-параметром. Часто этот оператор называют также оператором цикла со счётчиком. Синтаксис оператора можно записать в виде:

```
for (присвоение_начального_значения_переменной-параметру;
    условие;
    присвоение_следующего_значения_переменной-параметру)
    {группа_операторов}
```

Цикл `for` выполняется до тех пор, пока условие истинно (принимает значение `true`).

Всё, что указывается в круглых скобках после ключевого слова **`for`** часто называют **“шапкой” цикла**.

В шапке цикла один-единственный раз, в самом начале выполнения оператора `for`, с помощью оператора присваивания переменной-параметру (или переменной-счётчику) необходимо присвоить начальное значение. В качестве переменной-параметра может быть использована любая переменная.

Затем вычисляется условие - операция сравнения или логическое выражение, которое вычисляется каждый раз перед выполнением тела цикла. Операторы тела цикла могут ни разу не выполниться, если условие с самого начала ложно (равно false).

Значение переменной-параметра изменяется каждый раз после очередного прохода цикла, то есть уже после того, как выполнятся операторы тела цикла.

Предупреждение! При неправильном задании условия цикл может повторяться бесконечно, как, например, здесь: `for(x=1;x>0;x++)`.

Чтобы усвоить определение оператора for необходимо выполнить следующий пример.

Листинг 3.6.5.

```
<html>
```

```
<head><title>Пример 3.6.5</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
</head>
<body>
<h2>Пример 3.6.5. Оператор <b>for</b> - оператор цикла с переменной-
параметром</h2>
<script type="text/javascript">
var x = 0;
for (var i = 0; i < 4; i=i+1)
  {
    x=x+1;
    alert(x);
  }
alert ('переменная-параметр после выхода из цикла = ' + i)
</script>
</body>
</html>
```

В листинге 3.6.5 «шапка» цикла - `for(var i=0;i<4;i=i+1)`. Значение переменной **`i`** будет увеличиваться от нуля с шагом, равным единице, до тех пор, пока оно не станет равным 4. Каждый раз после проверки условия, пока это условие истинно, выполняются операторы тела цикла: `{x=x+1;alert(x);}`. Как только условие перестаёт выполняться, осуществляется переход к операторам, стоящим после оператора цикла. В примере 3.6.5 после цикла стоит оператор, выводящий на экран значение переменной цикла, которое сделало ложным значение условия и привело к завершению оператор цикла.

Оператор `for` удобен при работе с массивами. Вспомним пример листинга 3.4.1 из раздела 3.4 и сравним его со следующим примером из листинга 3.6.6.

Листинг 3.6.6.

```
<html>
<head><title>Пример 3.6.6. Цикл for и одномерный массив</title>
<META http-equiv=Content-Type content="text/html; charset=UTF-8" />
</head>
<body>
<h2>Пример 3.6.6. Использование цикла for для работы с одномерным
массивом</h2>
<script>
//
// создаём массив, содержащий дни недели
//
var days_of_week_russ = new Array(7);
days_of_week_russ[0]="Понедельник";
days_of_week_russ[1]="Вторник";
days_of_week_russ[2]="Среда";
days_of_week_russ[3]="Четверг";
days_of_week_russ[4]="Пятница";
days_of_week_russ[5]="Суббота";
days_of_week_russ[6]="Воскресенье";
//
// выводим значения элементов массива
```

```
//
for (var i=0; i<days_of_week_russ.length; i++) {
    document.write("<p>");
    document.write(days_of_week_russ[i]);
    document.write("</p>");
}
</script>
</body>
</html>
```

В примере 3.6.6 массив создаётся по-прежнему «вручную». А вывод массива осуществляется с использованием цикла `for`. Нетрудно представить себе насколько сокращается трудоёмкость программирования по сравнению с программированием «вручную» при выводе значений массива, имеющего большую длину.

Ещё более эффективно использование цикла for для работы с многомерными массивами. Пример работы с двумерным массивом из листинга 3.4.2 раздела 3.4 можно переписать следующим образом:

Листинг 3.6.7.

```
<html><head>
<title>Пример 3.6.7. Цикл for и двумерные массивы.</title>
<META http-equiv=Content-Type content="text/html; charset=UTF-8" />
<link rel="stylesheet" type="text/css" href="Petrov.css" />
<style type="text/css">
table {border: 1px solid black; border-collapse:collapse}
td {border: 1px solid gray;}
</style
</head>
<body>
<h2>Пример 3.6.7. Использование оператора цикла for<br/>
для работы с двумерными массивами.</h2>
<script type="text/javascript">
var a=-50.00, b=50.00;
var n=3, m=2;
var i,j;
// создаём пустой двумерный массив размером n*m
var arr_2 = new Array(n);
for (var i=0; i<n; i++)
    arr_2[i]=new Array(m);
// заполняем массив случайными числами
for (i=0; i<n; i++)
    for (j=0; j<m; j++)
        arr_2[i][j]=a+(b-a)*Math.random();
// выводим массив в виде таблицы
// вначале выводим "шапку" таблицы с номерами столбцов
document.write("<table>");
document.write("<tr><td></td>");
for (j=0; j<m; j++) {
    document.write("<td>Столбец"+j+"</td>");
}
document.write("</tr>");
// затем - номера строк и строку со значениями массива
```

```
for (i=0; i<n; i++){
    document.write("<tr><td>Строка"+i+"</td>");
    for (j=0; j<m; j++)
        document.write("<td>"+arr_2[i][j].toFixed(2)+"</td>");
    document.write("</tr>");
}
document.write("</table>");
</script></body></html>
```

В комментариях листинга 3.6.7 даны пояснения. Дополним комментарии разъяснениями относительно так называемых вложенных циклов. Первый такой цикл используется для заполнения массива случайными числами:

```
// заполняем массив случайными числами
for (i=0; i<n; i++)
    for (j=0; j<m; j++)
        arr_2[i][j]=a+(b-a)*Math.random();
```

В этом фрагменте в тело внешнего оператора цикла с «шапкой» `for (i=0; i<n; i++)` вложен один оператор, который также является оператором цикла: `for(j=0;j<m; j++) arr_2[i][j]=a+(b-a)*Math.random();` .

Возможно вы забыли, что означают операторы `i++` и `j++`, используемые в шапке цикла. Это операторы инкремента. Эти операторы аналогичны операторам `i=i+1` и `j=j+1.`

Оператор `for … in` – последний, рассматриваемый нами оператор цикла. Синтаксис оператора:

`for (переменная in имя_массива) {группа_операторов}`

Группа операторов, входящая в тело цикла, будет выполнена столько раз, сколько элементов имеет массив, указанный, после ключевого слова **`in`**. При этом переменной, указанной перед `in`, будет последовательно присваиваться значение, содержащее индекс текущего элемента массива. Пример 3.6.6 с использованием оператора for...in будет выглядеть следующим образом:

Листинг 3.6.8.

```
<html>
<head><title>Пример 3.6.8. Цикл for...in и одномерный массив</title>
<META http-equiv=Content-Type content="text/html; charset=UTF-8" />
</head>
<body>
<h2>Пример 3.6.8. Использование цикла for...in для работы с одномерным
массивом</h2>
<script type="text/javascript">
//
// создаём массив, содержащий дни недели
//
var days_of_week_russ = new Array(7);
days_of_week_russ[0]="Понедельник";
days_of_week_russ[1]="Вторник";
days_of_week_russ[2]="Среда";
```

```
days_of_week_russ[3]="Четверг";
days_of_week_russ[4]="Пятница";
days_of_week_russ[5]="Суббота";
days_of_week_russ[6]="Воскресенье";
//
// выводим значения элементов массива
//
for (i in days_of_week_russ) {
    document.write("<p>");
    document.write(days_of_week_russ[i]);
    document.write("</p>");
}
</script>
</body>
</html>
```

Более эффективно использование оператора `for...in` для массивов, имеющих неупорядоченные или текстовые индексы (вспомним: массивы, имеющие текстовые индексы называют **ассоциативными массивами**).

Следующий пример иллюстрирует создание документа, содержащего расписание занятий по дисциплине. В качестве индексов используются наименования дней недели, а значениями являются строки, содержащие расписание. Заполняется расписание по-прежнему «вручную», а вывод автоматизирован с использованием оператора `for...in`.

Листинг 3.6.9.

```
<html><head>
<title>
Пример 3.6.9. Цикл for...in и одномерный массив с текстовыми индексами
</title>
</head>
<body>
<h2>Пример 3.6.9. Использование цикла for...in для работы<br/>
с одномерным массивом, имеющим текстовые индексы</h2>
<h3>Расписание занятий по дисциплине "ВТ и программирование"</h3>
<script type="text/javascript">
//
// создаём массив, содержащим расписание занятий по дням недели
//
var days_of_week = new Array(7);
days_of_week["Понедельник"]="1-я и 2-я пары. Лекция.";
days_of_week["Вторник"]="3-я и 4-я пары. Лабораторные работы";
days_of_week["Среда"]="6-я и 7-я пары. Самостоятельная работа";
days_of_week["Четверг"]="Нет занятий по \"ВТ и программирование\"";
days_of_week["Пятница"]="6-я пара. Консультации.";
days_of_week["Суббота"]="Свободный график. Самостоятельная работа";
days_of_week["Воскресенье"]="Выходной";
//
// выводим индексы и значения элементов массива
//
for (i in days_of_week) {
    document.write("<p>");
    document.write(i+" : "+days_of_week[i]);
```

```
    document.write("</p>");
}
</script></body></html>
```

Благодаря строке for (i in days_of_week) переменная i последовательно принимает значения, соответствующие индексам элементов массива и используется в теле цикла для вывода этого индекса и значения элемента массива, соответствующего этому индексу.

3.6.4. Метки и операторы: `continue`, `break`, `switch`

Метки обеспечивают возможность идентификации операторов для последующей ссылки на них из операторов break и continue по имени метки. Правила определения имён для меток аналогичны правилам именования переменных и констант. Синтаксис определения метки имеет следующий вид:

имя_метки : оператор

Оператор **continue** используется в сочетании с операторами цикла. Этот оператор позволяет прервать исполнение операторов тела цикла и перейти к выполнению следующей итерации этого оператора цикла, пропустив все следующие за ним операторы тела цикла.

В следующем примере на экран выводятся только нечетные числа:

Листинг 3.6.10.

```
<html><head>
<title>Пример 3.6.10. Оператор continue</title>
</head>
<body>
<h2>Пример 3.6.10. Использование оператора continue в цикле while</h2>
<h3>Вывод на экран нечётных чисел</h3>
<script type="text/javascript">
var x = 0;
while (x < 10)
  {
    x++;
    if (x % 2 == 0)
      {
        continue;
      }
    alert(x);
  }
</script></body></html>
```

В теле цикла `while` находятся три оператора. Вторым оператором является оператор `if`, в условии которого операция сравнения `(x % 2 == 0)` выдаёт `true`, если x чётно. В этом случае выполняется оператор `continue` и оператор `alert(x)` не выполняется. Если операция сравнения выдаёт `false`, то оператор `continue` не выполняется, `alert(x)` срабатывает и выдаёт на экран нечётное значение перменной `x`.

Оператор continue может использоваться и в сочетании с меткой. В этом случае нарушается предусмотренный по умолчанию порядок выполнения цикла, поскольку переход осуществляется к помеченному оператору. Пример использования метки показан в листинге 3.6.11.

Листинг 3.6.11.

```
<html><head>
<title>Пример 3.6.11. Оператор continue и метка в цикле for</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
</head>
<body>
<h2>Пример 3.6.11. Использование оператора continue и метки в цикле for</h2>
<h3>Вывод на экран значений главной диагонали двумерной матрицы<br/>
случайных чисел a[i,j]=i*j*0.1*Math.random()</h3>
<script type="text/javascript">
var n=m=6;
// создаём массив-матрицу случайных чисел размером n*m
var a=new Array(n);
for (var i=0; i<n; i++) {
    a[i]=new Array(m)
    for (var j=0; j<m; j++)
     a[i][j]=i*j+0.1*Math.random();
}
// печатаем значения главной диагонали матрицы
m1: for (var i=0; i<n; i++)
      for (var j=0; j<m; j++)
          if (i==j) {
      document.write("<p>"+a[i][j].toFixed(3)+"</p>");
      continue m1
          }
          else
      continue;
</script></body></html>
```

Из примера можно понять, что, если во вложенном цикле после continue указывается метка, идентифицирующая внешний цикл, то итерации вложенного цикла прекращаются, когда выполняется переход на новую итерацию помеченного внешнего цикла.

Оператор break (прервать) также, как и continue используется в сочетании с операторами организации цикла. Однако, если оператор continue приводит к переходу к следующему циклу, то на операторе break циклы вообще прекращаются. Оператор break используется также с оператором switch, который рассматривается далее. В следующем примере оператор break, также как и оператор continue применяется в сочетании с оператором if.

Листинг 3.6.12

```
<html><head>
<title>Пример 3.6.12. Оператор break в цикле while</title>
</head>
<body>
<h2>Пример 3.6.12. Использование оператора break в цикле while</h2>
<h3>Вывод на экран последовательности целых чисел<br/>
```

```
от нуля до заданного числа</h3>
<script type="text/javascript">
var x = 0, br;
br=parseInt(prompt("Введите число для прерывания цикла от 1 до 40", ""));
document.write("<p>");
while (x < 40)
  {
  if (x > br)
    {
      break;
    }
  document.write(x+"  ");
  x++;
  }
document.write("</p>");
</script></body></html>
```

В этом примере вводится число, при котором следует прервать выполнение оператора `while`. Цикл оператора `while` будет выполняться до тех пор, пока значение `x` не станет больше значения переменной `br`, и тогда цикл прервется. Например, если вы введете в окне запроса значение 4, на печать будут выведены окна предупредительных сообщений со значениями 0 1 2 3 4.

Оператор `break` можно использовать совместно с меткой.

Оператор `switch` (перейти к…) позволяет выполнить один или несколько операторов, если значение указанного выражения совпадает с меткой. Синтаксис оператора имеет следующий вид:

```
switch (выражение)
{
case метка1;
группа операторов
break;
case метка2;
группа операторов
break;
. . .
default:
группа операторов
break;
}
```

В условии оператора `switch` может стоять имя переменной или выражение, возвращающее значение любого типа.

Тело оператора `switch` заключается в операторные скобки и состоит из повторяющихся ключевых слов `case`, после которых стоят метки. При срабатывании оператора `switch` проверяется совпадение метки и значения, полученного при вычислении выражения. При совпадении выполняется группа операторов, стоящая после метки.

Каждая группа операторов должна заканчиваться оператором `break`, в противном случае будут выполняться все последующие за ней операторы.

Если совпадения не обнаружены, выполняется группа операторов, стоящая после метки **`default`**.

В листинге 3.6.13 приведён пример, в котором в окно запрос/ответ вводится целое число, а в программе с помощью оператора `switch` определяется совпадение введённого числа с меткой, которой помечен соответствующий оператор `alert()`, выводящий введённое число в текстовом виде.

Листинг 3.6.13.

```
<html><head>
<title>Пример 3.6.13. Оператор switch</title>
</head>
<body>
<h2>Пример 3.6.13. Использование оператора switch</h2>
<h3>Вывод на экран вводимого целого числа в текстовом виде</h3>
<script type="text/javascript">
var yourchoice;
yourchoice = parseInt(prompt("Загадайте целое число  от 1 до 4. Можно и
другие, но ...", "1, 2, 3 или 4"))
switch (yourchoice)
{
case 1:
  alert("Вы ввели число: один");
  break;
case 2:
  alert("Вы ввели число: два");
  break;
case 3:
  alert("Вы ввели число: три");
  break;
case 4:
  alert("Вы ввели число: четыре");
  break;
default: //эта метка предусмотрена для перехода по умолчанию
  alert("Вы не ввели число, попадающее в промежуток от 1 до 4");
  break;
}
</script></body></html>
```

3.6.5. Примеры типовых алгоритмов

Во всех примерах предварительно создаются массивы случайных чисел и по таким образом полученным данным демонстрируются некоторые элементарные алгоритмы вычисления числовых характеристик (статистических характеристик), часто используемых на практике. При обновлении документа массивы заполняются новыми значениями и, соответственно, меняются вычисляемые значения. Точные теоретические величины значений, получаемых по массиву случайных чисел,

известны из теории вероятностей (вероятностные характеристики). Вероятностные характеристики можно использовать для проверки правильности расчёта статистических характеристик при отладке и опробовании программ.

Необходимо обратить внимание, что во всех приведённых ниже программах используются так называемые **динамические массивы**, которые в отличие от **статических массивов** создаются и увеличиваются в размерах по мере необходимости.

Пример **вычисления суммы и среднего арифметического** по значениям одномерного массива с использованием оператора цикла с постусловием (используется цикл `do...while`) приведён в листинге 3.6.14.

Можно выделить пять основных блоков программы (или этапов выполнения программы). Собственно вычисление суммы и среднего выполняется в четвёртом и пятом блоках, а первые блоки являются вспомогательными. Опишем содержание каждого блока;

1) описываются необходимые переменные;
2) присваиваются необходимые значения некоторым переменным (исходные данные);
3) создаётся (генерируется) и выводится в документ массив случайных чисел. Значения массива при выводе округляются до трёх знаков после десятичной точки. Следует понимать, что в памяти компьютера эти числа не округлены.
4) созданный выше массив используется для накопления и вывода суммы;
5) полученная сумма используется для расчёта и вывода среднего арифметического.

Листинг 3.6.14.

```
<html><head>
<title>Пример 3.6.14. Сумма и среднее с оператором do...while</title>
</head>
<body>
<h2>Пример 3.6.14. Вычисление суммы и среднего арифметического</h2>
<h3>Используется оператор цикла с постусловием (do...while)</h3>
<script type="text/javascript">
// 1) описываем необходимые переменные
var n, i=sum=0, sr, a=new Array();
// 2) вводим исходные данные
n=20;
// 3)создаём и выводим  массив случайных чисел;
do {
    i=i+1;
    a[i]=Math.random()
}
while (i<n);
i=0;
document.write("<p>Массив случайных чисел:</p><p>")
do {
    i=i+1;
    document.write(a[i].toFixed(3)+"  ");
}
```

```
while (i<n); document.write("</p>");
// 4) накапливаем и выводим сумму
i=0;
do {
    i=i+1;
    sum=sum+a[i];
}
while (i<n);
document.write("<p>Сумма = "+sum.toFixed(3)+"</p>");
// 5) вычисляем и выводим среднее арифметическое
sr=sum/a.length;
document.write("<p>Среднее арифметическое = "+sr.toFixed(3)+"</p>");
</script></body></html>
```

В листинге 3.6.15 приводится пример определения экстремумов (минимального и максимального значений) по значениям одномерного массива. В примере используется оператор цикла с предъусловием (цикл `while`). Этапы выполнения программы поясняются в комментариях.

Листинг 3.6.15.

```
<html><head>
<title>Пример 3.6.15. Экстремумы с оператором while</title>
</head> <body>
<h2>Пример 3.6.15. Вычисление экстремумов (минимального и максимального
значений</h2>
<h3>Используется оператор цикла с предусловием (while)</h3>
<script type="text/javascript">
// 1) описываем переменные
var n, i, min, max, a=new Array();
// 2) вводим исходные данные
n=20;
// 2)генерируем и выводим массив случайных чисел
i=0
var a=new Array();
while (i<n) {
    i=i+1;
    a[i]=Math.random()
}
//
i=0;
document.write("<p>Массив случайных чисел:</p><p>")
while (i<n) {
    i=i+1;
    document.write(a[i].toFixed(3)+"  ");
}
document.write("</p>");
// 3)вычисляем и выводим минимум
i=0;
min=a[1];
while (i<n) {
    i=i+1;
    if (a[i]<min) min=a[i];
}
document.write("<p>Минимум = "+min.toFixed(3)+"</p>");
// 4)вычисляем и выводим максимум
```

```
i=0;
max=a[1];
while (i<n) {
    i=i+1;
    if (a[i]>max) max=a[i];
}
document.write("<p>Максимум = "+max.toFixed(3)+"</p>");
</script></body></html>
```

Вычисление суммы и среднего арифметического по значениям двумерного массива показано в листинге 3.6.16. В примере используется цикл с переменной-параметром (цикл `for`).

Листинг 3.6.16.

```
<html><head>
<title>Пример 3.6.16. Сумма и среднее по значениям двумерного массива</title>
</head>
<body>
<h2>Пример 3.6.16. Вычисление суммы и среднего арифметического по значениям
двумерного массива.</h2>
<h3>Используется оператор цикла с переменной параметром (for)</h3>
<script type="text/javascript">
// 1) описываем переменные
var n, i, j, sum, sr, a=new Array();
// 2) вводим исходные данные
n=6;
// 3) создаём пустой двумерный массив
a=new Array(n);
    for(j=1; j<=n; j++)
       a[j]=new Array(n)
// 4) заполняем массив случайными числами:
for (i=1; i<=n; i++){
    for(j=1; j<=n; j++)
       a[i][j]=Math.random();
}
// 5) выводим массив в виде таблицы
document.write("<p>Матрица случайных чисел:</p>");
document.write("<table>");
for (i=1; i<=n; i++) {
    document.write("<tr>");
    for(j=1; j<=n; j++) {
       document.write("<td>"+a[i][j].toFixed(3)+" </td>");
    }
    document.write("</tr>");
}
document.write("<table>");
// 6) рассчитываем и выводим сумму значений массива
sum=0;
for (i=1; i<=n; i++)
    for(j=1; j<=n; j++)
       sum=sum+a[i][j];
document.write("<p>Сумма = "+sum.toFixed(3)+"</p>");
// 7) рассчитываем и выводим среднее арифметическое
sr=sum/(n*n);
document.write("<p>Среднее арифметическое = "+sr.toFixed(3)+"</p>");
</script></body></html>
```

Вычисление сумм и средних арифметических по значениям каждого столбца двумерного массива показано в листинге 3.6.17. В примере используется цикл `for`.

Листинг 3.6.17.

```
<html><head>
<title>Пример 3.6.17. Суммы и средние по столбцам двумерного массива</title>
</head>
<body>
<h2>Пример 3.6.17. Вычисление сумм и средних арифметических по столбцам
двумерного массива.</h2>
<h3>Используется оператор цикла с переменной-параметром (for)</h3>
<script type="text/javascript">
// 1) описываем переменные
var n, i, j, sr, a=new Array(), sum=new Array(), sr=new Array();
// 2) вводим исходные данные
n=6;
// 3) создаём пустой двумерный массив
a=new Array(n);
    for(j=1; j<=n; j++)
       a[j]=new Array(n)
// 4) заполняем двумерный массив случайными числами,
// причём числа в столбцах вычисляются по формуле:
// Math.random+(j-1), j=1,2,3; j-индекс столбца
for (i=1; i<=n; i++){
    for(j=1; j<=n; j++)
       a[i][j]=Math.random()+(j-1);
}
// 5) выводим двумерный массив в виде таблицы
document.write("<p>Матрица случайных чисел:</p>");
document.write("<table>");
for (i=1; i<=n; i++) {
    document.write("<tr>");
    for(j=1; j<=n; j++) {
       document.write("<td>"+a[i][j].toFixed(3)+" </td>");
    }
    document.write("</tr>");
}
document.write("<table>");
// 6) рассчитываем суммы по столбцам и запоминаем их
// в одномерном массиве
for (j=1; j<=n; j++) {
    sum[j]=0;
    for(i=1; i<=n; i++) {
       sum[j]=sum[j]+a[i][j];
    }
}
// 7) выводим суммы
document.write("<p>Суммы по столбцам:</p>");
for (j=1; j<=n; j++)
    document.write(sum[j].toFixed(3)+" ");
// 8) рассчитываем, запоминаем и выводим средние арифметические
document.write("<p>Средние арифметические по столбцам:</p>");
for (j=1; j<=n; j++) {
    sr[j]=sum[j]/n;
    document.write(sr[j].toFixed(3)+" ");
```

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

Следующий пример отличается от примера в листинге 3.6.16 тем, для вычисления среднего арифметического не используются значения, выходящие за установленные пределы. Так, например, строятся фильтры, отсеивающие ошибочные значения. В примере используется цикл `for` совместно с операторами `if`, `continue` и меткой.

Листинг 3.6.18.

```
<html><head>
<title>Пример 3.6.18. Среднее по отфильтрованным значениям двумерного
массива</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
</head>
<body>
<h2>Пример 3.6.16. Вычисление суммы и среднего арифметического по значениям
двумерного массива.</h2>
<h3>Используется оператор цикла с переменной параметром (for)</h3>
<script type="text/javascript">
// 1) описываем переменные
var n, i, j, sum, sr, k, min, max, a=new Array();
// 2) вводим исходные данные
n=6; min=-0.4; max=0.4;
// 3) создаём пустой двумерный массив
a=new Array(n);
    for(j=1; j<=n; j++)
       a[j]=new Array(n)
// 4) заполняем массив случайными числами:
for (i=1; i<=n; i++){
    for(j=1; j<=n; j++)
       a[i][j]=Math.random()-0.5;
}
// 5) выводим массив в виде таблицы
document.write("<p>Матрица случайных чисел:</p>");
document.write("<table>");
for (i=1; i<=n; i++) {
    document.write("<tr>");
    for(j=1; j<=n; j++) {
       document.write("<td>"+a[i][j].toFixed(3)+" </td>");
    }
    document.write("</tr>");
}
document.write("<table>");
// 6) рассчитываем сумму значений массива
sum=0;
for (i=1; i<=n; i++)
m1: for(j=1; j<=n; j++)
       if (a[i][j]>max||a[i][j]<min) {
           document.write("<p>Значение "+a[i][j].toFixed(3)+"(строка "+i+";
столбец "+j+") исключено из расчётов.</p>");
           continue m1
       }
       else sum=sum+a[i][j];
// 7) рассчитываем и выводим среднее арифметическое
sr=sum/(n*n);
```

```
document.write("<p>Среднее арифметическое = "+sr.toFixed(3)+"</p>");
</script></body></html>
```

3.6.6. Упражнения

Веб-страницы с выполненными упражнениями необходимо оформлять на своём персональном сайте так же, как это вы делали в предыдущих упражнениях. Не забывайте присоединять стили, созданные при освоении раздела, посвящённого HTML и CSS, чтобы сохранялось выбранное стилевое оформление всех страниц сайта. Помните также о тэге <meta ...> для указания кодировки UTF-8.

Примеры некоторых упражнений можно посмотреть у студента Петрова на сайте pvn.ho.ua.

Упражнение 3.6.1.

Создать html-документ с программой на языке Javascript, изменив пример из листинга 3.6.1. При этом необходимо учесть следующие условия:

1) повторите пример из листинга 3.6.1, заменив в нём первый полный оператор `if` двумя сокращёнными операторами `if`, а два последних сокращённых оператора `if` одним полным оператором `if`;

2) замените все выводы результатов методом `alert()` на вывод результатов методом `document.write()`;

3) в этом же html-документе поместите ответы на первые три контрольных вопроса из пункта 3.6.7 (с 1-го по 3-й);

Код упражнения необходимо сохранить в файле exercise_3_6_1.html и оформить его отображение на своём сайте так, как это вы делали с предыдущими упражнениями.

Напоминание! Полный оператор `if` содержит ключевые слова `if` и `else`, а сокращённый — только лишь `if` (без `else`).

Упражнение 3.6.2.

Напишите html-документ с программой на языке Javascript, в котором:

1) с помощью метода prompt() вводится целое число;

2) с помощью одного условного оператора делается проверка условия на чётность/нечётность введённого числа и, в зависимости от выполнения условия с помощью метода document.write() выводится введённое число и текстовое сообщение о чётности/нечётности числа;

3) в этом же html-документе поместите ответы на два контрольных вопроса из пункта 3.6.7 (4-й и 5-й вопросы).

Код упражнения необходимо сохранить в файле exercise_3_6_2.html и оформить его отображение на своём сайте так, как это вы делали с предыдущими упражнениями.

Упражнение 3.6.3.

Напишите html-документ с программой на языке Javascript, в котором повторите пример из листинга 3.6.3, изменив его так, чтобы:

1) число циклов n вводилось с помощью метода prompt();
2) вывод переменной цикла осуществлялся с помощью метода document.write();
3) выводились бы только лишь чётные значения переменной цикла;
4) в этом же html-документе поместите ответы на четыре контрольных вопроса из пункта 3.6.7 (с 6-го по 9-й).

Код упражнения необходимо сохранить в файле exercise_3_6_3.html и оформить его отображение на своём сайте так, как это вы делали с предыдущими упражнениями.

Упражнение 3.6.4.

Напишите html-документ с программой на языке Javascript, в котором повторите пример из листинга 3.6.4, изменив его так, чтобы:

1) число циклов n вводилось с помощью метода prompt();
2) вывод переменной цикла осуществлялся с помощью метода document.write();
3) выводились бы только лишь нечётные значения переменной цикла;
4) в этом же html-документе поместите ответы на три контрольных вопроса из пункта 3.6.7 (с 10-го по 12-й).

Код упражнения необходимо сохранить в файле exercise_3_6_4.html и оформить его отображение на своём сайте так, как это вы делали с предыдущими упражнениями.

Упражнение 3.6.5.

Напишите html-документ с программой на языке Javascript, в котором повторите пример из листинга 3.6.5, изменив его так, чтобы:

1) число циклов n вводилось с помощью метода prompt(), при этом: n > 10;
2) вывод переменной цикла осуществлялся с помощью метода document.write();
3) выводились бы значения переменной-параметра цикла только лишь попадающие в интервал [n-2, n-10];
4) в этом же html-документе поместите ответы на два контрольных вопроса из пункта 3.6.7 (13-й и 14-й).

Код упражнения необходимо сохранить в файле exercise_3_6_5.html и оформить его отображение на своём сайте так, как это вы делали с предыдущими упражнениями.

Упражнение 3.6.6.

Напишите html-документ с программой на языке Javascript, в котором повторите пример из листинга 3.6.6, изменив его так, чтобы:

1) в документ выводилось лишь два параграфа, в одном из них выбирался бы день недели «Вторник», а во втором - «Пятница» с добавлением текста « - я должен заниматься программированием»;

2) поиск в массиве дней недели необходимого дня осуществлялся бы с помощью оператора цикла for с вложенным оператором if, а в качестве условия для поиска вторника использовалась бы строка «Вторник», для поиска пятницы — индекс ;

3) в этом же html-документе поместите ответ на два контрольных вопроса из пункта 3.6.7 (15-й и 16-й).

Код упражнения необходимо сохранить в файле exercise_3_6_6.html и оформить его отображение на своём сайте так, как это вы делали с предыдущими упражнениями.

Упражнение 3.6.7.

Напишите html-документ с программой на языке Javascript, в котором повторите пример из листинга 3.6.14, изменив его так, чтобы:

1) число циклов n вводилось с помощью метода prompt();

2) вместо оператора do...while использовался бы цикл for;

3) в этом же html-документе поместите ответы на два контрольных вопроса из пункта 3.6.7 (17-й и 18-й).

Код упражнения необходимо сохранить в файле exercise_3_6_7.html и оформить его отображение на своём сайте так, как это вы делали с предыдущими упражнениями.

Упражнение 3.6.8.

Напишите html-документ с программой на языке Javascript, в котором повторите пример из листинга 3.6.15, изменив его так, чтобы:

1) число циклов n вводилось с помощью метода prompt();

2) вместо оператора while использовался бы цикл for;

3) в этом же html-документе поместите ответы на четыре контрольных вопроса из пункта 3.6.7 (с 19-го по 22-й).

Код упражнения необходимо сохранить в файле exercise_3_6_8.html и оформить его отображение на своём сайте так, как это вы делали с предыдущими упражнениями.

Упражнение 3.6.9.

Напишите html-документ с программой на языке Javascript, в котором повторите пример из листинга 3.6.16, изменив его так, чтобы:

1) число циклов n вводилось с помощью метода prompt();

2) для вычисления суммы и среднего арифметического не использовались бы значения последнего столбца двумерной матрицы;

3) в этом же html-документе поместите ответы на два контрольных вопроса из пункта 3.6.7 (23-й и 24-й).

Код упражнения необходимо сохранить в файле exercise_3_6_9.html и оформить его отображение на своём сайте так, как это вы делали с предыдущими упражнениями.

Упражнение 3.6.10.

Напишите html-документ с программой на языке Javascript, в котором повторите пример из листинга 3.6.17, изменив его так, чтобы:

1) число циклов n вводилось с помощью метода prompt(), причём n>2 и при вводе n <3 выводилось бы сообщение с просьбой повторить ввод;

2) для вычисления сумм и средних арифметических не использовались бы первое и последнее значения в каждом столбце двумерной матрицы;

3) в этом же html-документе поместите ответы на два контрольных вопроса из пункта 3.6.7 (25-й и 26-й).

Код упражнения необходимо сохранить в файле exercise_3_6_10.html и оформить его отображение на своём сайте так, как это вы делали с предыдущими упражнениями.

Упражнение 3.6.11.

Напишите html-документ с программой на языке Javascript, в котором повторите пример из листинга 3.6.18, изменив его так, чтобы:

1) число циклов n вводилось с помощью метода prompt(), причём n>2 и при вводе n <3 выводилось бы сообщение с просьбой повторить ввод;

2) для вычисления суммы и среднего арифметического значений главной диагонали матрицы использовался бы оператор if без использования метки и оператора continue;

3) в этом же html-документе поместите ответы на два контрольных вопроса из пункта 3.6.7 (27-й и 28-й).

Код упражнения необходимо сохранить в файле exercise_3_6_11.html и оформить его отображение на своём сайте так, как это вы делали с предыдущими упражнениями.

Упражнение 3.6.12.

Создать html-документ с программой на языке Javascript для генерации и отображения в документе двумерной квадратной матрицы случайных вещественных чисел. При это необходимо учесть следующие условия:

1) входные данные: n, a, b, l - вводятся с помощью метода prompt();

2) размер матрицы: n*n, где n - целое число, n<10;

2) случайные числа задаются с помощью метода Math.random() на интервале [a,b], где a и b — вещественные числа, a<b;

3) при создании и отображении матрицы необходимо использовать оператор цикла for (оператор цикла с переменной-параметром);

4) вывод матрицы в документ реализовать в виде html-таблицы с использованием метода document.wrtite();

5) значения элементов матрицы должны быть округлены с точностью до l знаков после десятичной точки;

6) упражнение необходимо сохранить в файле exercise_3_6_12.html и оформить его отображение на своём сайте так, как это вы делали с предыдущими упражнениями.

3.6.7. Контрольные вопросы

1. Синтаксис полного оператора `if`?
2. Синтаксис сокращённого оператора `if`?
3. В каком случае можно не использовать операторные скобки в операторе `if`?
4. Синтаксис условного оператора?
5. В каких случаях можно использовать условный оператор, вместо оператора `if`?
6. Операторы цикла ?
7. Ключевые слова оператора цикла с предусловием?
8. Выполнится ли оператор цикла с предусловием, если условие в «шапке» оператора цикла с самого начала ложно?
9. Будет ли ошибкой, если в теле оператора цикла с предусловием не будет ни одного оператора?
10. Ключевые слова оператора цикла с постусловием?
11. Выполнится ли оператор цикла с постусловием, если условие в «шапке» оператора цикла с самого начала ложно?
12. Будет ли ошибкой, если в теле оператора цикла с постусловием не будет ни одного оператора?

13. Чему равна переменная-параметр в цикле for после завершения оператора цикла?

14. Будет ли ошибкой, если в теле оператора цикла for не будет ни одного оператора?

15. Можно ли использовать оператор цикла for для работы с ассоциативным массивом (массивом, индексы которого — неупорядоченная последовательность чисел или текстовых значений)?

16. Пример вложенного цикла for?

17. Пример цикла for...in?

18. Можно ли использовать оператор цикла for...in для работы с обычным массивом (массивом, индексы которого — упорядоченная последовательность целых чисел)?

19. Назначение оператора continue?

20. Назначение оператора break?

21. Может ли использоваться метка совместно с оператором break?

22. Ключевые слова оператора switch?

23. Пример операторов для вычисления суммы по значениям одномерного массива?

24. Пример операторов для поиска минимума среди значений одномерного массива?

25. Пример операторов для поиска максимума среди значений одномерного массива?

26. Пример операторов для вычисления суммы по значениям двумерного массива?

27. Пример операторов для вычисления суммы по значениям главной диагонали двумерной матрицы?

28. Пример операторов для поиска минимума среди значений главной диагонали двумерной матрицы?

29. Пример операторов для вычисления суммы по значениям под главной диагональю двумерной матрицы?

30. Пример операторов для поиска максимума среди значений над главной диагональю двумерной матрицы?

3.6.8. Литература и веб-источники к подразделу

[2] Дронов В.А. Javascript в Web-дизайне. – СПб.: БХВ-Петербург, 2004. - 880 с.

[3] Кингсли-Хью Э., Кингсли-Хью К. JavaScript 1.5: учебный курс. – СПб.: Питер, 2001. – 272 с.

[4] Полупанов В.Н. Программирование на стороне клиента. Конспект лекций. Керчь, КГМТУ, 2006. - 280 с.

Веб-источники

[6] http://pvn.ho.ua/pvn.org.ua/ – лекции на сайте преподавателя.

[8] http://analog.com.ua - HTML - справочник (наиболее простой).

[9] http://css.manual.ru/ - CSS-справочник (наиболее простой).

3.7. События и функции

3.7.1. События и обработчики событий

Каждое действие пользователя, например, щелчок кнопкой мыши или перемещение указателя мыши, формирует некоторый сигнал, сообщающий операционной системе о произошедшем. Операционная система анализирует эти сигналы и посылает браузеру сообщения о том, что была нажата такая-то кнопка мыши или указатель мыши имеет такие-то координаты. Браузер фиксирует эти сообщения как **события** среди перечня возможных событий. Если в момент щелчка или при перемещении указатель находился над каким-то HTML-элементом web-страницы и в коде открывающего тэга этого элемента предусмотрен запуск программы, связанный с этими событиями, то события вызовут выполнение этой программы. Программа (сценарий, скрипт), связанная с событием, называется **обработчиком события**.

В этом подразделе будут использоваться всего три события. Этих событий достаточно для выполнения заданий настоящего подраздела. Другие события будут вводиться по мере необходимости, но основные принципы их использования одинаковы. Обо всех событиях можно узнать в справочниках, например в этом: http://javascript.ru/manual.

Перечислим необходимые нам три события:

1) **Click** - щелчок мышью на элементе web-страницы (поддерживается большинством дескрипторов);

2) **MouseOver** - наведение указателя мыши поверх элемента html-документа (поддерживается большинством дескрипторов);

3) **Load** - web-страница полностью загружена (поддерживается дескриптором <BODY>).

События можно использовать для запуска программ. Для этого событие нужно

«прикрепить» к нужному элементу документа, вставив его, как обычный атрибут в открывающем HTML-тэге, только к наименованию события нужно прибавить приставку **on**. В качестве параметра атрибута в кавычках указываются операторы языка Javascript. Как уже говорилось, эти операторы называют **обработчиком события**. Например, таким образом записанный тэг: `<body onLoad='var name="Петров Иван"; alert(name)'>` приведёт к тому, что после полной загрузки web-страницы создастся переменная **name**, этой переменной присвоится данное-строка **Петров Иван**, а затем эта переменная будет выведена в диалоговом окне с помощью метода alert().

Ниже приведён пример использования трёх перечисленных событий.

Листинг 3.7.1.

```
<html><head><title>Пример 3.7.1. Первое использование событий.</title>
<META http-equiv=Content-Type content="text/html; charset=utf-8">
</head>
<body onLoad='stud="Студент Петров."; alert(stud)'>
<h2>Пример 3.7.1. Первое использование событий</h2>
<h3>Вычисление частного от деления двух чисел, вызов сообщения, многократное
суммирование.</h3>
<script>
var n1 = prompt("Введите первое число");
var n2 = prompt("Введите второе число");
var text="Вас предупреждали, что суммы не будет!";
//---------------------------------------
document.write("<p style='background-color:red' onClick='ch=n1/n2; s=0;
alert(ch)'>");
document.write("Щёлкни на красном, чтобы получить частное исходных чисел.");
document.write("</p>");
//---------------------------------------
document.write("<p style='background-color:yellow' onClick='alert(text)'>");
document.write("На жёлтом щёлкнешь - сумму не получишь. Убедись в этом.");
document.write("</p>");
//---------------------------------------
document.write("<p style='background-color:green' onMouseOver='s=s+ch; alert(s)'>");
document.write("Проведёшь мышкой несколько раз над зелёным и столько же раз ");
document.write("просуммируешь частное, которое получено, щёлкая на красном.");
document.write("</p>");
//---------------------------------------
</script>
</body>
</html>
```

В примере листинга 3.7.1 обработчики событий предусмотрены для четырёх тэгов: для `<body>` и трёх абзацев, закрашенных красным, жёлтым и зелёным цветом. Дадим пояснения лишь для обработчика события **Load**, что в тэге `<body>`, - почему это событие срабатывает не сразу, а лишь после закрытия второго диалогового окна. Дело в том, что диалоговые окна останавливают процесс загрузки страницы. В примере - это друг за другом появляющиеся два диалоговых окна, вызываемые методом `prompt()`, используемые для ввода исходных значений. Загрузка данной страницы завершается лишь после закрытия этих двух диалоговых окон. И лишь после полного завершения загрузки страницы происходит событие **Load**, предусмотренное для тэга `<body>`.

Чтобы закрепить полученные знания рассмотрим ещё один пример, в котором также, как в предыдущем примере, вводится два исходных числа и проводятся типовые вычисления с использованием арифметических операций.

Листинг 3.7.2.

```
<html><head><title>Пример 3.7.2. Второе использование событий.</title>
<META http-equiv=Content-Type content="text/html; charset=utf-8">
</head><body>
<h2>Пример 3.7.2. Студент Петров. Второе использование событий</h2>
<h3>Вычисление частных от деления и сумм и произведений, накапливающихся при щелчках
мышкой.</h3>
<script>
var text="На белом кликать бесполезно!";
var n1 = prompt("Введите первое число");
var n2 = prompt("Введите второе число");
//---------------------------------------
document.write("<p style='background-color:red' onClick='ch=n1/n2;alert(ch)'>");
document.write("Щёлкни на красном и получишь частное от деления исходных
чисел.</p>");
//---------------------------------------
document.write("<p style='background-color:green; color=yellow'
onClick='s=Number(n1)+Number(n2); alert(s)'>");
document.write("Щёлкни на зелёном и получишь сумму исходных чисел.</p>");
//---------------------------------------
document.write("<p style='background-color:blue; color=gold' onClick='p=n1*n2;
alert(p)'>");
document.write("Щёлкни на синем и получишь произведение исходных чисел.</p>");
//---------------------------------------
document.write("<p style='background-color:white' onClick='alert(text)'>");
document.write("-------------------------------------------------------------</p>");
//---------------------------------------
document.write("<p style='background-color: salmon' onClick='ch=ch/(n1+n2);
alert(ch)'>");
document.write("Пощёлкай на светло-красном, чтобы получать ");
document.write("частное от многократного деления получаемого частного на сумму
исходных чисел.</p>");
//---------------------------------------
document.write("<p style='background-color:lightgreen' onClick='s=s+s; alert(s)'>");
document.write("Пощёлкай на светло-зелёном, чтобы получать ");
document.write("накопительную сумму от суммы исходных чисел.</p>");
//---------------------------------------
document.write("<p style='background-color:lightblue' onClick='p=p*p; alert(p)'>");
document.write("Пощёлкай на светло-синем и получишь накопительное ");
document.write("произведение от произведения исходных чисел.</p>");
</script>
</body>
</html>
```

3.7.2. Создание и вызов функции

До сих пор в качестве обработчика событий использовались фрагменты программ в виде нескольких операторов языка Javascript. Но обработка может быть очень сложной, состоящей из сотен операторов, и тогда в коде страницы будет трудно разобраться. Кроме того, одинаковые фрагменты могут понадобиться в качестве обработчиков многих событий и тогда придётся много раз дублировать одинаковые фрагменты программ. Для того, чтобы избежать этих недостатков,

придумали конструкции - **подпрограммы**.

Подпрограммы подразделяются на подпрограммы-процедуры и подпрограммы-функции или просто — на **процедуры и функции**. Мы далее будем использовать лишь функции. Вы уже использовали встроенные функции `alert()`, `Math.random()` и другие. В этом подразделе мы научимся создавать и использовать собственные функции.

Функция – это группа операторов, предназначенных для определенной цели и объединенных под общим именем. Функцию можно вызвать для выполнения, обратившись к ней по имени. Взаимодействие функции с внешней программой, из которой она была вызвана, происходит путём передачи функции параметров и приёма от неё результатов вычислений.

Функцию вначале необходимо создать (говорят - описать). Описание функции имеет следующий вид:

```
Function имя_функции (формальный_параметр1, формальный_параметр2, ...)
{
операторы;
}
```

В качестве **формальных параметров** используются переменные и выражения, но можно использовать и непосредственно данные.

Описание функции (или просто — функцию) можно размещать в любом месте документа, но так, чтобы к моменту вызова функции, часть документа с описанием функции уже была загружена. Мы чаще всего будем размещать описания функций в блоке `<head>...</head>` перед закрывающим тэгом `</head>`.

После описания функцию можно использовать (говорят - вызывать на исполнение или просто вызывать). Вызов функции имеет вид:

имя_функции(фактический_параметр1, фактический_параметр2, …)

Фактические параметры - это переменные, выражения или непосредственно данные, при вызове функции присваиваемые (говорят - передаваемые) формальным параметрам в описании функции.

Ниже приведён листинг 3.7.3, в котором показан пример, похожий на пример 3.7.2, но с использованием функций в качестве обработчиков событий.

Листинг 3.7.3.

```
<html><head><title>Пример 3.7.3. Первое использование функций.</title>
<meta http-equiv=Content-Type content="text/html; charset=utf-8">
<script>
function sum_ab(a, b){var a, b, sum; sum=a+b; return sum}
function alert_s(a){var a; alert(a)}
function alert_text(){alert("На белом кликать бесполезно!")}
function sum_sum(a, b)
{
        var a,b;
        ss=sum_ab(a,b)
        return ss
}
</script></head>
<body>
<h2>Пример 3.7.3. Первое использование функций.</h2>
<h3>Использование функций в качестве обработчика событий:<br>
Вычисление частных от деления и сумм и произведений, накапливающихся при щелчках
мышкой.</h3>
<script>
var n1 = prompt("Введите первое число");
var n2 = prompt("Введите второе число");
var ss=0;
//----------------------------------------
document.write("<p style='background-color:green; color=yellow' ");
document.write("onClick='s=sum_ab(Number(n1),Number(n2)); alert_s(s)'>");
document.write("Щёлкни на зелёном и получишь сумму исходных чисел.");
document.write("</p>");
//----------------------------------------
document.write("<p style='background-color:white' onClick='alert_text()'>");
document.write("------------------------------------------------------------</p>");
//----------------------------------------
document.write("<p style='background-color:lightgreen' onClick='sum_sum(ss,s);
alert_s(ss)'>");
document.write("Пощёлкай на светлозелёном, чтобы получать ");
document.write("накопительную сумму от суммы первого и второго чисел.");
document.write("</p>");
//----------------------------------------
document.write("<p style='background-color:white' onClick='alert_text()'>");
document.write("------------------------------------------------------------</p>");
//----------------------------------------
document.write("<p style='background-color:silver'>");
document.write("Функции можно использовать не только в качестве обработчиков
событий:<br>");
s=sum_ab(Number(n1),Number(n2)); // вычисляем сумму исходных чисел
for(var i=1;i<=3;i++) {ss=sum_sum(ss,s)}; // три раза считаем сумму суммы исх.чисел
document.write("сумма исходных чисел = "+s+"; накопленная сумма после трёх итераций =
"+ss );
document.write("</p>");
</script>
</body>
</html>
```

Последний блок вычисления накопительной суммы с помощью цикла for демонстрирует, что функцию можно использовать не только в качестве обработчика событий.

3.7.3. Оператор return

Приведем пример функции для вычисления факториала, известного вам из школьного курса

математики (факториал целого положительного числа равен n!=1*2*3*…*n, если n>=1, и n!=1, если n=0 и n=1):

Листинг 3.7.4.

```
function factorial_iter(n){
  if (n<=1) return 1;
  var f = 2;
  for (var i = 3; i <= n; i++){
    f = f * i;
  }
  return f;
}
```

В примере использован еще неизвестный вам оператор **return**, возвращающий результат в программу, из которой вызывается наша функция. Описав функцию factorial_iter() теперь мы можем вызывать её на выполнение различными способами. Кроме рассмотренных в предыдущем пункте способов вызова функций (каких ?) – функции можно вызывать на исполнение в выражениях (или проще — использовать в выражениях). Ранее мы освоили выражения с операндами и операциями (арифметическими и др.). Теперь, зная про функции, мы можем использовать их в качестве операндов в правой части оператора присваивания для вычисления конкретных чисел, например так:

```
fctrl_10 = factorial_iter(10);   // 10! = 3628800, можете проверить
```

или совместно с арифметическим операторами деления и умножения, например, так:

```
fctrl_12 = fctrl_10 / factorial_iter(12) * 4;
```

В последнем примере предполагается, что в переменную fctrl_10 уже занесено значение факториала, вычисленное с помощью предыдущего вызова функции с параметром, равным 10-ти. Тогда вначале будет вычислено значение factorial_iter(12) и лишь потом продолжатся операции в порядке их приоритета, то есть – **операции вычисления значений функции обладают наиболее высоким приоритетом** перед всеми ранее рассмотренными операциями (конечно, этот приоритет можно нарушать с помощью круглых скобок).

Определение функции может содержать вызов этой же функции, то есть в языке JavaScript допускается так называемая **рекурсия** или **рекурсивный вызов функции**.

Выражение для факториала можно записать в другой (рекуррентной) форме, когда каждое последующее значение сомножителя выражается через предыдущее: n!=n*(n-1)!, если n>=1, и n!=1, если n=0 и n=1.

Приведем пример вычисления факториала теперь уже с использованием рекурсии. Для этого приведенный выше пример перепишем так:

Листинг 3.7.5.

```
function factorial_rec(n){
  if (n<=1) return 1;
  return n*factorial_rec(n-1);
}
fctrl_10 = factorial_rec(10).
```

Теперь, когда у нас имеется две функции вычисления факториала, мы можем использовать эти функции, например, так (заодно можно проверить правильность описания функций):

```
delta = factorial_iter(10) - factorial_rec(10) // равно нулю.
```

Рекомендуется повторить эти примеры и убедиться в правильности написания функций.

Чтобы понять, как работает механизм рекурсивного вызова функций, нужно вспомнить, что вызывающая программа приостанавливается до тех пор, пока не завершится выполнение вызываемой программы. При каждом такой остановке в памяти сохраняются все локальные переменные, принадлежащие остановленной программе (имя функции тоже является переменной). Как только завершится самый последний вызов программы (в случае функции из листинга 3.7.5, при `n`=1), то срабатывает предыдущий вызов (при `n`=2), и так далее, пока не дойдёт очередь до самого первого вызова функции (когда входным параметром является величина `n`).

Использование рекурсии позволяет создавать более изящные и короткие алгоритмы по сравнению с использованием итераций. Но при реализации на ЭВМ могут возникнуть проблемы с нехваткой памяти для хранения локальных переменных. Хотя в последнее время в современных интерпретаторах и компиляторах эта проблема успешно решается за счёт специальных оптимизационных процедур по предварительному переводу рекурсивных процедур в итерационные.

3.7.4. Использование таймера

Кроме рассмотренных методов, функции можно также вызывать на выполнение с помощью так называемых **таймеров** - функций, встроенных в объект `window`. Как уже не раз упоминалось ранее, функции, встроенные в объект, называются методами. Таймеры запускают события по истечении определенного интервала времени без участия пользователя. Объекту `window` принадлежит два таймера:

- setTimeout(), выполняющий программу один раз по истечении установленного промежутка времени;
- setInterval(), выполняющий программу циклически через установленный промежуток времени.

Оба метода используют одинаковый набор параметров и имеют следующий синтаксис:

```
var timerRef=window.setTimeout(выражение, период);
var timerRef=window.setInterval(выражение, период);
```

где первый параметр представляет собой строку, содержащую выражение-обработчик (в том числе, вызов функции), а второй параметр – целое число, указывающее временную задержку в миллисекундах перед очередным выполнением выражения, указанного в качестве первого параметра.

Имя объекта `window` можно не указывать:

```
var timerRef=setTimeout(выражение, период);
var timerRef=setInterval(выражение, период).
```

Переменной `timerRef` присваивается идентификатор целого типа, но эту переменную можно не использовать и вызывать таймеры следующим способом:

```
var timerRef=setTimeout(выражение, период);
var timerRef=setInterval(выражение, период).
```

Таймеры могут быть удалены в любой момент посредством методов удаления **`ClearTimeout()`** или **`ClearInterval()`**. Оба метода удаления принимают в качестве параметра значение, присваиваемое переменной `timerRef.`

Рассмотрим пример использования функции, запускаемой с помощью таймера `setInterval()`.

Листинг 3.7.6.

```
<html><head>
<title>Пример 3.7.6: таймер setInterval() и функции</title>
<meta content="text/html; charset=utf-8" http-equiv="Content-Type">
<script>
//  начальные установки
var d = document, w = window, circle = false,   timerCircle;
var radius = 50, angle = 0;
////////////////////////////////////////////
function initCircle(){
  //   движение изображения-спутника по окружности
  circle=false;
  w.clearInterval(timerCircle);
  timerCircle=w.setInterval("moveImage()", 10);
}
////////////////////////////////////////////
////////////////////////////////////////////////////////
function moveImage(){
  //  вычисление координат следующего положения изображения
  rad = angle * Math.PI/180;
  var xbegin = 70, ybegin = 150;
  d.myimage.style.pixelLeft = xbegin + radius * Math.sin(rad);
  d.myimage.style.pixelTop = ybegin + radius * Math.cos(rad);
  angle=angle+1;
  if (circle) w.clearInterval(timerCircle);
}
```

```
//////////////////////////////////////////////////////
</script>
<style>
img#mycenter {
position:absolute; left:70; top: 150; width:40; height:16
}
img#myimage {
position:absolute; left:24; top:150; width:42; height:23
}
</style></head>
<body>
<h2>Пример 3.7.6. Таймер и функции</h2>
<h3>Движение изображения по окружности</h3>
<button type="button" title="Запустить" onClick="initCircle()">Старт</button>
<button type="button" title="Остановить" onClick="circle=true">Стоп</button>
<img id="mycenter" name="mycenter" src="./3_7_img/img5.GIF">
<img id="myimage" name="myimage" src="./3_7_img/ANGEL.GIF" >
</body></html>
```

В примере 3.7.6 эффект вращения спутника создаётся с помощью оператора `timerCircle=setInterval('moveImage()', 10)`, где первый параметр является именем запускаемой функции `moveImage()`, а второй параметр равен 10-ти миллисекундам. В результате каждые 10 миллисекунд выполняется вызов функции `moveImage()`, в ней происходит перевычисление координат спутника и присвоение этих координат переменным — свойствам стилей изображения: `d.myimage.style.pixelLeft = xbegin + radius * Math.sin(rad); d.myimage.style.pixelTop = ybegin + radius * Math.cos(rad)`. Эти свойства являются координатами левого верхнего угла изображения. Обновление свойств каждые 10 миллисекунд вызывает перемещение спутника. Таким образом создается эффект вращения изображения.

Прекращение перемещения изображения выполняется с помощью оператора `clearInterval(timerCircle)`.

3.7.5. Библиотеки функций

Программы (сценарии) JavaScript, могут размещаться не только непосредственно в html-документе между тэгами `<script>` и `</script>`, но и **присоединяться к web-странице из отдельного файла с помощью атрибута SRC тэга <SCRIPT> языка HTML**. Такие программы называют присоединяемыми программами или присоединяемыми скриптами (сценариями).

Обычно таким образом присоединяются наборы функций или **библиотеки функций**. К настоящему времени имеется очень много библиотек функций для Javascript, с помощью которых существенно облегчается и упрощается решение большого круга инженерных задач.

Поскольку очень важно уметь использовать присоединяемые библиотеки, то далее в этом подразделе вначале на простых примерах будет показан способ присоединения, а затем с применением библиотек мы научимся выполнять типовые математические алгоритмы и

отображать результаты в графическом виде.

Присоединяемый сценарий. Файл с присоединяемыми сценариями должен иметь расширение **js**. Присоединяется такой файл с помощью атрибута **src** тэга <script> (подобно тому, как используется такой атрибут в тэге <img> для помещения изображений на странице), например:

Листинг 3.7.7.

```
<html>
<head>
<title>Пример 3.7.7: присоединение файла с JavaScript-кодом</title>
<meta http-equiv=Content-Type content="text/html; charset=utf-8">
<script src="msg1.js"></script>
</head>
<body>
<h2>Пример 3.7.7: присоединение файла с JavaScript-кодом</h2>
</body></html>
```

Если файл msg1.js, присоединяемый в примере 3.7.7, будет содержать только лишь код JavaScript (без HTML-тегов), в данном случае – единственную строку с оператором:

```
alert("Сообщение из присоединенного файла с кодом на JavaScript!");
```

то эта строка вначале встроится в месте тэга с вызовом библиотеки и код выполнится также, как если бы находился там сразу же.

Не надо забывать об относительных и абсолютных путях при указании месторасположения присоединяемого файла. Как видно из данного примера при указанном относительном пути присоединяемый файл, должен находиться в той же папке, что и файл, к которому он присоединяется. Конечно, точно также, как и в случаях с изображениями, можно присоединять файл из любой папки, к которой вы имеете доступ, и даже из файла, размещенного на другом компьютере и другом сайте.

Возможность подобного присоединения используется для создания **библиотек**, содержащих многократно используемый код в виде набора функций на определенную тему. Присоединённые описания функций можно использовать точно так, как это делалось ранее с функциями, описания которых помещались между тэгами <script>...</script>.

Рассмотрим пример вычисления среднего дохода студента с использованием библиотеки. В папке lib размещены библиотеки функций, среди которых имеется библиотека функций для статистических расчетов (файл **stat_pvn.js**). Одна из функций этой библиотеки, **mean(x)** возвращает среднее арифметическое по входным данным, задаваемым одномерным массивом **x**. Ниже приводится программа для определения среднемесячного дохода с использованием функции

mean(x), имеющейся в библиотеке **stat_pvn.js**.

Листинг 3.7.8.

```
<html><head>
<title>Пример 3.7.8. Использование библиотек. Вычисление дохода</title>
<meta http-equiv=Content-Type content="text/html; charset=utf-8">
<script src="../lib/stat_pvn.js"></script>
<script>
function income(ob_frm) {
    var x1 = new Array();
    for (var i=0; i<ob_frm.elements.length-3; i++) {
      x1[i] = Number(ob_frm.elements[i].value);
    }
    ob_frm.result.value = mean(x1).toFixed(3);
}
</script>
</head>
<body style="background-color: #f8f8ff">
    <h2>Пример 3.7.8. Использование библиотечной функции.</h2>
    <h3>Вычисление среднемесячного дохода.</h3>
    <p>Введите полученные Вами суммы (вместо сумм по умолчанию)за месяцы:
    <form id="form1">
      январь: <input type="text" size=8 value="125.55">
      февраль: <input type="text" size=8 value="135.55">
      март: <input type="text" size=8 value="145.55"><br>
      апрель: <input type="text" size=8 value="155.55">
      май: <input type="text" size=8 value="165.55">
      июнь: <input type="text" size=8 value="175.55"><br><br>
      Средний доход за
      <script>document.write(form1.elements.length)</script> месяцев:
      <input type="text" size=8 id="result"><br><br>
      <input type="button" value="Вычислить" onClick="income(form1)"> 
      <input type="reset" value="Обновить">
    </form>
</body></html>
```

В присоединяемом файле `stat_pvn.js` находятся функции, среди которых есть функция для расчёта среднего арифметического (листинг 3.7.9):

Листинг 3.7.9.

```
/* Вычисление среднего арифметического по значениям одномерного массива*/
function mean(x)
{ var mean;
  var sum=x[0];
  for (var i=1; i<x.length; i++)
  {
    sum+=x[i];
  }
  return sum/x.length;
}
```

Для ввода исходных данных в примере 3.7.8 используется html-форма, с идентификатором `form1`. По этому идентификатору можно обращаться к форме, как к объекту в основной программе, которая находится в теле документа. К полям формы можно обращаться через массив `elements`, принадлежащий форме-объекту. Например, первое значение дохода можно извлечь из

элемента `form1.elements[0]`, второе — из `form1.elements[1]` и так далее. Последний элемент `form1.elements[form1.elements.length-1]` будет содержать значение «Обновить».

Имя формы указано в качестве фактического параметра при обращении к вспомогательной функции `income()`, размещённой в блоке `<head>...</head>`: `onClick="income(form1)".` В этой функции формируется одномерный массив исходных значений x1, который передаётся в качестве фактического параметра при вызове функции `mean(x1)`.

В функции `income()` к объекту необходимо обращаться уже по имени формального параметра. Например, к первому значению - `ob_frm.elements[0]` . Необходимо учесть, что длина массива элементов формы на три единицы больше числа исходных значений за счёт трёх полей в конце формы.

Вывод полученного значения осуществляется с использованием имени элемента, предназначенного для вывода, а не с использованием массива `elements[]`:

```
ob_frm.result.value = mean(x1).toFixed(3)
```

Такой способ вывода можно использовать, если вы присвоили идентификатор или имя соответствующему тэгу, в данном случае этому:

```
<input type="text" size=8 id="result">,
```

хотя можно использовать и массив `elements[]`:

```
ob_frm.ob_frm.elements[ob_frm.elements.length-3].value = mean(x1).toFixed(3)
```

Рекомендуется проверить оба варианта вывода (с использованием именованного тэга и с использованием массива `elements[]`).

Рассмотрим далее пример использования графической библиотеки.

Часто при выполнении вычислений полезно выводить результаты в виде графиков. Для Javascript имеется много графических библиотек, позволяющих быстро сформировать программу для построения графиков. Одну из таких библиотек (`JSChart`) можно найти и взять по адресу: http://www.jscharts.com/. Далее эта библиотека будет часто использоваться, поэтому следующий пример рекомендуется освоить для последующего самостоятельного применения библиотеки.

В примере показаны наиболее простой вариант графика. Более подробно о построении графиков и диаграмм с помощью библиотеки `JSChart` можно узнать по указанному выше адресу.

В примере листинга 3.7.10 имеется форма, предназначенная для ввода 12-ти значений

среднемесячной климатической температуры поверхности моря и вывода этих значений в виде графика. На графике по оси абсцисс откладываются номера месяцев, а по оси ординат — температура. Значения температуры отмечаются маркерами (кружками) и соединяются отрезками прямых линий. Вычерчивание графика осуществляется при нажатии на кнопку «Чертить». Значения можно исправить и после исправления повторить построение графика.

Листинг 3.7.10.

```
<html><head>
<title>Пример 3.7.10. Использование библиотек. Построение графика</title>
<meta http-equiv=Content-Type content="text/html; charset=utf-8">
<style type="text/css">
div.all {height: 450px; width: 870px;}
div.in {text-align: left; float:left; border:1px solid gray;
  margin-right: 5px; width: 200px; height: 450px;
}
div.out_grph {text-align: left; float:left;
  border: none; margin-left: 5px; width: 650px; height: 450px}
}
input{text-align: right; border: 1px solid gray;}
</style>
<script type="text/javascript" src="../lib_jscharts/jscharts.js"></script>
<script type="text/javascript" src="../lib_graph/lib_graph.js"></script>
<script>
function ingraph(frm)   {
    var x1 = new Array();
    for (var i=0; i<frm.elements.length-2; i++) {
      x1[i] = Number(frm.elements[i].value);
    }
    out_graph(x1);
}
</script></head>
<body style="background-color: #f8f8ff">
<div class="all">
    <h2>Пример 3.7.10. Использование графической библиотеки JSCharts</h2>
    <h3>Климатическая среднемесячная температура поверхности моря у городища
Мирмекий.</h3>
    <p>Введите (исправьте) значения среднемесячных температур:
    <div class="in">
    <form id="form1">
    <ol>
      <li><input type="text" size=8 value="-1.5"> январь
      <li><input type="text" size=8 value="-0.5"> февраль
      <li><input type="text" size=8 value="3.1"> март
      <li><input type="text" size=8 value="10.0"> апрель
      <li><input type="text" size=8 value="15.7"> май
      <li><input type="text" size=8 value="19.9"> июнь
      <li><input type="text" size=8 value="22.2"> июль
      <li><input type="text" size=8 value="21.5"> август
      <li><input type="text" size=8 value="16.6"> сентябрь
      <li><input type="text" size=8 value="10.4"> октябрь
      <li><input type="text" size=8 value="5.9"> ноябрь
      <li><input type="text" size=8 value="1.9">декабрь
    </ol>
      <input type="button" value="Чертить" onClick="ingraph(form1)"> 
      <input type="reset" value="Обновить">
    </form>
    </div>
```

```
    <div id="out_gr" class="out_grph"><h2>Место для графика</h2></div>
</div>
</body></html>
```

В программе — два подключаемых файла с библиотеками функций:

```
<script type="text/javascript" src="../lib_jscharts/jscharts.js"></script>
<script type="text/javascript" src="../lib_graph/lib_graph.js"></script>
```

Файл `jscharts.js` – это функции библиотеки `JSCharts`, которые необходимо получить по адресу:

http://www.jumpeyecomponents.com/my_account.login.details.245.get_trial.htm.

В файл помещена функция `out_graph()`, в которую помещено всё, связанное с построением графика (листинг 3.7.11). Построение графика в этой функции осуществляется путём последовательного вызова подпрограмм-функций библиотеки JSCharts. Назначение каждой вызываемой функции поясняется в комментариях.

Листинг 3.7.11.

```
/*Вычерчивание графика годового хода среднемесячной температуры воды*/
/*Входные данные: одномерный массив значений температуры*/
function out_graph(y) {
      // Создаём двумерный массив данных для графика:
      // первый столбец – номера месяцев; второй - температура
      var data1=new Array(12);
        for (var i=0; i<12; i++) {
      data1[i]=new Array(2);
      data1[i][0]=Number(i+1);
      data1[i][1]=Number(y[i]);
      }
      // Указываем html-блок для вывода графика (ID) и тип графика
      // и присваиваем имя объекту-графику (myChart)
      var myChart = new JSChart('out_gr', 'line');
      // Присваиваем имя массиву с исходными данными
      myChart.setDataArray(data1,'t_w');
      // Задаём отступы от осей и подписей
      myChart.setAxisPaddingTop(40);
      myChart.setAxisPaddingBottom(40);
      myChart.setTextPaddingBottom(10);
      // Задаём число делений на оси Y
      myChart.setAxisValuesNumberY(14);
      // Задаём начальное и конечное значения на оси Y
      myChart.setIntervalStartY(-2);
      myChart.setIntervalEndY(24);
      // Задаём число делений на оси X
      myChart.setAxisValuesNumberX(12);
      //и чтобы подписи отображались (true, иначе - false)
      myChart.setShowXValues(true);
      // Задаём надпись графику (пока что латиницей, про русс.- у преподавателя)
      myChart.setTitle('Climate the average temperature of the sea surface in the
settlement Mirmeky.');
      // и цвет надписи
      myChart.setTitleColor('#454545');
      // подписи осей, цвет подписей и осей
      myChart.setAxisNameX('Month');
      myChart.setAxisNameY('Temperature');
      myChart.setAxisValuesColor('#454545');
```

```
        // цвет соединительной линии графика
        myChart.setLineColor('#ff0000', 't_w');
        // тип маркера и его цвет
        for (var i=1;i<=12;i++) myChart.setTooltip([i]);
        myChart.setFlagColor('#ff0000');
        myChart.setFlagRadius(5);
        // графическая подложка
        myChart.setBackgroundImage('./3_7_img/3_7_10_bg.png');
        // размер области, занимаемой графиком
        myChart.setSize(650, 450);
        // чертить график
        myChart.draw()
}
```

3.7.6. Упражнения

Веб-страницы с выполненными упражнениями необходимо оформлять на своём персональном сайте так же, как это вы делали в предыдущих упражнениях. Не забывайте присоединять стили, созданные при освоении раздела, посвящённого HTML и CSS, чтобы сохранялось выбранное стилевое оформление всех страниц сайта. Помните также о тэге <meta ...> для указания кодировки UTF-8.

Примеры некоторых упражнений можно посмотреть у студента Петрова на сайте pvn.ho.ua.

Упражнение 3.7.1. Напишите html-документ с программой на языке Javascript, изменив пример из листинга 3.7.1. При этом необходимо сделать следующее:

1) воспроизведите пример из листинга 3.7.1;

2) измените обработчик соответствующего события так, чтобы при загрузке страницы в окне сообщения выводились ваши фамилия и имя;

3) ответьте на следующие контрольные вопросы, изучая код задания и экспериментируя с документом :

- Что произойдёт и почему, если ввести в качестве второго данного ноль или малое число **10e-310**, а первое - равное единице, и затем щёлкнуть по первому абзацу (с красным фоном)?
- Что произойдёт и почему, если ввести в качестве второго данного большое число **10e310**, а первое - равное единице, и затем щёлкнуть по первому абзацу (с красным фоном)?
- Что произойдёт и почему, если ввести оба исходных числа, но не вызывать событие первого (с красным фоном) абзаца, то есть не создавать переменную ch и не заполнять эту переменную частным от деления исходных данных, а сразу же начинать вычислять суммы, проводя мышкой над последним абзацем (с зелёным фоном)?
- Что произойдёт и почему, если в результате многократного вычисления переменной **s** вы получите число, превышающее максимум для чисел, поддерживаемых языком Javascript (для более быстрого получения такого числа вам нужно задать такие исходные числа,

частное от деления которых будет достаточно большим числом)? Какое значение в этом случае (в случае превышения максимума) выведется в диалоговом окне?

Упражнение 3.7.2. На основе примера из листинга 3.7.2 создайте свой html-документ и разберитесь с работой обработчиков событий в этом документе. Поэкспериментируйте с вычислениями, чтобы убедиться, что результаты получаются верными.

Особое внимание обратите на вычисление суммы исходных чисел. В этом месте уберите преобразования исходных данных с помощью функции Number() и сравните получаемый результат с предыдущим. Если результаты не равны, то объясните — почему? Это — контрольный вопрос, на который вам нужно написать ответ в разделе ответов на контрольные вопросы.

Затем к коду воспроизведённого вами примера 3.7.2 добавьте два блока с серым и светлосерым фонами:

1) для вычисления остатка от целочисленного деления исходных чисел;

2) для вычисления остатка от целочисленного деления накопленного произведения на накопленную сумму.

В результате добавления этих двух блоков при отображении документа в браузере у вас должны появиться два дополнительных абзаца с серым и светлосерым фоном. Щелкая мышкой (то есть вызывая событие Click), на странице должны появляться правильные результаты в соответствии с заданием.

Упражнение 3.7.3. Напишите html-документ, в котором бы использовались функции вычисления факториала из листингов 3.7.4 и 3.7.5. Число **n** должно вводиться из диалогового окна, а вывод каждого значения факториала необходимо вывести в отдельные параграфы с пояснительными текстами, чтобы было понятно, с помощью какого типа алгоритма получено значение факториала — итерационного или рекурсии.

Упражнение 3.7.4. Добавьте в пример `листинга 3.7.8` вычисление и вывод максимального и минимального месячных доходов с использованием функций max_arr(x) и min_arr(x) библиотеки **`stat_pvn.js`** и соответствующих им названий месяцев с использованием данных за 12 месяцев.

Упражнение 3.7.5. На основе примера из листинга 3.7.6 создайте свой html-документ и разберитесь, как он работает. Затем измените этот пример так, чтобы орбита спутника была не окружностью, а эллипсом, а в качестве спутника использовалась бы ваша собственная фотография (маленький портретик).

Упражнение 3.7.6. Воспроизведите пример из листинга 3.7.8, изменив его так, чтобы библиотека с функцией для вычисления суммы находилась в вашей собственной библиотеке (в файле с именем lib_Petrov.js, где вместо Petrov должна стоять ваша фамилия латиницей). После того, как убедитесь, ваша программа с использованием библиотечной функции работает, сделайте следующие изменения:

1) добавьте число месяцев так, чтобы средний доход считался с учётом поступлений за все 12 месяцев;

2) в качестве результатов в форму должны выводиться не только среднее значение, но и минимальное и максимальное значение дохода, с использованием двух функций: для нахождения минимального значения и для нахождения максимального значения. Эти функции должны быть помещены в вашу библиотеку lib_Petrov.js.

В этом же html-документе необходимо поместить ответы на семь контрольных вопросов (с 1-го по 7-й) из пункта 3.7.7. Контрольные вопросы.

Упражнение 3.7.7. Воспроизведите пример из листинга 3.7.11, изменив его так, чтобы функции `ingraph()` и `out_graph()` находились в вашей присоединяемой библиотеке `lib_Petrov.js`. Присоедините также графическую библиотеку `JSCharts`, которую можно найти и взять по адресу: http://www.jscharts.com/. Сделайте следующие изменения в функции `out_graph()`:

1) измените цвет осей и подписи делений на осях;

2) увеличьте радиус маркеров на графике;

3) измените цвет линии, соединяющей маркеры на графике;

4) замените фоновый рисунок, используемый в качестве подложки под областью, на которой расположен график, одной из ваших личных фотографий со страницы “О себе” вашего персонального сайта (можно другую фотографию, но принадлежащую лично вам);

В этом же html-документе необходимо поместить ответы на семь контрольных вопросов (с 8-го по 14-й).

3.7.7. Контрольные вопросы

1. Не менее пяти событий, связанных с манипулятором мышь?
2. Не менее трёх событий, связанных с клавиатурой?
3. Обработчик события?
4. Правила создания и использования функций?
5. Формальные и фактические параметры функций?
6. Локальная и глобальная области видимости переменных?

7. Оператор `return`?

8. Различие между итерационными и рекуррентными алгоритмами?

9. Рекурсивный вызов функции?

10. Использование формы для ввода данных?

11. Использование формы для вывода результатов?

12. Создание кнопки для запуска программы на стороне клиента?

13. Создание кнопки для запуска программы на стороне сервера?

14. Подключаемые сценарии и библиотеки функций?

3.7.8. Литература и веб-источники к подразделу.

[2] Дронов В.А. Javascript в Web-дизайне. – СПб.: БХВ-Петербург, 2004. - 880 с.

[3] Кингсли-Хью Э., Кингсли-Хью К. JavaScript 1.5: учебный курс. – СПб.: Питер, 2001. – 272 с.

[4] Полупанов В.Н. Программирование на стороне клиента. Конспект лекций. Керчь, КГМТУ, 2006. - 280 с.

Веб-источники

[6] http://pvn.ho.ua/pvn.org.ua/ – лекции на сайте преподавателя.

[8] http://analog.com.ua - HTML - справочник (наиболее простой).

[9] http://css.manual.ru/ - CSS-справочник (наиболее простой).

3.8. Объекты в JavaScript

3.8.1. Понятие объекта в Javascript

В Javascript можно работать с тремя типами объектов:

- объекты, встроенные в Javascript;
- объекты браузера;
- объекты HTML-документа;

- объекты, создаваемые программистом.

Со встроенными объектами, а именно со свойствами и методами объектов Math, Number, вы уже работали в подразделе 3.5. Более подробно о встроенных объектах можно узнать здесь [9, http://javascript.ru/manual].

С методами объектов браузера вы тоже знакомы: window.alert(), window.confirm(), window.prompt() (или просто: alert(), confirm(), prompt()), document.write, - методы, которые вы много раз использовали. Более подробно об использовании методов и свойств объектов браузера можно узнать здесь [9, http://javascript.ru/window].

Обращаться из Javascript-кода к объектам html-документа вы уже тоже умеете - посредством метода getElementById("id"), где "id" - идентификатор (имя) html-элемента. Вы также использовали свойство innerHTML для вывода значений в HTML-элемент из кода на языке Javascript. Более подробно об объектах объектной модели документа (DOM), работе с её объектами и innerHTML можно узнать здесь [9, http://javascript.ru/tutorial/dom, http://javascript.ru/tutorial/dom/intro#innerhtml].

В этом подразделе будут даны сведения о четвёртом типе объектов - объектах, создаваемых самостоятельно программистом. Сведения, достаточные для того, чтобы вы могли создавать и использовать свои собственные объекты, а также пользоваться объектами, создаваемые другими и размещаемыми в Сети.

Объект в Javascript можно представить как обычный ассоциативный массив или, иначе говоря, "хэш". Он хранит любые соответствия "ключ => значение" и имеет несколько стандартных методов.

Свойство объекта - это просто некоторая переменная, а **метод объекта** - это просто функция, которые добавляются в качестве элементов ассоциативного массива.

Основное преимущество объекта перед обычными переменными и функциями в том, что в объекте можно накапливать и хранить все свойства и методы, свойственные изучаемому объекту, обращаясь к ним по мере необходимости посредством единообразного и эффективного механизма.

Более подробно и в то же время понятно об объектах можно узнать здесь [9 , http://javascript.ru/tutorial/object/intro]. Нам же достаточно будет сведений, изложенных в этом подразделе.

3.8.2. Создание объектов в Javascript

Для начала объясним, как объекты действуют в JavaScript. В следующих примере создаётся переменная-объект и затем для этой переменной задаётся три свойства и один метод.

Листинг 3.8.1.

```
var myObj  = new Object;
myObj.a    = 5;
myObj['b'] = 10;
```

```
myObj.c    = 20;
myObj.getSum = function(){
  alert(this.a+this.b+this.c);
};
alert(myObj.a);
alert(myObj['a'])
myObj.getSum()
//или
var myObj = {a:5, b:10, c:20,
 getSum:function() { alert(this.a+this.b+this.c); }};

alert(myObj.a);
alert(myObj['a'])
myObj.getSum()
```

В примере создаётся переменная-объект myObj двумя способами. Первый способ использует синтаксис new Object(). Второй способ использует сокращенную нотацию и делает то же самое. В результате каждого из этих способов мы имеем абсолютно одинаковую переменную myObj, которая содержит три переменные (свойства) для чисел: a, b и c. Задаются свойства двумя способами: myObj['b'] или myObj.b. Для доступа к любому из свойств можно использовать способы: myObj.a или myObj['a'].

Объект myObj имеет также в качестве свойства функцию getSum. Такое свойство называют методом объекта. Обращение к getSum осуществляется таким же образом, как и к свойствам a, b и c: myObj.getSum(). Функция getSum() обращается к переменным в переменной-объекте myObj с помощью ключевого слова this. При выполнении кода внутри функции в объекте используется ключевое слово this для ссылки на сам объект myObj (что такое ключевое слово this можно вспомнить здесь: [9, http://javascript.ru/tutorial/object/thiskeyword]).

Оператор delete можно использовать для удаления всего объекта: delete myObj.

С помощью оператора delete можно удалить также любое свойство или метод объекта: delete myObj.a; delete myObj.getSum .

Объектами, создаваемыми описанными выше способами можно пользоваться, если объект один или их мало, но такие способы создания объектов имеют недостаток, - если объектов много, то приходится повторять слишком много данных каждый раз, когда мы хотим создать новый объект. Покажем это на примере. Допустим, нам нужно создать объект для описания рыб Керченского пролива:

Листинг 3.8.2.

```
var KerchFish = {
  name:     'пиленгас',
  type:  'рыбы',
  classis:   function(){alert(this.name+': вид - '+this.type+ '; класс -
лучепёрые'); },
  subclassis:   function(){alert('подкласс - новопёрые'); },
  infraclassis:      function(){alert('инфракласс - костистые'); }
```

```
}
```

Мы создали переменную-объект KerchFish, в которой записаны данные об изучаемой рыбе: вид (type)- рыбы, наименование (name) - пиленгас. Однако, если потребуется создать данные для других наименований рыб того же класса, подкласса и инфракласса, то понадобится вводить одни и те же данные много раз. В этом случае, целесообразно использовать так называемый **конструктор**, предусмотренный в рамках объектно-ориентированного подхода, реализованного в Javascript. Вместо перепечатки всякий раз всего объекта можно определить функцию-конструктор, которая создает объект при вызове с использованием ключевого слова new:

Листинг 3.8.3.

```
function KerchFish(name){
  this.name  = name;
  this.type  = 'рыбы';
  this.classis = function(){alert(this.name+': вид - '+this.type+ '; класс -
лучепёрые'); }
  this.subclassis = function(){ alert('подкласс - новопёрые'); },
  this.infraclassis = function(){ alert('инфракласс - костистые'); }
}

var pilengas = new KerchFish('пиленгас');
var mullet   = new KerchFish('кефаль');
var anchovy = new KerchFish('хамса');

pilengas.classis();  // выводит 'пиленгас: вид - рыбы; класс - лучепёрые'
mullet.subclassis(); // выводит 'подкласс - новопёрые'
pilengas. infraclassis() // выводит 'инфракласс костистые'
alert(anchovy.type); // выводит 'рыбы'
```

В этом примере вначале создана одна функция KerchFish, а затем создаются три новых переменных-объектов с помощью этой функции: pilengas, mullet и anchovy. Каждый из этих объектов имеет одинаковые функции-методы: classis, subclassis и infraclassis, и каждая из них имеет свойство type, заданное как 'рыбы'.

Принято говорить, что pilengas, mullet и anchovy являются экземплярами одного объекта: KerchFish. Функция KerchFish называется **конструктором**. Мы передаем ей переменную name и используем ее для задания наименования рыб: this.name.

Использование конструктора облегчает создание множества экземпляров похожих объектов и такой способ создания объектов более приемлем, чем первый из рассмотренных примеров, когда много похожих объектов создаётся многократной набивкой.

Однако, есть недостаток и у этого способа создания объектов, связанный с ресурсами компьютера. Дело в том, что при создании каждого нового экземпляра объекта с помощью конструктора в действительности создается новая копия этой функции-конструктора со всеми свойствами и методами. Так как переменная type и функции classis, subclassis и

infraclassis являются идентичными, то нам в действительности требуется только одна копия каждой функции. Чтобы избежать такого повторения можно использовать так называемое **прототипирование** [9, http://javascript.ru/tutorial/object/inheritance].

3.8.3. Прототипы, изменение и добавление свойств и методов объекта

Рассмотрим способ наследования, изменения и добавления свойств и методов объектов с использованием прототипирования. Для справки: слово **прототип** (от греч. protótypon — прообраз) означает первообраз, основной образец, или в нашем случае - объект-оригинал, с которого мы будем делать копии).

В Javascript предусмотрено свойство с названием prototype - оно указывает, откуда брать прототип при создании новой копии объекта. При этом реализуется **одно из основных отличий объектов от других переменных, а именно: копия (дочерний объект) наследует все свойства и методы прототипа (родительского объекта)**. Например, если указать это свойство для некоторого имеющегося объекта pilengas: pilengas.prototype = kerch_fish, где kerch_fish также имеющийся объект, то этим предписывается, что копии объекта pilengas в качестве прототипа будут использовать объект kerch_fish.

Покажем использование prototype на примере.

Листинг 3.8.4.

```
<html><head>
<title>Пример 3.8.4. Объекты в Javascript. Прототипы</title>
<meta http-equiv=Content-Type content="text/html; charset=utf-8">
</head>
<body style="background-color: #f8f8ff">
    <h2>Пример 3.8.4. Объекты в Javascript. Прототип объектов</h2>
<script>
// создаём конструктор KerchFish()
function KerchFish(name){
  this.name  = name;
  this.type  = 'рыбы';
  this.classis = function(){alert(this.name+': вид - '+this.type+ '; класс -
лучепёрые'); }
  this.subclassis = function(){ alert('подкласс - новопёрые'); },
  this.infraclassis = function(){ alert('инфракласс - костистые'); }
}
// создаём объект kerch_fish
var kerch_fish = new KerchFish('Рыбы Керченского пролива');
// создаём конструктор pilengas()
function pilengas(name){
  this.name  = name
}
// указываем, что прототипом для pilengas будет объект kerch_fish
// и создаём экземпляр объекта kerch_pilengas, наследующего свойства и методы
//kerch_fish
pilengas.prototype = kerch_fish;
kerch_pilengas = new pilengas('пиленгас из Керченского залива');

function mullet(name){
```

```
  this.name  = name
}
mullet.prototype = kerch_fish;
kerch_mullet = new mullet('кефаль из Керченского залива');

function anchovy(name){
  this.name  = name
}
anchovy.prototype = kerch_fish;
kerch_anchovy = new anchovy('хамса из Керченского пролива');

</script>
<h3>Нажмите на нужную кнопку, чтобы вызвать свойство или метод копии объекта
KerchFish</h3>
<p>Копия объекта "pilengas", прототип - объект kerch_fish:
<button onClick="alert(kerch_pilengas.name)">Наименование</button>
<button onClick="alert(kerch_pilengas.type)">Вид</button>
<button onClick="kerch_pilengas.classis()">класс</button>
<button onClick="kerch_pilengas.subclassis()">подкласс</button>
<button onClick="kerch_pilengas.infraclassis()">инфракласс</button>
</p>
<p>Копия объекта "mullet", прототип - объект kerch_fish:
<button onClick="alert(kerch_mullet.name)">Наименование</button>
<button onClick="alert(kerch_mullet.type)">Вид</button>
<button onClick="kerch_mullet.classis()">класс</button>
<button onClick="kerch_mullet.subclassis()">подкласс</button>
<button onClick="kerch_mullet.infraclassis()">инфракласс</button>
</p>
<p>Копия объекта "anchovy", прототип - объект kerch_fish:
<button onClick="alert(kerch_anchovy.name)">Наименование</button>
<button onClick="alert(kerch_anchovy.type)">Вид</button>
<button onClick="kerch_anchovy.classis()">класс</button>
<button onClick="kerch_anchovy.subclassis()">подкласс</button>
<button onClick="kerch_anchovy.infraclassis()">инфракласс</button>
</p>
</body></html>
```

Последовательность создания экземпляров объектов на основе объекта-прототипа, описывающих конкретных рыб в примере листига 3.8.4, описана в комментариях для пиленгаса. Объект `kerch_pilengas` создаётся с помощью конструктора `pilengas`, при этом с помощью этого конструктора в объекте создаётся лишь одна переменная-свойство `name`. Свойство `type` и методы `classis`, `subclassis` и `infraclassis` добавляются в объект `kerch_pilengas` посредством прототипирования из ранее созданного объекта `kerch_fish`. Для кефали и хамсы экземпляры объектов создаются аналогичным образом.

Так как у всех созданных объектов один и тот же прототип, то изменения свойств и методов в прототипе вызовут одни и те же изменения во всех дочерних объектах.

Есть один несущественный недостаток при использовании прототипирования для создания похожих экземпляров объектов - создаётся промежуточный объект-оригинал `KerchFish`, который не всегда нужен по смыслу решаемой задачи, как это оказалось в нашем примере. Способ преодоления этого недостатка описывается в [9, http://javascript.ru/tutorial/object/inheritance].

Рассмотрим, далее, способ добавления свойств и методов объекам. Перепишем функцию-конструктор KerchFish следующим образом:

Листинг 3.8.5.

```
function KerchFish(name){
  this.name  = name;
}
  KerchFish.prototype.type  = 'рыбы';
  KerchFish.prototype.classis = function(){alert(this.name+': вид - '+this.type+ ';
класс - лучепёрые'); }
  KerchFish.prototype.subclassis = function(){ alert('подкласс - новопёрые'); },
  KerchFish.prototype.infraclassis = function(){ alert('инфракласс - костистые'); }
```

Код этого способа создания прототипа немного отличается от того, что приведён в примере листинга 3.8.4. Вместо объявления всех свойств и методов внутри функции-конструктора KerchFish, они теперь объявляются с помощью KerchFish.prototype. Но этот способ приводит к тому же результату, что и в примере листинга 3.8.4.

Способ, приведённый в листинге 3.8.5 можно использовать для добавления свойств и методов объекту-прототипу, а значит, и всем дочерним объектам.

3.8.4. Переменные Private, Public и Static

Способ объявления переменных в объекте определяет, какие методы этого объекта можно использовать для доступа к этим переменным. В JavaScript при работе с объектами используется пять уровней методов и свойств.

1. Скрытый (Private) - объявляется с помощью var variableName или function functionName внутри объекта. Могут быть доступны только другим скрытым или привилегированным функциям.
2. Открытый (Public) - объявляется с помощью this.variableName внутри объекта. Может изменяться любой функцией или методом.
3. Привилегированный (Privileged) - объявляется с помощью this.functionName = function(){...} внутри объекта. Доступна для любой функции или метода и может обращаться или изменять любую скрытую переменную.
4. Прототипированный (Prototype) - объявляется с помощью Class.prototype.variableName или Class.prototype.functionName. Объявленные таким образом функции будут иметь доступ к любой открытой или прототипированной функции. Попытки изменить созданную таким образом переменную будут вместо этого создавать новую открытую переменную на объекте, а прототипированная переменная будет недоступна.
5. Статический (Static) - объявляется с помощью Class.variableName или Class.functionName. Может изменяться любой функцией или методом. Такой метод используется редко.

Чтобы понять различия между уровнями, рассмотрим пример:

Листинг 3.8.6.

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

```
<title>Пример 3.8.6. Объекты в Javascript. Переменные Private, Public и
Static</title>
<meta http-equiv=Content-Type content="text/html; charset=utf-8">
<style>
  div {
  width:100px;
  height:20px;
  display:inline;
  vertical-align:middle;
  border:1px lightblue solid
  }
</style>
</head>
<body style="background-color: #f8f8ff">
    <h2>Пример 3.8.6. Объекты в Javascript. Переменные Private, Public и
Static</h2>
<script>
function KerchFish(name, n_fishs){
  /*
  Конструктор: при создании объекта выполняется находящийся здесь код
  */
  /* Скрытые переменные и функции доступны только скрытым или привилегированным
  функциям. Отметим, что  'name', передаваемая в KerchFish, уже являются
  скрытой переменной.
  */
  var age  = 0;
  var fins = 6;
  function growOlder(){
    age++;
  }
  /*
  Открытые переменные доступны открыто или скрыто
  */
  n_fishs++;
  this.n_fishs=n_fishs;
  this.name_fish = name;
  this.weight = 100;
  this.length = 60;
  this.n_fins = fins;
  /*
  Привилегированные функции доступны открыто или скрыто.
  Могут обращаться к скрытым переменным.
  Невозможно изменить, можно только заменить открытой версией
  */
  this.fun_age = function(){
    if(age==0) {
      this.length+=150;
      this.weight=this.weight+=175;
    }
    else {
      this.length+=125;
    }
    //age++;
    growOlder();
    this.weight=this.weight+100*age;
    this.fish_age=age;
  }
} // end коструктора
/*
Прототипированные функции доступны открыто
```

```
*/
KerchFish.prototype = {
  classis:   function(){return this.cl=this.name_fish+': вид - '+this.type+ ';
класс - лучепёрые'; },
  subclassis:   function(){return this.subcl='подкласс - новопёрые'; },
  infraclassis:      function(){return this.infracl='инфракласс - костистые'; }
}
/*
Прототипированные переменные доступны открыто.
Нельзя перезаписать, только заменить открытой версией
*/
KerchFish.prototype.type = 'рыбы';
/*
Статические переменные и функции доступны открыто
*/
n = 0;
/*
Используем объект, заодно проверяем правильность уровней доступа
*/
kerch_fish = new KerchFish('Пиленгас',n);
</script>
<h3>Нажмите на нужную кнопку, чтобы вызвать свойство или метод объекта
kerch_fish<br />
(нажав кнопку "возраст" вы увеличиваете возраст и показатели,зависящие от
возраста)</h3>
<button
onClick="getElementById('nm_f').innerHTML=kerch_fish.name_fish">Наименование</butto
n>
<div id="nm_f"></div><br />
<button onClick="getElementById('ty').innerHTML=kerch_fish.type">Вид</button>
<div id="ty"></div><br />
<button
onClick="getElementById('cl').innerHTML=kerch_fish.classis()">класс</button>
<div id="cl"></div><br />
<button
onClick="getElementById('subcl').innerHTML=kerch_fish.subclassis()">подкласс</butto
n>
<div id="subcl"></div><br />
<button
onClick="getElementById('infracl').innerHTML=kerch_fish.infraclassis()">инфракласс<
/button>
<div id="infracl"></div><br />
<button onClick="getElementById('n_f').innerHTML=kerch_fish.n_fishs">количество
рыб</button>
<div id="n_f"></div><br />
<button
onClick="getElementById('n_fns').innerHTML=kerch_fish.n_fins">плавников</button>
<div id="n_fns"></div><br />
<button onClick="kerch_fish.fun_age();
getElementById('f_a').innerHTML=kerch_fish.fish_age">возраст</button>
<div id="f_a"></div><br />
<button onClick="getElementById('w').innerHTML=kerch_fish.weight">вес</button>
<div id="w"></div><br />
<button onClick="getElementById('l').innerHTML=kerch_fish.length">длина</button>
<div id="l"></div>
</body></html>
```

Поэкспериментировав с примером можно убедиться, что существует несколько уровней доступа. Как упоминалось выше, все скрытые, привилегированные и открытые функции-методы и

переменные-свойства копируются всякий раз, когда создается новый экземпляр объекта.

Обычно почти всё, что нужно сделать, можно реализовать с помощью прототипированных и открытых переменных. Поэтому лучше избегать использования скрытых, привилегированных и статических переменных, если это не требуется для решения вашей задачи.

3.8.5. Упражнения

Упражнение 3.8.1. Создайте два объекта двумя способами, использованными в листинге 3.8.1 с различными именами, но с одинаковыми свойствами и методами, такими же, как в листинге 3.8.2. Убедитесь, обратившись к методам и свойствам этих двух объектов, что свойства и методы этих объектов приводят дают одинаковые результаты.

В этом же html-документе поместите ответы на первые восемь контрольных вопросов (с 1-го по 8-й).

Упражнение необходимо сохранить в файле exercise_3_8_1.html и оформить его отображение на своём сайте так, как это вы делали с предыдущими упражнениями.

Упражнение 3.8.2. Создайте такой же объект, приведённый в листинге 3.8.1, но с использованием функции-конструктора.

В этом же html-документе поместите ответы на первые семь контрольных вопросов (с 9-го по 15-й).

Упражнение необходимо сохранить в файле exercise_3_8_2.html и оформить его отображение на своём сайте так, как это вы делали с предыдущими упражнениями.

Упражнение 3.8.3. В примере листинга 3.8.6 предусмотрена, но "не работает" переменная-свойство для подсчёта рыб - при нажатии на кнопку "количество рыб" сохраняется "1" в поле вывода, хотя при каждом нажатии количество рыб должно увеличиваться на единицу. На основе кода из листинга 3.8.6 создайте документ, внеся соответствующие исправления, чтобы при каждом нажатии на кнопку "количество рыб" значение в поле вывода увеличивалось бы на единицу

В этом же html-документе поместите ответы на первые шесть контрольных вопросов (с 16-го по 21-й).

Упражнение необходимо сохранить в файле exercise_3_8_3.html и оформить его отображение на своём сайте так, как это вы делали с предыдущими упражнениями.

3.8.6. Контрольные вопросы

1. Что такое объект?
2. Что такое свойство объекта?
3. Что такое метод объекта?

4. Перечислите объекты, встроенные в Javascript.
5. Как округлить значение вещественного числа с точностью до n знаков после запятой, используя метод объекта Number, встроенного в Javascript.
6. Назовите известные вам объекты браузера и их методы.
7. Назовите объекты HTML-документа, если web-странице имеется заголовок, параграф, рисунок и форма с элементами управления <input type="text"> и <button>...</button>.
8. Как создать объект без использования конструктора?
9. Как создаётся объект с использованием функции-конструктора?
10. Как добавить свойство в объект?
11. Способы доступа к свойству объекта?
12. Как добавляется метод в объект?
13. Способ доступа к методу объекта?
14. Как удалить объект?
15. Как удалить свойство или метод из объекта?
16. Что такое прототип объекта?
17. Как создаются новые объекты с использованием прототипирования?
18. Как добавляется новое свойство с использование прототипирования?
19. Как добавляется новый метод с использованием прототипирования?
20. Как создаётся скрытая для доступа (Private) переменная в объекте?
21. Как создаётся открытая для доступа (Pablic) переменная в объекте?

3.8.7. Литература и веб-источники к подразделу.

[2] Дронов В.А. Javascript в Web-дизайне. – СПб.: БХВ-Петербург, 2004. - 880 с.

[3] Кингсли-Хью Э., Кингсли-Хью К. JavaScript 1.5: учебный курс. – СПб.: Питер, 2001. – 272 с.

[4] Полупанов В.Н. Программирование на стороне клиента. Конспект лекций. Керчь, КГМТУ, 2006. - 280 с.

Веб-источники

[6] http://pvn.ho.ua/pvn.org.ua/ – лекции на сайте преподавателя.

[8] http://analog.com.ua - HTML - справочник (наиболее простой).

[9] http://css.manual.ru/ - CSS-справочник (наиболее простой).

[15] http://www.intuit.ru/department/internet/jscs/ - Учебный курс на INTUIT.ru. Майк Вест, лекция 7 «Объекты в Javascript».