ОГЛАВЛЕНИЕ

1. ВВЕДЕНИЕ

Книг по программированию на языке JavaScript написано очень много — от тонких брошюр до солидных фолиантов. Автору по роду своей работы пришлось ознакомиться со многими из таких изданий. Все они по-своему хороши и неплохо решают задачу обучения программированию. К сожалению, подавляющее большинство из них рассматривают основы языка и его возможности, но не учат непосредственно писать сценарии для тех или иных практических нужд.

Автор по мере сил на протяжении нескольких лет пытался восполнить этот пробел. Было написано несколько книг, рассказывающих о простейших сценариях, о работе с изображениями, о создании визуальных редакторов и CMS, о приемах креативного программирования. Все эти книги вышли в издательстве «Лань».

Теперь пришло время коснуться такой темы, как создание визуальных эффектов на веб-страницах. Эти эффекты можно внедрить в оформление сайта, в рекламные блоки, в слайдеры, в игры.

1.1. О чем эта книга

В данной книге разобраны приемы создания на HTML-страницах различных визуальных эффектов с помощью языка программирования JavaScript: от простейших манипуляций с фигурами и линиями до создания сложных галерей изображений.

Ознакомившись с содержанием книги, читатели научатся множеству разных приемов разработки:

– изменению размеров и форм элементов;
– имитации движения;
– «сборке» элементов из линий;
– конструированию динамичных графиков;
– рисованию на экране монитора;
– проектированию необычных слайдеров;
– написанию простейших игр.

Кроме того, автор напомнит читателям о множестве свойств и методов, которые могут оказаться полезными при разработке различных визуальных эффектов.

Совсем не обязательно применять изученные методы только в тех областях, что разобраны в книге. Такие алгоритмы могут использоваться при решении совершенно иных задач. Это как в математике: изучили формулу, решили несколько примеров вместе с преподавателем. Тем самым разобрались с алгоритмом, который теперь можно использовать при решении других задач с применением этой формулы.

1.2. Особенности изложения материала

В книге одиннадцать глав.

Первая — «Введение». Вы сейчас читаете ее.

Во второй мы вспомним ряд свойств, изменение которых у того или иного элемента создает различные визуальные эффекты. Кроме того, нам предстоит обсудить перечень событий, которые могут запускать сценарии визуальных трансформаций.

Третья глава рассказывает о том, как простыми способами можно изменять фигуры: например, их размер, форму, цвет, непрозрачность. Также она касается приемов вращения объектов.

Из главы 4 вы узнаете о других способах создания визуальных эффектов, в том числе в тексте. А еще мы рассмотрим несколько примеров влияния перемещения мыши на характеристики элементов веб-страницы.

Глава 5 посвящена методам сборки фигур из линий, а также имитации рисования фигур.

В шестой мы подробно разберем различные варианты движения элементов в окне браузера.

В седьмой рассмотрим, как можно создавать динамические графики.

Тема восьмой главы — рисование в окне браузера с помощью мыши. Фактически мы создадим три разных по сложности графических редактора.

Девятая глава рассказывает о применении изученных ранее методов программирования при создании различных слайдеров.

В десятой главе мы используем полученные навыки при написании двух простейших игр.

И в самой короткой главе 11 подведем итоги нашей работы.

Книгу сопровождает ZIP-архив со всеми рассмотренными программами. Его можно скачать по ссылке или QR-коду, указанным на странице 2.

Работать с архивом очень просто. Если описание какой-то программы начинается с указания «Файл», значит, ее страница находится в той или иной папке архива. Понять, где найти программу, легко. Вот пример:

> Файл **3/3.4.1.html**

Здесь до слеша указано название папки, где размещается необходимый вам файл. Цифра в имени папки соответствует номеру главы, к которой эта папка относится. Дальше идет имя файла — **3.4.1.html**. Первая цифра в его имени — номер главы, вторая — номер раздела, третья — порядковый номер программы, под которым она описана в разделе.

Глядя на пример выше, понятно, что перед нами сценарий из третьей главы четвертого раздела. В данном разделе этот сценарий разбирается первым.

Чтобы посмотреть код, откройте страницу в каком-либо из специализированных редакторов. Хотите посмотреть сценарий в действии? Просто запустите ту или иную страницу в браузере.

1.3. Оформление сценариев

Сценарии имеют различное типографское оформление в соответствии с их размещением в тексте.

Если код в книге выделен в отдельный блок, то он оформлен моноширинным шрифтом, например, так:

```
addEventListener("load", ()=>
{
addEventListener("click", ()=>
   {
   let squ=document.getElementById("square").style;
   squ.transition="width 2s, height 2s, top 2s";
   squ.width="400px";
   squ.height="400px";
   squ.top="50px";
   });
});
```

Если фрагменты сценария внедрены непосредственно в текст, то в этом случае части кода выделены полужирным шрифтом, например, так:

если условие **h-d>3** истинно, то

Обратите внимание: в некоторых блоках программ сделан перенос части кода на вторую и даже на третью или четвертую строку (из-за недостатка ширины страницы в книге). В реальном сценарии код записывается одной строкой. Запомните правило: все переносы строк кода существуют только в их типографском воспроизведении. Если вы в дальнейшем столкнетесь с подобной ситуацией, учитывайте данный аспект.

Еще один момент. Обычно таблицы стилей какой-либо страницы принято записывать следующим образом:

```
#spb {
   border: 5px solid #CC0000;
   margin-top: 100px;
   opacity: 1;
}
```

Однако подобный код занимает в книге слишком много места. Поэтому автор применил сокращенную форму записи стилей:

```
#spb {border: 5px solid #CC0000;
  margin-top: 100px; opacity: 1;}
```

Как видите, экономия налицо: в этом случае вместо пяти строк на запись потрачено только две.

В описании программ нередко имеет смысл не приводить весь код, который относится к разбираемому фрагменту, а показывать только ту его часть, что важна в контексте данных объяснений. Поэтому в таких случаях автор вводит сокращение в виде трех точек:

```
...
```

Например, сокращенный вариант для предыдущего примера с настройками стилей может выглядеть так:

```
#spb {... opacity: 1;}
```

Тем самым мы показываем, что на данном этапе описания нам важно обратить внимание исключительно на непрозрачность элемента.

И последнее. Все принципиально важные фрагменты текста или кода в отдельных блоках, на которые надо обратить особое внимание, выделены полужирным шрифтом.

2. ПОЛЕЗНЫЕ СВОЙСТВА И СОБЫТИЯ

Внешний вид любого элемента страницы определяется множеством различных свойств и их значений. В следующем разделе перечислим, какие из них наиболее важны при создании визуальных эффектов. В разделе 2.2 мы поговорим о событиях, которые наиболее часто используются программистами для запуска анимированных эффектов. В последнем разделе этой главы вспомним о таймерах, с помощью которых станем с определенной закономерностью менять характеристики объектов на веб-страницах.

2.1. Изменение свойств

Одно из важнейших CSS-свойств, значения которых мы научимся менять программными методами, — размер элемента на странице. Размер определяется двумя параметрами.

width — задает ширину элемента. Если значение указано в пикселях, то ширина будет фиксированной. Если значение указано в процентах, то ширина будет меняться в зависимости от ширины блока, содержащего объект.

height — задает высоту элемента. Если значение указано в пикселях, то высота будет фиксированной. Если значение указано в процентах, то высота будет меняться в зависимости от высоты блока, содержащего объект.

Теперь о рамках изображений или фигур. Нас заинтересуют четыре параметра: толщина, цвет, тип линии и скругление.

Первые три параметра можно задать, используя сокращенную запись свойств рамок — **border**. Например:

```
img {border: 1px solid #000000;}
```

или

```
img {border: 2px dashed #0000CC;}
```

Для скругления рамок используют свойство **border-radius**:

```
img {border-radius: 20px;}
```

Если применить это свойство, не задавая объекту рамку, то его углы все равно будут скруглены.

Скругление можно задавать для каждого угла отдельно:

```
img {border-radius: 20px 60px 100px 140px;}
```

Мы рассмотрели самые простые варианты создания рамок — именно такие использованы в некоторых примерах из книги. Но способов оформления рамок гораздо больше. Например, можно сделать так, что все четыре стороны рамки будут разного цвета. Или разного типа. Или разной толщины. И не только это — вы можете получить еще очень много нестандартных визуальных эффектов. Если данная тема интересна вам, рекомендуется проштудировать какой-нибудь справочник по CSS.

Для создания теней по краям элементов используется свойство **box-shadow**. Оно определяет параметры тени. Свойство имеет четыре числовых значения, которые записываются в следующем порядке:

– смещение по горизонтали;

– смещение по вертикали;

– расстояние размытия;

– расстояние распространения тени.

Кроме того, есть еще один параметр — цвет тени. Цветовое значение записывается либо первым пунктом свойства, либо последним.

Экспериментируя со смещениями, расстояниями и цветом, можно получить самые разнообразные варианты теней.

Для создания теней к тексту используют свойство **text-shadow**. Свойство описывает смещение тени по осям *X* и *Y*, радиус размытия и цвет тени.

По умолчанию, если иное не задано в таблице стилей, все элементы имеют полную непрозрачность. Когда в процессе манипулирования компонентами страницы надо изменить непрозрачность какого-либо элемента, используют свойство **opacity**. Его значение задается вещественным числом в интервале от **0** (объект полностью прозрачен) до **1** (объект полностью непрозрачен).

Элемент на странице можно расположить одним из трех способов:

– в потоке компонентов документа (это когда элементы в разметке страницы просто следуют один за другим и также последовательно отображаются на странице);

– позиционировать как самостоятельный элемент относительно границ документа;

– позиционировать относительно границ вмещающего контейнера, например, слоя div.

Первый способ — довольно редкий, так как при таком размещении трудно получить красивую страницу.

В остальных способах используют либо абсолютное позиционирование, либо относительное. Для этого устанавливают значение свойства **position**:

– **absolute** — при этом значении объект занимает строго фиксированное положение, привязанное к границам документа или вмещающего контейнера и не зависящее от положения остальных элементов;

– **fixed** — привязывает объект к границам окна браузера, при прокрутке страницы такой элемент остается на мониторе в одном и том же положении;

– **relative** — сначала высчитывается нормальное положение объекта в потоке элементов документа, а затем происходит его смещение на указанные величины — на экране клиент видит конечный результат.

Для абсолютного и относительного позиционирования используются четыре свойства:

– **left** — определяет смещение элемента вправо относительно левого края документа, вмещающего контейнера или исходного положения;

– **right** — определяет смещение элемента влево относительно правого края документа, вмещающего контейнера или исходного положения;

– **top** — определяет смещение элемента вниз относительно верхнего края документа, вмещающего контейнера или исходного положения;

– **bottom** — определяет смещение элемента вверх относительно нижнего края документа, вмещающего контейнера или исходного положения.

Смещение объекта по вертикали в линии других элементов задается свойством **vertical-align**. По умолчанию все элементы в строке позиционированы по базовой линии. При создании визуальных эффектов в качестве значений этого свойства можно указывать:

– ключевые значения, например

```
vertical-align: sub;
```

– числовые значения (в том числе отрицательные), например

```
vertical-align: -10px;
```

– значения в процентах, например

```
vertical-align: 30%;
```

Когда необходимо программными методами скрыть какой-то элемент или, наоборот, показать скрытый, используют свойство **visibility** с одним из двух значений:

– **visible** — элемент виден (значение по умолчанию);

– **hidden** — элемент не виден.

Если на странице несколько объектов и они частично или полностью перекрывают друг друга, необходимо установить порядок их следования, или иерархию. Для этого применяют свойство **z-index**, которое определяет расположение элементов по оси **z**, перпендикулярной плоскости отображения.

Чем выше порядковый номер **z-index**, тем выше находится элемент. Следует в таких случаях между значениями брать некоторый числовой интервал, например:

```
#img1 {z-index: 10;}
#img2 {z-index: 20;}
#img3 {z-index: 30;}
```

чтобы можно было легко встроить в иерархию новый объект, не меняя значения **z-index** у других элементов. Например:

```
#img1 {z-index: 10;}

#div1 {z-index: 15;}

#img2 {z-index: 20;}
#img3 {z-index: 30;}
```

Если вы хотите перевести цветное фото в режим «оттенки серого», используйте свойство **filter**. Для установки режима можно применить следующую запись:

```
img {filter: grayscale(100%);}
```

или

```
img {filter: grayscale(1);}
```

где **grayscale** — функция, определяющая уровень обесцвечивания снимка.

Если необходимо установить картинке «частичную» цветность, используйте значения в процентах от **0%** до **100%** (например, **50%**) или дробные числа от **0** до **1** (например, **0.5**).

Для поворота элементов применяют свойство **transform**, значение которого определяется функцией **rotate**. Если функция **rotate** оперирует положительными значениями, поворот выполняется вправо, если отрицательными — влево.

Очень интересный эффект перспективы можно придать объектам с помощью свойств **perspective** и **transform**.

Обратите внимание: для получения эффекта перспективы необходимо свойство **perspective** прописывать для родительского контейнера, а свойство **transform** — для внутренних элементов. Например, так:

```
<!DOCTYPE html>
...
<style>
div {... perspective: 800px;}
#spb {...}
</style>

<script>
addEventListener("load", ()=>
{
addEventListener("click", ()=>
   {
   let squ=document.getElementById("spb").style;
   squ.transition="transform 2s";
   squ.transform="rotateY(45deg)";
   });
});
</script>
</head>

<body>

<div>
<img src="img/spb.jpg" id="spb" alt="Фото">
</div>

</body>
</html>
```

2.2. События запуска визуальных эффектов

Перечислим некоторые события, которые наиболее часто запускают сценарии визуальных эффектов.

Событие **DOMContentLoaded** происходит сразу после загрузки объектной модели документа, но не дожидаясь окончания загрузки таблиц стилей, изображений и фреймов. В ряде случаев прослушивать данное событие предпочтительнее, чем событие полной загрузки документа.

Событие **load** происходит при загрузке страницы в браузер. Оно генерируется сразу после того, как элементы HTML-разметки, тексты, фреймы и рисунки (картинки, изображения, фото) полностью отображаются на экране ком-

пьютера. Такое же событие происходит и при получении ответа от сервера, на котором стоит программа, вызываемая с применением технологии Ajax.

Событие **click** возникает при щелчке кнопкой мыши на элементе разметки страницы. Это могут быть ссылки, кнопки, изображения, текстовые блоки, слои (событие **click**, как, впрочем, и некоторые другие, может также происходить даже на визуально пустом месте документа).

Событие **mouseover** происходит, когда указатель мыши перемещается внутрь границ элемента.

Событие **mouseout** противоположно предыдущему и возникает, когда указатель мыши покидает границы элемента.

Событие **mousemove** генерируется в продолжение всего времени, пока указатель мыши перемещается над элементом.

Событие **mousedown** происходит при нажатии кнопки мыши, когда ее указатель находится над элементом.

Событие **mouseup** возникает при отпускании кнопки мыши (при этом указатель должен находиться над элементом, для которого генерируется событие).

Событие **dragstart** происходит, когда посетитель начинает перетаскивать элемент по странице.

Событие **dragover** возникает в процессе перетаскивания элемента по странице.

Событие **drop** происходит в момент отпускания перетаскиваемого элемента в точке назначения.

Событие **focus** случается при получении элементом фокуса (например, когда курсор будет установлен в текстовом поле).

Событие **blur** происходит, когда элемент теряет фокус.

Событие **change** происходит на элементах **<select>**, **<input>** и **<textarea>**, когда меняются их значения.

Событие **scroll** возникает при прокручивании страницы в окне браузера или во фрейме.

Событие **keydown** происходит в момент нажатия любой кнопки на клавиатуре.

Событие **keyup** возникает в момент отпускания после нажатия клавиши.

Событие **keypress** — это «симбиоз» двух предыдущих событий (клавиша нажата и отпущена).

Событие **select** происходит в текстовых полях документа при выделении в них какого-либо фрагмента или всего текста в целом.

Событие **mouseenter** очень похоже на **mouseover**. Отличие в том, что событие **mouseenter** не всплывающее и не отменяемое (из этого понятно, что **mouseover** всплывающее и отменяемое).

Событие **mouseleave**, соответственно, похоже на **mouseout**. Разница такая же, как и в случае, описанном в предыдущем определении. Событие **mouseleave** не всплывающее и не отменяемое.

Разработчик сайта заранее выбирает, какие события на странице должны вызвать программный отклик, а какие нет.

2.3. Разные таймеры

Таймер — это устройство, выполняющее определенные действия с равными промежутками времени или задающее интервал между какими-либо событиями.

В обычной жизни мы имеем дело с таймерами повсеместно. Заводите дома будильник, чтобы встать пораньше, — тем самым включаете таймер. Судья, дав свисток в начале футбольного матча, запускает секундомер — это тоже таймер. Изображение на экране компьютера переписывается 60 раз в секунду — этим процессом также управляет таймер. Скорость автомобиля показывается на спидометре — для этого измеряется пройденный путь за промежуток времени, заданный таймером.

Естественно, что такая функциональная возможность, как вызов таймеров, есть и в JavaScript. Таймеры используются для многократного повторения некоторых фрагментов кода либо для задержки выполнения какой-то операции. Для наших примеров таймеры (в большинстве случаев, но не во всех) будут играть решающую роль. Именно они станут задавать ритм выполнения трансформаций, которые создают различные визуальные эффекты.

Мы с вами рассмотрим три варианта таймеров.

Периодический таймер **setInterval**. Многократно срабатывает через выбранный промежуток времени и выполняет заданную функцию. Форма записи:

```
setInterval(функция, интервал времени, аргументы);
```

Пример реальной записи данного таймера с аргументами:

```
setInterval(func, 1000, 3, 2);
```

Пример таймера без аргументов:

```
setInterval(func, 1000);
```

Следующий на очереди — таймер **setTimeout**. Он срабатывает однократно, запуская указанную функцию через выбранный промежуток времени. Форма записи:

```
setTimeout(функция, интервал времени, аргументы);
```

Пример данного таймера с аргументами:

```
setTimeout(func, 3000, 3, 2);
```

Пример данного таймера без аргументов:

```
setTimeout(func, 3000);
```

С помощью таймера **setTimeout** можно многократно запускать одну и ту же функцию. Для этого таймер располагают в теле функции. Таймер, вызывающий функцию из ее тела, называется вложенным. Пример:

```
function func()
{
...
setTimeout(func, 1000);
}
```

Такой таймер будет запускать функцию **func** из нее самой каждую секунду.

В качестве третьего таймера мы рассмотрим метод **requestAnimation-Frame**. У него специфическое назначение. Он необходим для синхронизации кода, выполняющего анимацию, и частоты перерисовки содержимого в окне браузера. Благодаря применению этого метода достигается плавное изменение анимированных параметров объекта на странице. Форма записи:

```
requestAnimationFrame(функция);
```

Главное отличие от предыдущих таймеров — в **requestAnimationFrame** мы не задаем интервал времени между срабатываниями. Частота срабатывания, повторимся, синхронизируется с частотой перерисовки картинки на экране. В сценарии таймер может располагаться внутри функции:

```
function func()
   {
   ...
   requestAnimationFrame(func);
   }
```

Наконец, с некоторой натяжкой в разряд однократных таймеров можно отнести CSS-свойство **transition**, с помощью которого задается один или несколько параметров, которые необходимо плавно изменить, и время их изменения. Например, так:

```
let squ=document.getElementById("square").style;
squ.transition="width 2s, height 2s, top 2s";
squ.width="400px";
squ.height="400px";
squ.top="50px";
```

Данный пример мы рассмотрим уже в следующей главе.

3. ОЖИВШИЕ ФИГУРЫ

В этой главе мы рассмотрим элементарные приемы создания визуальных эффектов. Научимся менять размеры, форму и цвет фигуры; трансформировать рамки и тени; вращать и менять перспективу отображения объектов; растворять и проявлять изображения. Будет много примеров, которые проиллюстрируют наши рассуждения, а заодно продемонстрируют использование разных таймеров при создании визуальных эффектов.

3.1. Изменение размера

Начнем с самого простого — изменения размера квадрата в течение 2 секунд.

Запустите в браузере файл **3/3.1.1.html**. Вы увидите картинку, показанную на рисунке 3.1.1. Посредине страницы расположился квадрат. Кликните мышью в любом месте, и вы увидите, как в течение 2 секунд квадрат плавно увеличится в 2 раза, одновременно оставаясь неподвижным относительно центра исходной фигуры (рис. 3.1.2).

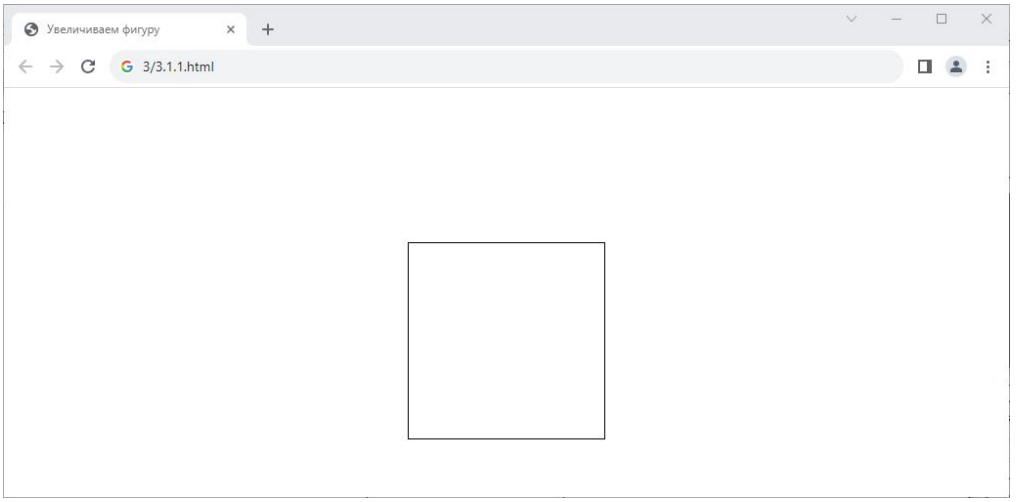

Рис. 3.1.1
Исходный вид фигуры

Код страницы очень короткий.

Есть контейнер

Файл **3/3.1.1.html**

```
<div id="square"></div>
```

квадратной формы с обозначенными границами:

```
div {position: relative; margin: auto; top: 150px;
    width: 200px; height: 200px;
      border: 1px solid #000000;}
```

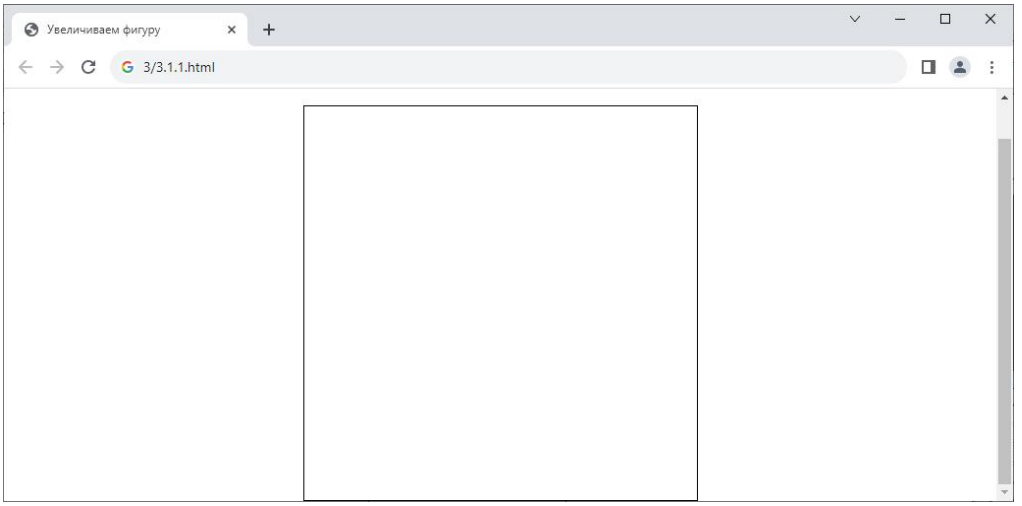

Рис. 3.1.2
Квадрат увеличился вдвое

Зарегистрируем обработчик клика мышью:

```
addEventListener("load", ()=>
{
addEventListener("click", ()=>
   {
   ...
   });
});
```

Создадим переменную, через имя которой мы будем обращаться к свойствам фигуры:

```
let squ=document.getElementById("square").style;
```

Для плавного изменения размеров контейнера используем свойство **transition**. В течение 2 секунд у квадрата будет увеличиваться ширина, высота и положение относительно верхней границы окна браузера (чтобы он оставался неподвижным относительно центра исходной фигуры):

```
squ.transition="width 2s, height 2s, top 2s";
```

На завершающем этапе указываем новые размеры фигуры:

```
squ.width="400px";
squ.height="400px";
squ.top="50px";
```

Как видите, сценарий действительно оказался элементарным.

Полный код программы:

```
<!DOCTYPE html>
<html lang="ru">
<head>
<meta charset="utf-8">
<title>Увеличиваем фигуру</title>
<style>
div {position: relative; margin: auto; top: 150px;
```

```
    width: 200px; height: 200px;
      border: 1px solid #000000;}
</style>
<script>
addEventListener("load", ()=>
{
addEventListener("click", ()=>
   {
   let squ=document.getElementById("square").style;
   squ.transition="width 2s, height 2s, top 2s";
   squ.width="400px";
   squ.height="400px";
   squ.top="50px";
   });
});
</script>
</head>

<body>

<div id="square"></div>

</body>
</html>
```

Следующий пример продемонстрирует тот же самый эффект. Только теперь мы используем таймер **setInterval**. При этом код сценария кардинально изменится.

Начинается скрипт с регистрации трех переменных, которые хранят начальные размеры ширины, высоты и значение отступа от верхней границы:

Файл **3/3.1.2.html**

```
let w=200;
let h=200;
let t=150;
```

Дальше создаем переменную, через имя которой станем обращаться к свойствам фигуры:

```
let squ=document.getElementById("square").style;
```

Запускаем таймер:

```
let si=setInterval(()=>
   {
   ...
   }, 10);
```

На каждом его проходе увеличиваем значения переменных

```
w+=2;
h+=2;
t-=1;
```

и, соответственно, размеры контейнера, одновременно смещая его вверх:

```
squ.width=w+"px";
squ.height=h+"px";
squ.top=t+"px";
```

Когда квадрат увеличится в 2 раза, вызываем метод **clearInterval**, который останавливает таймер.

```
if(w==400)
   clearInterval(si);
```

Полный код сценария:

```
addEventListener("load", ()=>
{
addEventListener("click", ()=>
   {
   let w=200;
   let h=200;
   let t=150;

   let squ=document.getElementById("square").style;

   let si=setInterval(()=>
      {
      w+=2;
      h+=2;
      t-=1;

      squ.width=w+"px";
      squ.height=h+"px";
      squ.top=t+"px";

      if(w==400)
         clearInterval(si);
      }, 10);
   });
});
```

В третьем примере мы используем метод **setTimeout**.

Клик мышью запускает функцию **func**:

Файл **3/3.1.3.html**

```
addEventListener("click", func);
```

Следующий фрагмент кода вам уже знаком:

```
let w=200;
let h=200;
let t=150;

let squ=document.getElementById("square").style;
```

Осталось написать саму функцию:

```
function func()
   {
   if(w<400)
      {
      w+=2;
      h+=2;
      t-=1;

      squ.width=w+"px";
      squ.height=h+"px";
      squ.top=t+"px";
```

```
      setTimeout(func, 10);
      }
   }
```

При каждом ее запуске значения переменных увеличиваются, квадрат «растет» и смещается вверх.

Полный код сценария:

```
addEventListener("load", ()=>
{
addEventListener("click", func);

let w=200;
let h=200;
let t=150;

let squ=document.getElementById("square").style;

function func()
   {
   if(w<400)
      {
      w+=2;
      h+=2;
      t-=1;

      squ.width=w+"px";
      squ.height=h+"px";
      squ.top=t+"px";

     setTimeout(func, 10);
      }
   }
});
```

В четвертом примере задействуем таймер **requestAnimationFrame**. Все отличие новой программы от предыдущей — функция **func** перезапускается следующей инструкцией:

Файл **3/3.1.4.html**

```
requestAnimationFrame(func);
```

Полный код сценария:

```
addEventListener("load", ()=>
{
addEventListener("click", func);

let w=200;
let h=200;
let t=150;

let squ=document.getElementById("square").style;

function func()
   {
   if(w<400)
      {
```