3.7. СОБЫТИЯ И ФУНКЦИИ

Содержание

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. Источники

1. Справочник по языку Javascript: http://javascript.ru/manual
2. Библиотека (`JSChart`) для построения графиков и диаграмм: http://www.jscharts.com/.