Дмитриева Марина Валерьевна

JAVA SCRIPT. ЗАНЯТИЕ 1. ПРОСТЫЕ СЦЕНАРИИ (функции, графика, простые анимации, формы)

От редакции:

Читатели уже знакомы с Мариной Валерьевной Дмитриевой, автором серии публикаций в нашем журнале. Недавно в издательстве BHV вышла новая книга М.В. Дмитриевой по JavaScript. В этой книге автор рассматривает язык JavaScript не только как инструмент для оформления web-страниц, но и как средство обучения программированию. Редакция считает, что такой подход представляет большой интерес для преподавания информатики в школе по следующим причинам:

– традиционный путь обучения программированию, требующий достаточно больших затрат учебного времени, оказался далек от интересов большинства учеников;

– в то же время, работа в Интернет имеет высокую социальную значимость практически для всех учеников, поэтому идея переноса акцента в обучении программированию на создание программ, работающих в Интернет, кажется весьма перспективной с точки зрения сохранения традиций хорошей программистской подготовки в российской школе.

Другие доводы в пользу такого построения курса программирования – возможность естественной демонстрации и обмена продуктами своего труда. Действительно, любой браузер является в то же самое время интерпретатором JavaScript. Таким образом, достаточно подготовить текст программы в любом текстовом редакторе, а затем дать этому файлу расширение .htm, как он превращается в готовый к исполнению модуль. Результаты его работы покажет браузер при вызове html-файла.

Тем самым, появляется возможность применить любые, скромные или весьма продвинутые, знания по программированию для создания Web-сайтов. Более того, работу по созданию сайтов можно естественным образом распределить между ее участниками работы, как, например, при выпуске школьной стенгазеты. Тот, кому ближе художественная часть, займется оформлением с применением простых элементов языка, а тот, кого интересует программирование, сделает содержательные модели или инструменты.

По просьбе редакции М.В. Дмитриева подготовила новый курс в Заочной школе современного программирования, который можно было бы назвать «Основы программирования через создание Web-приложений» или короче «Создание Web-приложений».

Этот курс является естественным продолжением курса Н.Н. Паньгиной по VisualBasic, который создавался под девизом «Программирование для непрограммистов», и основное назначение которого было научить созданию программ с современным интерфейсом, основываясь на небольшом объеме знаний, который получают школьники по языку Basic.

В этом курсе ученик тоже научится пользоваться всеми стандартными объектами, определяющими особенности интерфейса современных приложений, только теперь это будут средства языка JavaScript. Сверх того, ученик получит возможность уча-

ствовать в создании общего Web-сайта. Отсюда девиз нового курса – «Создаем наш Web-сайт». Особенностью этого курса будет то, что каждый ученик получит индивидуальное задание, а результаты выполнения этого задания станут частью проекта по созданию нового учебного сайта и будут помещаться в Интернет.

Занятия в школе по этому курсу начнутся, как всегда, 1 октября, а заявки в школу можно подавать с момента выхода журнала. Предварительные публикации занятий ЗСШП по курсу в журнале начинаются, как всегда, с первого номера журнала. Особенностью этих публикаций будет специальный набор многовариантных заданий, который учитель может использовать на уроках информатики.

Путешествуя по Всемирной Паутине (WorldWideWeb или WWW), вы обратили внимание на различный вид Web-страниц. Некоторые привели вас в восторг как оформлением, так и содержанием, другие оставили равнодушным, а иные и просто совсем не понравились.

Если вы решили готовить свои Web-публикации, то для этого можно выбрать различные средства. Самый простой и быстрый способ – использование различных Мастеров Web-страниц. В этом случае Вам следует лишь подготовить текст, который требуется разместить на странице, рисунки для фона и переднего плана, ссылки и, может быть, другую дополнительную информацию. Дальнейшая работа состоит в ответе на те вопросы, которые задает Мастер Web-страниц. После ответа на все вопросы Мастер сформирует страницу. Процесс создания страницы не займет много времени.

Другой путь – создание Web-страницы непосредственно на сервере, предоставляющем такие услуги. После регистрации на сервере пользователю предоставляются средства для создания Web-страниц. Вы можете выбрать стиль страницы, графические изображения, которые могут располагаться в определенном месте страницы, цветной фон или фоновый рисунок, разделительные линии, анимированное изображение. Иногда на создаваемую страницу можно поместить дополнительные объекты, такие как счетчик посещений, гостевую книгу и т.п. Такая технология создания Web-страниц может вполне удовлетворить начинающих пользователей.

Все пользователи, владеющие основами работы с программами пакета MS Office, становятся потенциальными разработчиками Web-страниц. Можно, например, подготовить документ в текстовом процессоре MS Word, а затем преобразовать его в Web-страницу. Созданную с помощью Power Point презентацию можно преобразовать в Web-сайт.

Визуальная среда создания и обслуживания Web-сайтов Front Page специально разработана для того, чтобы облегчить непрофессионалам процесс создания Web-сайтов.

Все перечисленные средства не предполагают умения (или желания) пользователей программировать. Программистская деятельность – очень увлекательное занятие, создание Web-приложений не является исключением. Обсудим вопросы создания Web-приложений на стороне клиента. Клиентская часть среды проектирования Web-приложений содержит следующие компоненты:

1. Браузер (или обозреватель), отображающий HTML-документ на экране монитора и являющийся пользовательским интерфейсом для Web- приложений.

2. Язык HTML (Hyper Text Markup Language), с помощью которого создаются Web-страницы.

3. Языки сценариев. В настоящее время в качестве стандарта принят язык JavaScript.

4. Клиентские расширения, такие как элементы управления ActivX, Java-аплеты, подключаемые модули и др.

Для понимания рассматриваемого материала предполагается знание основ HTML. Напомним некоторые принципы построения HTML-документов.

ОСНОВНЫЕ ПРИНЦИПЫ ПОСТРОЕНИЯ HTML-ДОКУМЕНТОВ

1. Символы, которые управляют отображением текста и при этом сами не отображаются на экране, в HTML принято называть тегами (tag-ярлык, признак). Все теги языка HTML выделяются символами-ограничителями (< >), между которыми записывается идентификатор (имя) тега и, возможно, параметры. Название тегов и их параметров можно записывать на любом регистре.

2. Большинство тегов записывается попарно, то есть для определенного тега, называемого *открывающим*, в документе имеется соответствующий *закрывающий тег*. По правилам HTML закрывающий тег выглядит так же, как открывающий, но с символом / (прямой слэш) перед именем тега. Закрывающий тег не использует параметров.

3. Теги, которые нуждаются в завершающих тегах, иногда называют *тегами-контейнерами*; все, что находится между открывающим и закрывающим тегом, называется *содержимым тега*.

4. Иногда закрывающий тег можно опускать. Однако такая практика не может быть рекомендована.

5. Ряд тегов в принципе не нуждается в завершающих тегах. Примером может быть тег вставки изображений, принудительного перевода строки и т.д.

6. Тег комментария имеет более сложную структуру < !-- и -->.

ОБЩИЕ ПРАВИЛА ЗАПИСИ ПАРАМЕТРОВ

1. Теги могут быть записаны с параметрами. Набор допустимых параметров индивидуален для каждого тега.

2. После имени тега могут следовать параметры, которые отделяются друг от друга пробелами.

3. Порядок следования параметров произволен.

4. Многие параметры требуют указания их значений. Если параметр требует значения, то оно указывается после названия параметра через знак равенства.

5. Значение параметра может быть записано в кавычках или без них. Если в значении параметра имеются пробелы, то кавычки обязательны.

6. В значениях параметров иногда важен (в отличие от названия тегов и самих параметров) регистр записи.

7. Некоторые параметры не имеют значений или могут записываться без них, принимая значения по умолчанию.

8. Существует ряд параметров, единый практически для всех тегов.

ОБЩИЕ ПРАВИЛА ИНТЕРПРЕТАЦИИ ТЕГОВ ОБОЗРЕВАТЕЛЯМИ

Если программа на языке программирования содержит ошибку, то при обработке ее компилятором выдается соответствующее сообщение об ошибке. В отличие от языка программирования, в HTML не принято реагировать на неправильную запись тегов. Неверно записанный тег или параметр тега игнорируется обозревателем, который интерпретирует текст, написанный на HTML. Это правило касается не только ошибочных тегов, но и тех тегов, которые не распознаются данной версией браузера. Примером могут служить теги, предложенные и реализованные для одного браузера, но неизвестные для другого.

СТРУКТУРА ДОКУМЕНТА

Первым тегом, с которого начинается описание документов HTML, является тег <HTML>. Завершить описание документа должен тег </HTML>. Между тегами <HTML> и < /HTML>. Располагается сам документ. Документ может состоять из двух разделов:

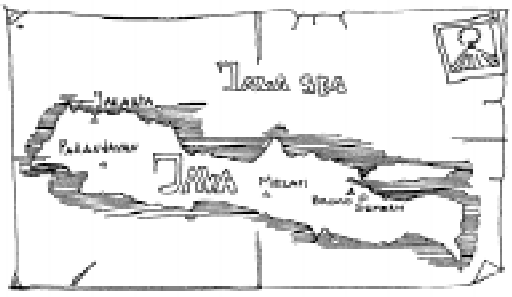
- Раздел заголовка (начинается тегом <HEAD>).
- Раздел содержательной части, или тело документа (начинается тегом <BODY>).

Для документов, описывающих фреймовую структуру, вместо раздела <BODY>, используется раздел <FRAMSET>. Сначала рассмотрим правила составления разделов <HEAD> и <BODY>.

Тег-контейнер <TITLE> служит для того, чтобы дать документу название, которое обычно показывается в заголовке окна браузера. Раздел <BODY> имеет ряд параметров, ни один из которых не является обязательным.

Простейший HTML-документ представляет собой обычный текстовый файл, для просмотра и редактирования которого можно воспользоваться любым редактором.

ЯЗЫК СЦЕНАРИЕВ JAVASCRIPT: ОСНОВНЫЕ ПОНЯТИЯ



Язык сценариев JavaScript предназначен для создания интерактивных HTML-документов. С помощью сценариев поддерживается диалог с пользователем, обеспечивается привлекательный вид Web-страниц, осуществляется навигация по страницам сайта, поиск элементов на странице и многое другое. Основой языка является понятие объекта. JavaScript внедрен в HTML, обеспечивает работу в среде, поддерживаемой браузерами.

Язык JavaScript позволяет обрабатывать исходные данные, представленные с помощью различных элементов управления, создавать тестирующие программы, осуществлять контроль вводимых данных и многое другое.

JavaScript позволяет создавать приложения, выполняемые как на стороне клиента, так и на стороне сервера. При разработке приложений обоих типов используется так называемое ядро, в котором содержатся определения стандартных объектов. Клиентские приложения выполняются браузером на машине пользователя.

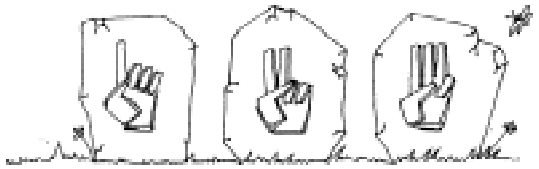
Программа (сценарий) на языке JavaScript представляет собой последовательность операторов, которые обрабатываются встроенным в браузер интерпретатором. Надо стремиться к тому, чтобы написанные сценарии корректно выполнялись в любом браузере. На первоначальном этапе обучения удовлетворить этому требованию сложно. Предлагаемые сценарии отлаживались в Internet Explorer версии 4.01 и выше.

Если несколько операторов располагаются на одной строке, то между ними следует поставить знак «точка с запятой». Если каждый оператор располагается на одной строке, то разделитель можно не писать. Один оператор может располагаться на нескольких строках.

Согласно принципам структурного программирования, программу рекомендуется записывать таким образом, чтобы в записи программы была отражена ее блочная структура. Это облегчает исследование программы и поиск ошибок.

В программах на JavaScript можно использовать комментарии. Для того чтобы задать комментарий, располагающийся на одной строке, достаточно перед текстом комментария поставить две косые черты. Если же поясняющий текст занимает несколько строк, то его следует заключать между символами /* и */. В JavaScript строчные и прописные буквы алфавита считаются разными символами.

ЛИТЕРАЛЫ



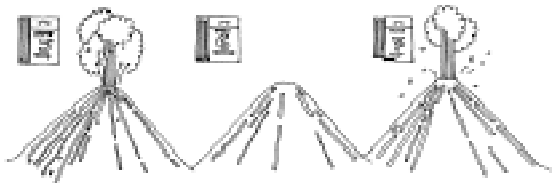
Простейшие данные, с которыми может оперировать программа, называются литералами. Литералы не могут изменяться. Литералы целого типа могут быть заданы в десятичном (по основанию 10), шестнадцатеричном (по основанию 16) или восьмеричном (по основанию 8) представлении.

Литерал целого типа в десятичном представлении записывается как последовательность десятичных цифр со знаком или без него, например, 15, 123, –156, +3567.

Кроме целых и вещественных значений, в языке JavaScript могут встречаться так называемые логические значения. Существуют только два логических значения: истина и ложь. Первое представляется литералом **`true`**, второе – **`false`**. В некоторых реализациях JavaScript может быть использована единица в качестве **`true`** и ноль в качестве **`false`**.

Строковый литерал представляется последовательностью символов, заключенной в одинарные или двойные кавычки. Примером строкового литерала может быть строка "результат" или 'результат'.

ПЕРЕМЕННЫЕ



Переменные используются для хранения данных. Переменные в сценарии представляются с помощью идентификаторов. Идентификатор должен начинаться с буквы латинского алфавита, либо с символа подчеркивания. Далее может следовать последовательность, содержащая буквы латинского алфавита, цифры или знак подчеркивания, например, test1, _my_test, test_1. Тип переменных зависит от хранимых в них данных. При изменении типа данных меняется тип переменной. Определить переменную можно с помощью оператора **`var`**, например, **`var test1`**. Тип переменной test1 не определен. Тип переменной станет известен только после присвоения переменной некоторого значения.

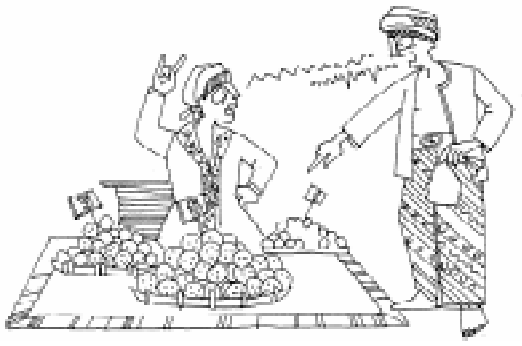
Оператор **`var`** можно использовать и для инициализации переменной, например, конструкцией **`var test2=276`** определяется переменная **`test2`** и присваивается ей значение **`276`**.

ОПЕРАТОР ПРИСВАИВАНИЯ

Значение переменной изменяется в результате выполнения оператора присваивания. Оператор присваивания может быть использован в любом месте программы. Он может изменить не только значение, но и тип переменной. Оператор присваивания выглядит так **`a=b`**, где **`a`** – переменная, которой мы хотим задать некоторое значение, **`b`** – выражение, которое определяет новое значение переменной.

Переменные, описанные в сценарии как в части <HEAD>, так и в части <BODY>, имеют одну и ту же область действия, доступны любому сценарию текущего документа. Такие переменные называются глобальными, в отличие от локальных переменных, определенных в теле функции.

ВЫРАЖЕНИЯ



Выражения строятся из литералов, переменных, знаков операций, скобок. В

результате вычисления выражения получается единственное значение, которое может быть либо числом (целым или вещественным), строкой, либо логическим значением. Используемые в выражении переменные должны быть инициализированы. Если при вычислении выражения встречается неопределенная или неинициализированная переменная, то фиксируется ошибка. В JavaScript определен литерал **`null`** для обозначения неопределенного значения. Если переменной присвоено значение **`null`**, то она считается инициализированной.

ПРИМЕР 1. ВЫЧИСЛЕНИЕ НАЛОГА

Создадим документ, в котором после ввода некоторой суммы вычисляется сумма налога с нее (13%). На странице первое текстовое поле используется для ввода дохода, по щелчку по кнопке **Вычислить** в поле **налог** помещается вычисленная сумма. Вид документа приведен на рисунке 1.

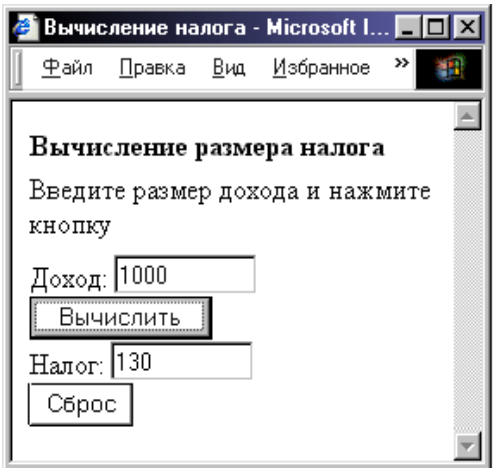


Рисунок 1.

ОБРАБОТЧИКИ СОБЫТИЙ

Интерактивные документы создаются с помощью форм. Форма на рисунке 1. содержит четыре элемента: два текстовых поля для ввода значения и для результата и две кнопки. При щелчке мышью по кнопке **Вычислить** мы хотим получить значение в поле **налог**.

Действие пользователя (например, щелчок кнопкой мыши) вызывает событие, которое производится, в основном, с элементами форм HTML. Обычно перехват и обработка события задается в параметрах элементов форм. Имя параметра обработки события начинается с приставки **`on`**, за которой следует имя самого события. Например, параметр обработки события **`Click`** будет выглядеть **`onClick`**.

Значением параметра обработки события могут быть операторы языка JavaScript. HTML-код документа, решающий задачу, представлен в листинге 1.

Рассмотрим подробнее значение параметра обработки события, представляющего собой в данном случае оператор присваивания

При интерпретации кода HTML браузером создаются объекты JavaScript. Взаимосвязь объектов между собой представляет иерархическую структуру. На самом верхнем уровне иерархии находится объект **`window`**, представляющий окно браузера. Объект **`window`** является предком или родителем всех остальных объектов. Каждая страница, кроме объекта **`window`**, имеет объект **`document`**. Свойства объек-

Листинг 1. Вычисление налога

```
<HTML>
<HEAD>
<TITLE>Вычисление налога</TITLE>
</HEAD>
<BODY>
<h4>Вычисление размера налога</h4>
<FORM name="form1">
<p>Введите размер дохода и нажмите кнопку</p>
Доход: <INPUT type="text" name="num" size=10><br>
<INPUT type="button" value=Вычислить
onClick="document.form1.res.value =0.13*document.form1.num.value"><br>
Налог: <INPUT type="text" name="res" size=10><br>
<INPUT type="reset">
</FORM>
</BODY>
</HTML>
```

та **document** определяются содержимым самого документа: цвет фона, цвет шрифта и т.д. В рассматриваемом примере на странице расположена форма. Форма (**form**) является потомком объекта **document**, а все элементы формы являются потомками объекта **form**. Ссылка на объект осуществляется по имени, заданному параметром **name** тега **html**. Для получения значения введенного в первом поле ввода формы требуется выполнить **document.form1.num.value**. При ссылке на формы и их элементы можно не указывать объект **document**, поэтому получить значение первого поля ввода можно и следующим образом: **form1.num.value**. Для того, чтобы поместить вычисленное значение во второе текстовое поле достаточно изменить значение **document.form1.res.value**.

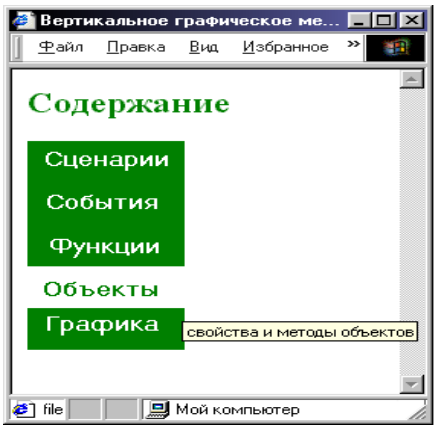


Рисунок 2

ПРИМЕР 2. ВЕРТИКАЛЬНОЕ ГРАФИЧЕСКОЕ МЕНЮ

Создадим документ, реализующий вертикальное графическое меню. При наведении курсора мыши над пунктом меню должна меняться цветовая палитра, соответствующая выделенному пункту меню (рисунок 2).

Каждому пункту меню соответствует два изображения: первое изображение, когда пункт меню не выбран, второе – при выбранном пункте меню цветовая палитра рисунка изменена. Графические изображения, соответствующие ситуации, когда пункты меню не выбраны, хранятся в файлах с именами pch1.gif, pch2.gif, pch3.gif, pch4.gif, pch5.gif. Соответствующие им графические изображения с измененной палитрой хранятся в файлах с именами wpch1.gif, wpch2.gif, wpch3.gif, wpch4.gif, wpch5.

Листинг 2. Вертикальное графическое меню

```
<html>
<head>
<title>Вертикальное графическое меню</title>
</head>
<body>
<h2><font color=green>Содержание</font></h2>
<a href="tch1.htm"
  onmouseover="document.pm1.src='wpch1.gif'"
  onmouseout="document.pm1.src='pch1.gif'">
<img src="pch1.gif" name="pm1" alt="Создание сценариев" border="0"
 width="103" height="35"></a><br>
...
<a href="tch5.htm"
  onmouseover="document.pm5.src='wpch5.gif'"
  onmouseout="document.pm5.src='pch5.gif'">
<img src="pch5.gif" name="pm5" alt="Использование графики"
     border="0" width="103" height="35"></a> <br>
</body>
</html>
```

При перемещении курсора над связью возникает событие **onmouseover**. В этом случае при решении задачи требуется изменить графическое изображение, соответствующее выбранному пункту меню, что осуществляется в результате выполнения **onmouseover="document.pm1.src='wpch1.gif '"**. Если курсор мыши перемещается из области связи, то возникает событие **onmouseout**, в результате обработки которого пункт меню должен принять первоначальный вид. Это достигается оператором присваивания **onmouseout="document.pm1.src='pch1.gif'"**. HTML-код документа, реализующего графическое вертикальное меню, может иметь следующий вид.

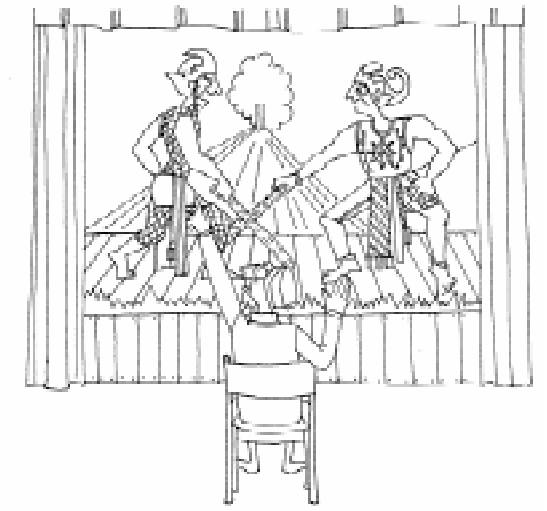
В предыдущих примерах мы использовали лишь оператор присваивания как значение параметра обработки события. В более сложных случаях требуется выполнить несколько действий и написать сценарий.

СЦЕНАРИИ В HTML-ДОКУМЕНТЕ

Сценарии, написанные на языке JavaScript, могут располагаться непосредственно в HTML-документе между тегами **<script>** и **</script>**.

Одним из параметров тега **<script>** является параметр **language**, он определяет используемый язык сценариев. Для языка JavaScript значение параметра равно «JavaScript». Если используется язык сценариев VBScript, то значение параметра должно быть равным «VBScript». В случае использования языка JavaScript параметр **language** можно опускать, так как этот язык используется браузером по умолчанию.

Обычно браузеры, не поддерживающие какие-либо теги HTML, их просто игнорируют. Попытка браузера проанализировать содержимое неподдержанных тегов может привести к неверному отображению страницы. Чтобы избежать такой ситуации, рекомендуется помещать операторы языка JavaScript в теги ком-



ментария <!---...--->. Для правильной работы интерпретатора перед закрывающим тегом комментария --> следует поставить символы //.

Итак, для размещения сценария в HTML-документе следует написать следующее:

```
<script language= "JavaScript">
<!--
операторы языка JavaScript
//-->
</script>
```

Документ может содержать несколько тегов **<script>**. Все они последовательно обрабатываются интерпретатором JavaScript.

Тег **<noscript>** определяет HTML-код, отображаемый на экране в случае, если JavaScript не поддерживается браузером или поддержка отключена. Этот тег следует после кода, заключенного в теги **<script>** и **</script>**. Если поддержка включена, то тег **<noscript>** игнорируется. В дальнейших примерах будем считать, что поддержка JavaScript включена.

ФУНКЦИИ, ОПИСАНИЕ И ИСПОЛЬЗОВАНИЕ

При создании программы разумно выделить в ней логически независимые части (так называемые подпрограммы). Каждую часть при необходимости можно разбить на отдельные подпрограммы и т.д. Разбиение программы на подпрограммы облегчает процесс отладки, так как позволяет отлаживать каждую подпрограмму отдельно. Можно распределить работу по созданию сложной программы между отдельными программистами. Некоторые подпрограммы можно использовать для решения разных задач. Подпрограммы один раз описываются, а затем используются произвольное число раз.

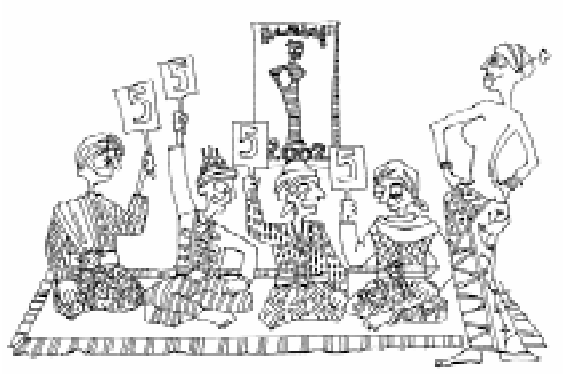
Во многих языках программирования понятие подпрограммы реализуется с помощью конструкций процедур, функций, модулей и т.п.

Основным элементом языка JavaScript является функция. Описание функции имеет вид: **function F (V) {S}**, где F – идентификатор функции, задающий имя, по которому можно обращаться к функции, V – список параметров функции, разделяемых запятыми, S – тело функции, в нем задаются действия, которые нужно выполнить, чтобы получить результат. Необязательный параметр **return e** определяет возвращаемое функцией значение.

Описание функции не может быть вложено в описание другой функции. Параметры функции внутри ее тела играют ту же роль, что и обычные переменные, но начальные значения этим параметрам задаются при обращении к функции. Если описание функции имеет вид **function F(**v_1**,**v_2**,...,**v_n**){S}**, то вызов функции должен иметь вид: **F(**e_1**,**e_2**,...,**e_n**)**, где $e_1,e_2,...,e_n$ – выражения, задающие фактические значения параметров. Параметры $v_1,v_2,...,v_n$, указанные в описании функции, называются формальными параметрами, чтобы подчеркнуть тот факт, что они получают смысл только после задания в вызове функции фактических параметров $e_1,e_2,...,e_n$, с которыми функция затем и работает. Если в функции параметры отсутствуют, то есть описание функции имеет вид **function F{S}**, то наличие скобок в операторе вызова функции обязательно, то есть вызов функции в этом случае должен иметь вид: **F()**.

Обычно все определения и функции задаются в разделе <HEAD> документа. Это обеспечивает интерпретацию и сохранение в памяти всех функций при загрузке документа в браузер.

ПРИМЕР 3. СУММА БАЛЛОВ



Создадим документ, в котором располагается форма для ввода оценок, при нажатии на кнопку вычисляется сумма баллов. Вид документа представлен на рисунке 3.

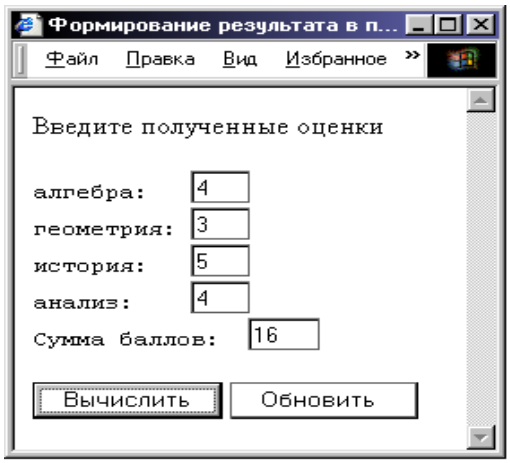


Рисунок 3

Функция **val(obj)** имеет один параметр – имя формы. Введенные в форме значения сохраняются в локальных переменных **a1,a2,a3,a4**. Локальная переменная s служит для определения суммы, вычисленное значение записывается в соответствующее поле формы. Стандартная функция **Number** преобразует строковое значение в число (см. листинг 3).

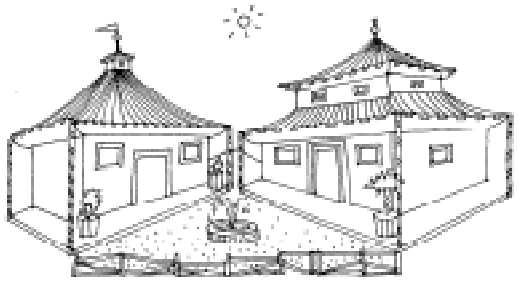
Листинг 3. Использование сценария с функцией

```
<HTML>
<HEAD>
<TITLE>Формирование результата в поле формы</TITLE>
<SCRIPT LANGUAGE="JavaScript">
<!--//
function val (obj)
 {var a1=Number(obj.num1.value);
  var a2=Number(obj.num2.value);
  var a3=Number(obj.num3.value);
  var a4=Number(obj.num4.value);
  var s = a1+a2+a3+a4
  obj.res.value = s
 }
//-->
</SCRIPT>
</HEAD>
<BODY>
Введите полученные оценки<BR>
<FORM name="form1">
<pre>
алгебра: <INPUT type="text" size=3 name="num1">
геометрия: <INPUT type="text" size=3 name="num2">
история: <INPUT type="text" size=3 name="num3">
анализ: <INPUT type="text" size=3 name="num4">
Сумма баллов: <INPUT type="text" size=4 name="res">
</pre>
<INPUT type="button" value=Вычислить onClick="val(form1)">
<INPUT type="reset" value="Обновить ">
</FORM></BODY></HTML>
```

ОБЪЕКТ MATH И ЕГО СВОЙСТВА

В языке JavaScript определены некоторые стандартные объекты и функции, пользоваться которыми можно без предварительного описания. Одним из стандартных объектов является объект **Math**. В свойствах объекта **Math** хранятся основные математические константы, его методы можно использовать для вызова основных математических функций. С помощью методов объекта **Math** можно вычислить абсолютное значение, натуральный логарифм, наименьшее и наибольшее значение функции двух аргументов, вычислить степень вещественного числа, вычислить квадратный корень.

ПРИМЕР 4. ПЛОЩАДЬ ТРЕУГОЛЬНИКА ПО ФОРМУЛЕ ГЕРОНА



Напишем сценарий вычисления площади и периметра треугольника, заданного длинами сторон. Документ представлен на рисунке 4.

Для того чтобы вычислить площадь треугольника по длинам сторон, используется формула Герона. В формуле Герона требуется применить функцию извлечения квадратного корня, для этого воспользуемся методом **sqrt** объекта **Math**: **Math.sqrt** (см. листинг 4).

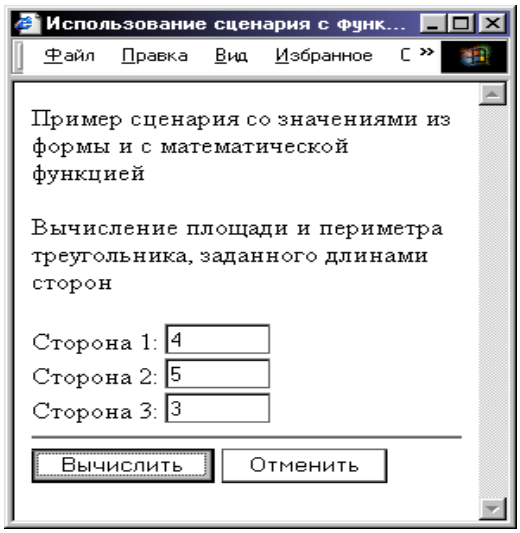
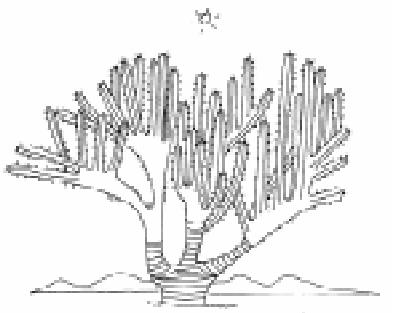


Рисунок 4

Вычисленные в функции значения помещаются в документ с помощью метода **write**.

ОРГАНИЗАЦИЯ ВЕТВЛЕНИЙ В ПРОГРАММАХ



При составлении программы часто необходимо выполнение различных действий, в зависимости от результатов проверки некоторых условий. Для организации ветвлений можно воспользоваться условным оператором, который имеет вид

```
if B {S1}
else {S2}
```

где **B** – выражение логического типа, **S1** и **S2** – операторы. Выполнение условного оператора осуществляется следующим образом: вычисляется значение выражения **B**, если оно истинно, то выполняются операторы **S1**, если ложно – операторы **S2**. Если последовательность операторов **S1** или **S2** состоит лишь из одного опера-

Листинг 4. Объект Math

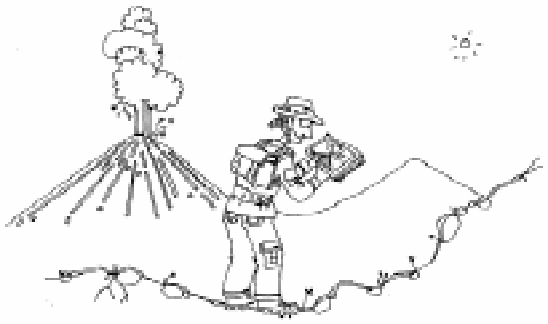
```
<html>
<head>
<title>Объект Math</title>
<script language="JavaScript">
<!--//
function care (obj)
 {var a= Number(obj.st1.value)
  var b= Number(obj.st2.value)
  var c= Number(obj.st3.value)
  var s; p=a+b+c;
document.writeln ("Периметр треугольника равен",p,"<BR>");
  p=p/2;
  s=Math.sqrt(p*(p-a)*(p-b)*(p-c));
document.write ("Площадь треугольника равна",s);
}
//–>
</script>
</head>
<body>
<p>Пример сценария с математической функцией</p>
<p>Вычисление площади и периметра треугольника</p>
<form name="form1">
Сторона 1: <input type="text" size=7 name="st1"><hr>
Сторона 2: <input type="text" size=7 name="st2"><hr>
Сторона 3: <input type="text" size=7 name="st3"><hr>
<input        type="button"        value=Вычислить
onClick="care(form1)"><hr>
 <input type="reset" value=Отменить
</form>
</body>
</html>
```

тора, то фигурные скобки можно опустить. Возможна сокращенная форма условного оператора: **if B {S}**, где **B** – выражение логического типа, **S** – последовательность операторов. Выполнение краткого условного оператора осуществляется так: вычисляется значение выражения **B**, если оно истинно, то выполняются операторы **S**.

ВИЗУАЛЬНЫЕ ЭФФЕКТЫ

С помощью сценариев JavaScript можно создавать простые визуальные эффекты: приближение и удаление изображения, перемещение изображения по странице и многое другое. Начнем с самых простых примеров, использующих только те средства, которые нам известны к настоящему времени.

ПРИМЕР 5. УДАЛЯЮЩЕЕСЯ ИЗОБРАЖЕНИЕ



Напишем сценарий, при выполнении которого создается эффект удаления изображения от зрителя, то есть изображение начинает уменьшаться в размерах.

Напомним, что свойство объекта **document** определяется содержимым самого документа. Если в документе содержится изображение, то доступ к объекту **image** можно получить с помощью значения Name тега IMG. Объект **document** имеет свойство **images**, которое содержит ссылки на все изображения, расположенные в документе. Ссылки перенумерованы, начиная с нуля. Доступ к первому изображению на странице можно получить с помощью конструкции **document.images[0]**, ко второму – **document.images[1]** и т.д.

При решении этой задачи будем использовать свойство **height** объекта **image**. При каждом вызове функции **succpict()** изменяется размер выводимого изображения и тем самым достигается эффект удаления от зрителя. Функция **setTimeout("succpict()", 500)** производит повторный вызов функции **succpict()** через каждые полсекунды. Когда размер изображения изменится до заданного, движение прекратится. В начальный момент движение имеет вид, как на рисунке 5.

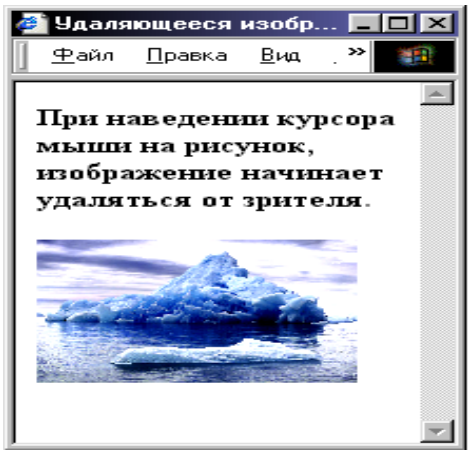


Рисунок 5. Эффект удаления от зрителя

Для каждого изображения параметры уменьшения изображения и время обновления следует подбирать индивидуально. Сценарий описан в листинге 5.

Нетрудно решить и обратную задачу, то есть написать сценарий, при работе которого при наведении курсора мыши на изображение оно начинает приближаться к зрителю. Используя сценарии из последних примеров, нетрудно написать сце-

Листинг 5. Удаляющееся изображение

```
<HTML>
<HEAD>
<TITLE>Удаляющееся изображение</TITLE>
<SCRIPT LANGUAGE="JavaScript">
<!--//
var h = 200 //начальная высота изображения
// эффект уменьшения изображения
var d= document
var h= 200
function succpict ()
 {if (h > 100)
     { h=h-10
       d.images[0].height= h
       setTimeout("succpict()", 500)
    }
 }
//-->
</SCRIPT>
</HEAD>
<BODY>
<H4>При наведении курсора мыши на рисунок изображение начинает удаляться
от зрителя.</H4>
<img src="msm.jpg" name=mypict height=200 onMouseOver="succpict()">
</BODY>
</HTML>
```

нарий создания пульсирующего изображения: изображение сначала увеличивается до определенного размера, затем уменьшается до заданного размера, затем опять увеличивается и т.д.

ПРИМЕР 6. ЧЕРЕДОВАНИЕ ИЗОБРАЖЕНИЙ С ПРЕДВАРИТЕЛЬНОЙ ЗАГРУЗКОЙ

Напишем сценарий, при выполнении которого после загрузки страницы реализован визуальный эффект чередования изображений: на странице через равные промежутки времени появляются различные изображения.

Мы умеем решать такие задачи, не используя метод предварительной загрузки изображений. Чтобы предварительно загрузить изображения, создадим массив **`imgslide: imgslide = new Array()`**. Каждый элемент массива представляет собой объект **`Image`**, который создается с помощью конструктора: **`new Image()`**. Первый элемент массива определяется так: **`imgslide[0] = new Image()`**. Свойство **`src`** только что созданного объекта задается в результате выполнения оператора присваивания:

```
imgslide[0].src = "sl1.gif"
```

Аналогично формируются и другие элементы массива. При работе сценария происходит обращение к элементам массива, и дополнительная загрузка изображений уже не потребуется. Листинг 6 – HTML-код документа, содержащего сценарий решения задачи.

Листинг 6. Чередование изображений с предварительной загрузкой

```
<html>
<head>
<title>Чередование изображений с предварительной загрузкой</title>
</head>
<script language="JavaScript">
// предварительная загрузка изображений
numimg=0
imgslide=new Array()
imgslide[0]=new Image()
imgslide[1]=new Image()
imgslide[2]=new Image()
imgslide[3]=new Image()
imgslide[0].src="sl1.gif"
imgslide[1].src="sl2.gif"
imgslide[2].src="sl3.gif"
imgslide[3].src="sl4.gif"
// чередование изображений
function demoslides()
{ document.images[0].src=imgslide[numimg].src
  numimg++
  if(numimg==4)
     numimg=0;
  setTimeout("demoslides()", 1000)
}
</script>
<body bgcolor="#FFFFFF" onLoad="demoslides()">
<p><img src="slidimg1.gif" width="100" height="100"></p>
</body>
</html>
```

Листинг 7. Движение изображения слева направо

```
<HTML>
<HEAD>
<TITLE>Простая анимация</TITLE>
<SCRIPT LANGUAGE="JavaScript">
<!--//
var stop=100
var n= 1 // номер кадра, куда вставляется изображение
var d= document
// предварительная загрузка изображений
imgempty = new Image()
imgempty.src = "p0.gif"
imganim = new Image()
imganim.src = "p1.gif"
// функция, осуществляющая анимацию
function anim1 ()
{var len= d.images.length
 if (n != stop)
 {n = n+1
 if (n > len)
   {n=1;
    d.images[n-1].src=imganim.src
    d.images[len-1].src=imgempty.src
   }
 else
  {d.images[n-2].src=imgempty.src
   d.images[n-1].src= imganim.src
  }
 setTimeout("anim1()",500)
 }
}
// функция для задания начальной позиции
function st ()
{ d.images[0].src=imganim.src
  for (var i=1; i <= d.images.length-1; i++)
    d.images[i].src= imgempty.src
}
//-->
</SCRIPT>
</HEAD>
<BODY>
<h4 align=center>Простая анимация <h4>
<hr>
<img src=p1.gif>
<img src=p0.gif>
<img src=p0.gif>
<img src=p0.gif>
<img src=p0.gif>
<img src=p0.gif>
<hr>
<FORM name="form1">
 <input type="button" value=Начало onClick= "n = stop; st()">
 <input type="button" value=Запустить onClick= "n=1; anim1()">
</FORM>
</BODY>
</HTML>
```

В предыдущей задаче разные изображения появлялись на одном и том же месте на странице. Немного усложним задачу.

ПРИМЕР 7. ДВИЖЕНИЕ ИЗОБРАЖЕНИЯ СЛЕВА НАПРАВО

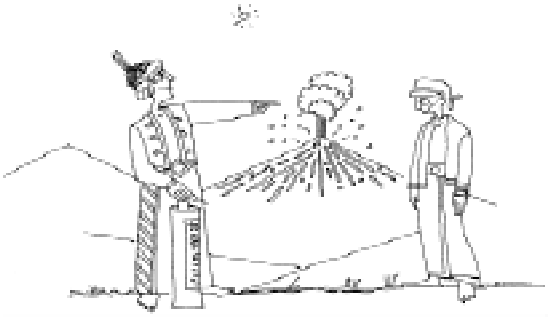
Напишем сценарий реализации простой анимации: перемещение изображения слева направо (см. листинг 7).

Для анимационного изображения будем использовать всего два рисунка. Первый рисунок – то изображение, которое перемещается. Второй рисунок должен соответствовать «пустому» изображению. Для этого можно создать одноцветный рисунок в формате GIF такого же размера, как и движущееся изображение. Для определенности будем считать, что имеется m кадров. Пусть n номер кадра, в который помещается изображение. В следующий момент времени в кадр с номером n следует поместить «пустой» рисунок, в кадр с номером $n+1$ изображение. Тем самым как бы переместили изображение из кадра с номером $n+1$ и создали эффект движения слева направо. Когда изображение появится в последнем кадре, его следует переместить в первый кадр. Выражение `document.images.length` определяет количество кадров. Сразу после загрузки документа изображение расположено в самом левом кадре. В остальных кадрах располагается «пустое» изображение.

При выполнении сценария перемещение изображения происходит небольшими скачками. Эти скачки можно сделать плавными, если увеличить число кадров на странице. Другой способ – использование набора картинок, которые отражают при движении промежуточное состояние, соответствующее моменту, когда изображение переходит из одного кадра в другой. Чем больше кадров и чем точнее набор рисунков, тем плавней будет казаться движение по странице.

Следует помнить, что можно только менять уже имеющиеся на странице рисунки, но нельзя добавить новый рисунок или удалить существующий.

Задание 1



1. Участникам тестирования было предложено 6 задач. За решение каждой из задач ставились баллы: 0, 0.3, 0.6 или 1. Всех участников, проходивших тестирование, разбили на четыре категории, в зависимости от результатов. В первую категорию включили участников, все решения которых оценены максимальным баллом 1. Во вторую категорию включили участников, все задачи которых оценивались не ниже, чем 0.6, но обязательно была хоть одна задача, решение которой оценено баллом 1. В третью категорию включили участников, у которых все задачи были оценены баллом 0.6. Остальных участников отнесли к четвертой категории.

• Создайте анкету конкретного участника тестирования. В анкете должна быть указана фамилия, номер школы, оценки за решения задач. При обработке анкеты для участника требуется определить сумму набранных баллов и категорию, в которую зачислен участник.

• Подготовьте четыре рисунка, на каждом из которых указан номер катего-

рии. Напишите сценарий обработки анкеты, при выполнении которого в документе появляется рисунок с номером категории тестируемого.

2. По результатам сдачи зачетов всех студентов разделили на несколько категорий. Те студенты, которые получили все зачеты, относятся к категории «отличники». Если у студента не получен один зачет, то его относят к категории «успевающий». Если студент не получил двух зачетов, то он не допускается к сдаче экзаменов. Назовем такую категорию студентов «не допущены к сессии». Если же студент не получил трех зачетов, то его представляют к отчислению (категория «представленные к отчислению»).

• Создайте анкету студента, в которой указана фамилия, номер группы и результаты сдачи зачетов по 7 предметам. При обработке анкеты требуется определить количество сданных зачетов и категорию.

• Подготовьте четыре рисунка, каждый из которых соответствует одной из категорий. Напишите сценарий обработки анкеты, при выполнении которого в документе появляется рисунок, соответствующий категории, в которую попадает студент.

3. Участникам олимпиады было предложено 6 задач. За решение каждой из задач ставились баллы: 0, 10, 20 или 30. Первое место присуждалось участникам, все задачи которых были оценены максимальным числом баллов. Второе место присуждалось участникам, решения всех задач которых были оценены не ниже, чем на 20 баллов, причем, по крайней мере, одна задача была оценена высшим баллом. Третье место занимали участники, решения всех задач которых оценивалось 20 баллами. Призовые места другим участникам не присуждались.

• Создайте анкету участника олимпиады. При обработке анкеты требуется определить сумму набранных за решения задач баллов и место, на которое участник олимпиады претендует.

• Подготовьте три рисунка, на каждом из которых указано место (1,2,3). Напишите сценарий обработки анкеты, при выполнении которого в документе появляется рисунок с номером присужденного места.

4. Заданы три целых значения. Напишите сценарий, при выполнении которого определяется и выводится в документ информация о том, можно ли построить треугольник с длинами сторон, равными заданным значениям. Если треугольник построить можно, то требуется определить его вид: прямоугольный, остроугольный, тупоугольный. При определении вида треугольника в документе должно появиться соответствующее изображение.

5. Три точки на плоскости задаются координатами своих вершин. Напишите сценарий, при работе которого определяется, можно ли построить треугольник с вершинами в заданных точках, и выводится соответствующая информация. Если треугольник построить можно, то требуется определить его вид: равносторонний, равнобедренный, разносторонний. При определении вида треугольника в документе должно появиться изображение треугольника соответствующего вида.

6. Два треугольника на плоскости задаются координатами своих вершин. Напишите сценарий, в результате работы которого определяется, являются ли эти два треугольника подобными. В каждом из вариантов решения должно быть выведено соответствующее графическое изображение.

7. Отрезок на плоскости задается координатами концов отрезка. Напишите сценарий, который для точки на плоскости, заданной своими координатами, определяет:

• лежит ли заданная точка на отрезке;

• точка не лежит на отрезке, но лежит на той же прямой, что и отрезок;

• точка не лежит на отрезке и не принадлежит прямой, содержащей отрезок.

Для каждого из вариантов ответа требуется вывести изображение, соответствующее ситуации.

8. Два круга на плоскости задаются координатами центра и радиусом. Напишите сценарий, в результате работы которого определяется положение кругов относительно друг друга. Варианты могут быть следующие:

- один круг лежит внутри другого;
- круги не имеют общих точек;
- круги пересекаются;
- круги имеют одну точку касания.

В каждом из вариантов ответа должно быть выведено соответствующее графическое изображение.

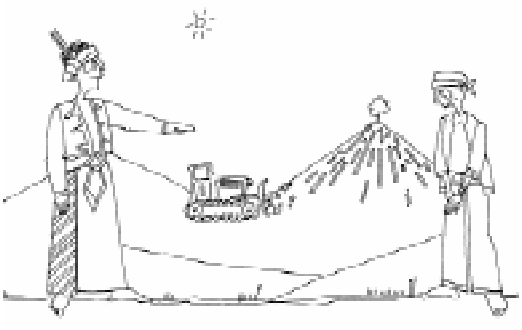
9. Круг на плоскости задается координатами центра и радиусом. Квадрат задается координатой левой верхней вершины и длиной стороны. Напишите сценарий, определяющий взаимное положение круга и квадрата. Возможны следующие варианты:

- круг и квадрат не имеют общих точек;
- круг лежит внутри квадрата;
- квадрат лежит внутри круга;
- круг и квадрат пересекаются.

В каждом из вариантов ответа должно быть выведено соответствующее графическое изображение.

10. Треугольник на плоскости задается координатами своих вершин. Напишите сценарий, в результате работы которого определяется, лежит ли заданная точка на плоскости внутри треугольника, вне него или на одной из сторон. При каждом из вариантов ответа должно быть выведено соответствующее графическое изображение.

Задание 2



В следующих заданиях требуется создать иллюзию движения. Анимация основана на смене кадров, каждый из которых соответствует очередному положению движущегося объекта.

11. Напишите сценарий создания спортивного мультфильма о подтягивании на перекладине.

12. Напишите сценарий спортивного фильма о поднятии штанги.

13. Напишите сценарий, изображающий человека, марширующего на месте.

14. Напишите сценарий изображающий парашютиста, выпрыгивающего из самолета и приземляющегося с раскрытым парашютом.

15. Напишите сценарий, демонстрирующий человека, выполняющего упражнение, например, приседание.

16. Напишите сценарий, изображающий игру музыканта на гитаре.

17. Напишите сценарий, демонстрирующий работу светофора.

18. Напишите сценарий спортивного фильма о прыжках в длину.

19. Напишите сценарий спортивного фильма о метании диска.

20. Напишите сценарий, иллюстрирующий работу маятника.

Дмитриева Марина Валерьевна, доцент кафедры информатики математико-механического факультета СПб государственного университета.