Лабораторная работа №1
«Знакомство с контекстом рисования элемента <canvas>»

Оглавление

Введение

В лабораторной работе №1 мы познакомимся с элементом <canvas> исследовав процесс создания нескольких примеров программ.

Что такое <canvas>?

До появления HTML5, если требовалось вывести изображение на веб-странице, единственным стандартным способом сделать это было использование тега <img>. Этот тег, хотя и является довольно удобным инструментом, имеет множество ограничений и не позволяет создавать и выводить изображения динамически. Это стало одной из основных причин появления сторонних решений, таких как Flash Player.

Однако тег <canvas> в HTML5 изменил положение дел, дав удобную возможность рисовать компьютерную графику динамически с помощью JavaScript.

Подобно холсту (canvas) художника, тег <canvas> определяет область рисования на веб-странице. Только вместо кистей и красок, для рисования в этой области используется JavaScript. С его помощью можно рисовать точки, линии, прямоугольники, окружности и другие геометрические фигуры, вызывая методы JavaScript, поддерживаемые для тега <canvas>. На рис. 1 показан простенький графический редактор, использующий тег <canvas>.

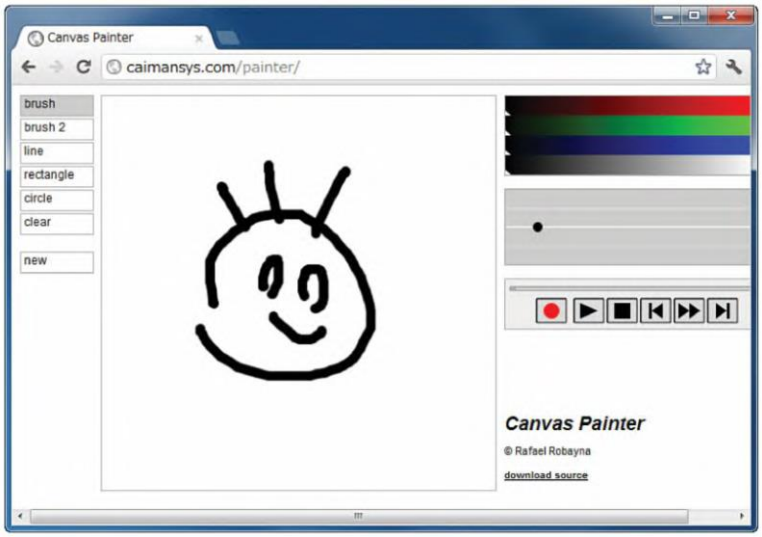


Рис. 1. Простой графический редактор, использующий тег <canvas>

(http://caimansys.com/painter/)

Этот графический редактор представляет собой веб-страницу и позволяет в интерактивном режиме рисовать линии, прямоугольники и окружности, и даже изменять их цвет.

Помимо графических редакторов, с помощью элемента <canvas> можно отображать диаграммы, игры, анимацию и т.п.

Полное описание текущей версии интерфейса API для элемента управления <canvas> находится по адресу https://html.spec.whatwg.org/multipage/canvas.html#the-canvas-element

Мы не будем создавать что-то сложное, а просто рассмотрим основные функции тега <canvas>, воспользовавшись примером программы 1, которая рисует закрашенный зеленый прямоугольник на веб-странице. На рис. 2 показан результат выполнения программы 1 в браузере.

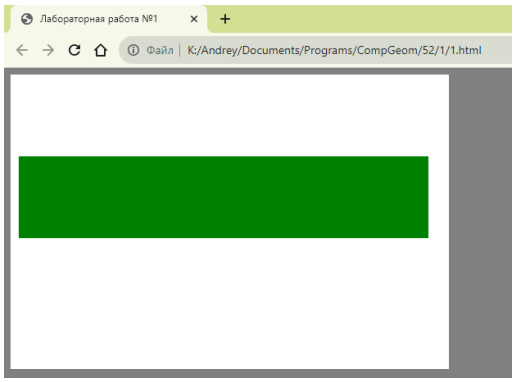


Рис. 2. Программа 1

Содержимое файла 1.html

Итак, посмотрим, как действует программа 1 и выясним, как использовать тег <canvas> в файле HTML.

Как и другие HTML-элементы, canvas – это просто элемент в составе веб-страницы с определенными размерами, внутри которого можно для рисования графики использовать JavaScript. С помощью атрибутов width и height тег <canvas> определяет область рисования размером 535×360 пикселей. Если они не указаны, размеры канвы составят 300x150 пикселов. Атрибут id определяет идентификатор этой области, чтобы в коде JavaScript было понятно к какому именно холсту идет обращение (ведь на странице может быть несколько холстов). Контент между тегами <canvas> и </canvas> добавляется для страховки и выводится на экран, только если элемент <canvas> не поддерживается.

Чтобы что-то нарисовать в области, определяемой тегом <canvas>, необходим код на JavaScript, выполняющий операции рисования. Соответствующий код можно включить непосредственно в файл HTML или сохранить его в отдельном файле. В наших примерах мы будем использовать второй подход, потому что так проще будет читать программный код. Независимо от выбранного подхода, нужно сообщить браузеру, как запустить код на JavaScript. В нашем примере с помощью атрибута onload элемента <body>, мы сообщаем браузеру, что нужно вызвать функцию main() после загрузки элемента <body>.

С помощью атрибута src элемента <script> происходит подключение файла 1.js с кодом на JavaScript, где определена функция main().

Для простоты, во всех примерах программ, файлам JavaScript будем присваивать те же имена, что и соответствующим им файлам HTML.

Содержимое файла 1.js

Файл 1.js содержит код программы на языке JavaScript, рисующей зеленый прямоугольник в области рисования, определяемой тегом <canvas>. Для рисования требуется выполнить три этапа:

1. Получить ссылку на элемент <canvas>.

2. Запросить контекст отображения двухмерной графики.

3. Нарисовать двухмерное изображение, используя методы контекста.

Рассмотрим эти этапы по порядку.

Получить ссылку на элемент <canvas>

Получить ссылку на элемент `<canvas>` в программе JavaScript можно с помощью метода `document.getElementById()`. Этот метод имеет единственный параметр – строковый идентификатор, определяемый атрибутом `id` в теге `<canvas>` в файле `1.html`.

Если метод вернет значение, отличное от `null`, значит программе удалось получить ссылку на искомый элемент. В противном случае попытка считается неудачной. Проверить успешность попытки можно с помощью простой инструкции `if`. В случае неудачи вызывается метод `console.log()`, который выводит свой параметр в виде строки в консоль браузера.

Примечание. В браузере Chrome консоль можно открыть, выбрав пункт меню Дополнительные инструменты->Инструменты разработчика, или нажав клавишу F12 (см. рис. 3); в Firefox то же самое можно сделать, выбрав пункт меню `Tools-> Web Developer->Web Console` (Инструменты->Веб-разработка->Веб-консоль) или нажав комбинацию клавиш `<Ctrl+Shift+K>`.

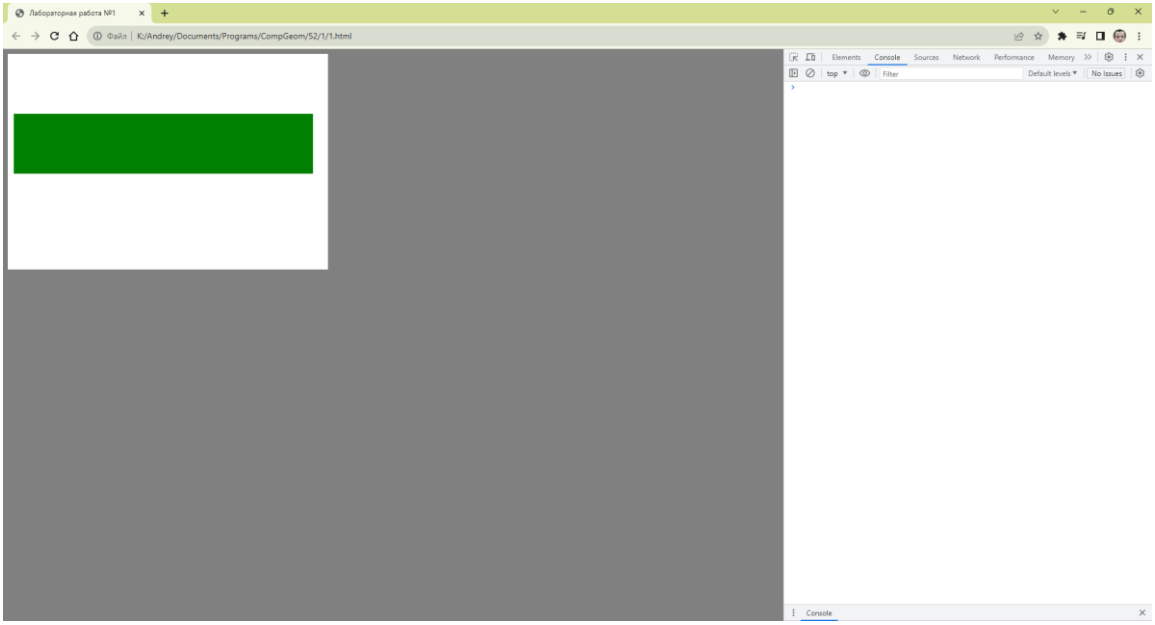


Рис. 3. Консоль в Chrome

Запросить контекст отображения двухмерной графики

Элемент `<canvas>` дает доступ к механизму, который называют контекстом рисования. Контекст рисования — это JavaScript-объект, обладающий методами и свойствами, при помощи которых можно рисовать на «холсте». Чтобы получить этот объект, мы вызываем для `canvas` метод `getContext()`.

Метод `canvas.getContext()` имеет параметр, определяющий тип требуемого механизма рисования. В данной лабораторной работе мы будем использовать двухмерную графику, поэтому методу передается строка `'2d'` (будьте внимательны, регистр символов имеет значение).

В результате этого вызова мы получаем контекст (объект класса `CanvasRenderingContext2D`) и сохраняем его в переменной `ctx`. Здесь не выполняется проверка ошибок, которую обязательно следует делать в реальных программах.

Нарисовать двухмерное изображение, используя методы контекста

Получив контекст отображения, посмотрим, как выполняется рисование зеленого прямоугольника. Делается это в два этапа. Сначала устанавливается цвет, а затем этим цветом рисуется (или заливается) сам прямоугольник.

Цвет может быть указан любым способом, поддерживаемым CSS.

Примеры указания цвета:

```
ctx.fillStyle = "Green"; // Set color to green

ctx.fillStyle = "#00ff00";

ctx.fillStyle = 'rgb(0, 255, 0)'; // Set color to green
```

Все эти выражения задают для заливки зеленый непрозрачный цвет.

JavaScript понимает английские названия более 100 цветов, например Green, Blue, Orange, Red, Yellow, Purple, White, Black, Pink, Turquoise, Violet, SkyBlue, PaleGreen и др. Полный список можно найти на сайте CSS-Tricks: http://css-tricks.com/snippets/css/named-colors-and-hex-equivalents/.

Ключевое слово `rgb` в строковом значении `'rgb(0, 0, 255)'` определяет три составляющие цвета: `r` (red – красный), `g` (green – зеленый), `b` (blue – синий), каждая из которых может иметь значение от `0` (наименьшее) до `255` (наибольшее). Вообще цвет в компьютерных системах обычно представлен комбинацией красной, зеленой и синей составляющих (три основных цвета), – этот формат называют `RGB`. При добавлении альфа-составляющей (прозрачность), формат называют `RGBA`. По умолчанию цвет заливок черный и непрозрачный.

Цвет также может определяться цифрами в шестнадцатеричном коде. Для каждой составляющей цвета (красного, зеленого и синего) задается значение в пределах от `00` до `FF`. Эти значения объединяются в одно число, перед которым добавляется символ "`#`", например, значение `#FF0000` соответствует красному цвету, `#00FF00` — ярко-зеленому, а `#FF00FF` — фиолетовому (смеси красного и синего). Шестнадцатеричные коды наиболее популярных цветов, соответствующих CSS-цветам, также можно посмотреть на сайте CSS-Tricks: http://css-tricks.com/snippets/css/named-colors-and-hex-equivalents/.

Свойство `fillStyle` определяет цвет заливки. Все фигуры, которые мы впоследствии нарисуем, будут заполнены указанным цветом.

Прежде чем погрузиться в детали аргументов функций рисования фигур, рассмотрим систему координат, используемую элементом `<canvas>` (см. рис. 4).

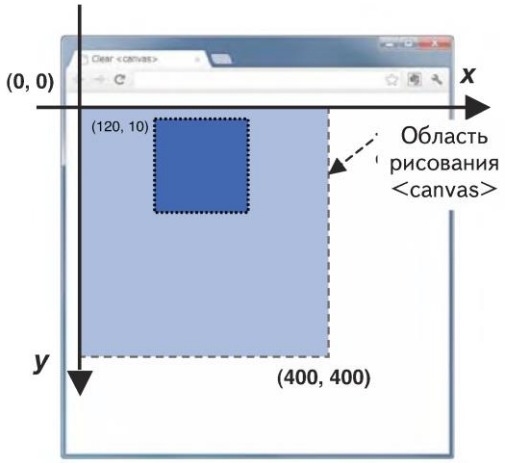


Рис. 4. Система координат элемента `<canvas>`

Как показано на рис. 4, начало системы координат элемента `<canvas>` находится в верхнем левом углу, ось X определяет координату по горизонтали (слева направо), а ось Y – координату по вертикали (сверху вниз).

Метод контекста рисования `fillRect` принимает четыре аргумента. Первые два параметра метода определяют координаты левого верхнего угла рисуемого прямоугольника в системе координат элемента `<canvas>`, а третий и четвертый параметры – ширину и высоту прямоугольника (в пикселях).

После того, как браузер загрузит файл `1.html`, вы увидите прямоугольник, как показано на рис. 2.

Поверх зеленого прямоугольника нарисуем прямоугольник с заливкой, но уже полупрозрачным синим цветом:

```
ctx.fillstyle = "rgba(0, 0, 127, 0.5)";
```

Значение альфа-канала определяется числом в диапазоне от `0.0` (прозрачный) до `1.0` (непрозрачный).

Свойство `globalAlpha` указывает уровень прозрачности для любой графики, которую мы впоследствии нарисуем. Уровень прозрачности также задается в виде числа от `0.0` (полностью прозрачный) до `1.0` (полностью непрозрачный; значение по умолчанию).

Для рисования прямоугольника без заливки (т. е. одного лишь контура прямоугольника) предназначен метод `strokeRect`. Он принимает те же аргументы, что и `fillRect`.

Изменить цвет контура можно с помощью свойства `strokeStyle`. Сам цвет также может быть задан любым поддерживаемым CSS способом.

Метод `clearRect` очищает заданную прямоугольную область от любой присутствовавшей там графики. Вызывается он так же, как методы `strokeRect` и `fillRect`.

Вывод текста

Для вывода текста, представляющего собой один лишь контур без заливки, используется метод `strokeText`:

```
<контекст рисования>.strokeText(<выводимый текст>, <горизонтальная координата>, <вертикальная координата> [ , <максимальная ширина>] );
```

С первыми тремя параметрами все ясно. Четвертый, необязательный, параметр определяет максимальное значение ширины, которую может принять выводимый на канву текст.

Метод `fillText` выводит заданный текст в виде сплошной заливки. Вызывается он так же, как метод `strokeText`.

Свойство `font` задает параметры шрифта, которым будет выводиться текст. Эти параметры указывают в том же формате, что и у значения атрибута CSS font.

Свойство `textAlign` устанавливает горизонтальное выравнивание выводимого текста относительно точки, в которой он будет выведен (координаты этой точки задаются вторым и третьим параметрами методов `strokeText` и `fillText`). Это свойство может принимать следующие значения:

- `"left"` — выравнивание по левому краю;
- `"right"` — выравнивание по правому краю;
- `"start"` — выравнивание по левому краю, если текст выводится по направлению слева направо, и по правому краю в противном случае (значение по умолчанию);
- `"end"` — выравнивание по правому краю, если текст выводится по направлению слева направо, и по левому краю в противном случае;
- `"center"` — выравнивание по центру.

Свойство `textBaseline` позволяет задать вертикальное выравнивание выводимого текста относительно точки, в которой он будет выведен. Доступны следующие значения:

- `"top"` — выравнивание по верху прописных букв;
- `"hanging"` — выравнивание по верху строчных букв;
- `"middle"` — выравнивание по средней линии строчных букв;
- `"alphabetic"` — выравнивание по базовой линии букв европейских алфавитов (значение по умолчанию);
- `"ideographic"` — выравнивание по базовой линии иероглифических символов (она находится чуть ниже базовой линии букв европейских алфавитов);
- `"bottom"` — выравнивание по низу букв.

Метод `measureText` позволяет узнать ширину текста, выводимого на канву:

```
<контекст рисования>.measureText (<текст>)
```

Текст указывается в виде строки. Метод возвращает объект с единственным свойством `width`, которое и хранит ширину текста в пикселах, заданную в виде числа.

Рисование сложных фигур

Канва может выводить не только прямоугольники. С ее помощью можно нарисовать фигуру практически любой сложности.

Как рисуются сложные фигуры

Рисование контура такой фигуры начинается с вызова метода `beginPath`. Этот метод не принимает параметров и не возвращает результат.

Рисование сложной фигуры вызывается одним из двух методов: `stroke` или `fill`. Первый метод просто завершает рисование контура фигуры, второй, помимо этого, замыкает контур, если он не замкнут, и осуществляет заливку получившейся фигуры. Оба этих метода не принимают параметров и не возвращают результатов.

Если до метода `stroke` вызывается метод `closePath`, то контур также замыкается.

Рассмотрим методы, которые предназначены для рисования разнообразных линий, образующих границы сложного контура.

Перо. Перемещение пера

При рисовании сложного контура используется концепция пера – воображаемого инструмента рисования. Перо можно перемещать в любую точку на канве. Рисование каждой линии контура начинается в точке, где в данный момент находится перо. После рисования каждой линии перо перемещается в ее конечную точку, из которой тут же можно начать рисование следующей линии контура.

Изначально, сразу после загрузки Web-страницы, перо находится в точке с координатами [0,0], т.е. в верхнем левом углу канвы. Переместить перо в точку канвы, где мы собираемся начать рисование контура, позволяет метод `moveTo`:

```
<контекст рисования>.moveTo(<горизонтальная координата>,
<вертикальная координата>);
```

Параметры указываются в пикселах в виде чисел. Метод `moveTo` не возвращает результат.

Прямые линии

Прямые линии рисовать проще всего. Для этого служит метод `lineTo`:

```
<контекст рисования>.lineTo(<горизонтальная координата>,
<вертикальная координата>);
```

Параметры задаются в виде чисел в пикселах. Метод `lineTo` не возвращает результат.

Дуги

Для рисования дуг предусмотрены два метода `arc` и `arcTo`.

```
<контекст рисования>.arc(<горизонтальная координата>,
<вертикальная координата>, <радиус>, <начальный угол>, <конечный
угол>, true|false);
```

Первые три параметра указываются в пикселах, четвертый и пятый — в радианах. Углы отсчитываются от горизонтальной оси. Если шестой параметр имеет значение `true`, то дуга рисуется против часовой стрелки, а если `false` — по часовой стрелке. Значение `false` установлено по умолчанию и его указание может быть опущено. Метод `arc` не возвращает значений.

Тот факт, что углы задаются в радианах, несколько осложняет работу. Нам придется пересчитывать величины углов из градусов в радианы с помощью следующего выражения:

```
radians = (Math.PI / 180) * degrees;
```

Здесь переменная `degrees` хранит значение угла в градусах, а переменная `radians` будет хранить то же значение, но в радианах.

```
<контекст рисования>.arcTo (
      <горизонтальная координата второй контрольной точки>,
      <вертикальная координата второй контрольной точки>,
      <горизонтальная координата конечной точки>,
      <вертикальная координата конечной точки>,
      <радиус>);
```

Метод строит дугу окружности по трем точкам. Эти точки формируют две пересекающиеся линии, которые являются касательными к строящейся дуге в ее начальной и конечной точке.

Метод `arcTo` рисует контур только до той точки, где дуга касается второй линии. Поэтому для полной визуализации второй линии, образующей угол, вызывается метод `lineTo`.

Когда метод `arcTo` вызывается со значением радиуса, равным `0`, создается острое соединение.

Кривые Безье

Для рисования кривых Безье по четырем контрольным точкам используется метод

```
<контекст рисования>.bezierCurveTo(
     <горизонтальная координата второй контрольной точки>,
     <вертикальная координата второй контрольной точки>,
     <горизонтальная координата третьей контрольной точки>,
     <вертикальная координата третьей контрольной точки>,
     <горизонтальная координата конечной точки>,
     <вертикальная координата конечной точки>);
```

Метод `quadraticCurveTo` рисует кривые Безье по трем контрольным точкам:

```
<контекст рисования>.quadraticCurveTo (
<горизонтальная координата второй контрольной точки>,
<вертикальная координата второй контрольной точки>,
<горизонтальная координата конечной точки>,
<вертикальная координата конечной точки>);
```

Прямоугольники

Нарисовать прямоугольник в составе сложного контура можно, вызвав метод `rect`:

```
<контекст рисования>.rect{<горизонтальная координата>,
<вертикальная координата>, <ширина>, <высота>);
```

По окончании рисования прямоугольника перо будет установлено в точку с координатами [0,0], т. е. в верхний левый угол канвы.

Задание стиля линий

Канва позволяет задать стиль линий, а именно толщину, форму их начальных и конечных точек и точек соединения линий друг с другом.

Свойство `lineWidth` задает толщину линий контура (в пикселах в виде числа).

Когда рисуемые линии заканчиваются и имеют ширину, превышающую 1 пиксел, можно выбрать тип появляющейся законцовки этой линии, используя свойство `lineCap`. Его значение может быть одной из следующих строк:

- `"butt"` (срез) — начальная и конечная точки как таковые отсутствуют (значение по умолчанию);
- `"round"` (круг) — начальная и конечная точки имеют вид кружков;
- `"square"` (квадрат) — начальная и конечная точки имеют вид квадратиков.

Свойство `lineJoin` задает форму точек соединения линий друг с другом. Его значение может быть одной из следующих строк:

- `"miter"` (клин) — точки соединения имеют вид острого или тупого угла (значение по умолчанию);
- `"round"` (закругление) — точки соединения, образующие острые углы, скругляются;
- `"bevel"` (скос) — острые углы, образуемые соединяющимися линиями, как бы срезаются.

Свойство `miterLimit` задает дистанцию от точки соединения, на которую могут выступать острые углы, образованные соединением линий, когда для свойства `lineJoin` задано значение `"miter"`. Углы, выступающие на большую дистанцию, будут срезаны.

Определение вхождения точки в состав контура

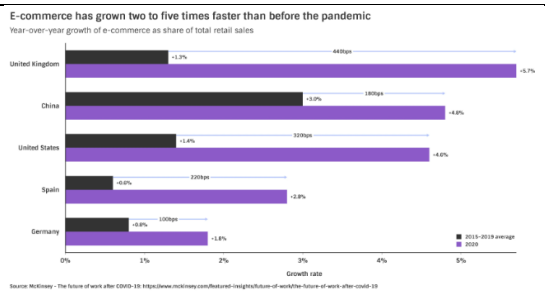
Иногда бывает необходимо выяснить, входит ли точка с заданными координатами в состав контура сложной фигуры. Это можно сделать с помощью метода `isPointInPath`:

```
<контекст рисования>.isPointInPath ( <горизонтальная координата>,
<вертикальная координата>);
```

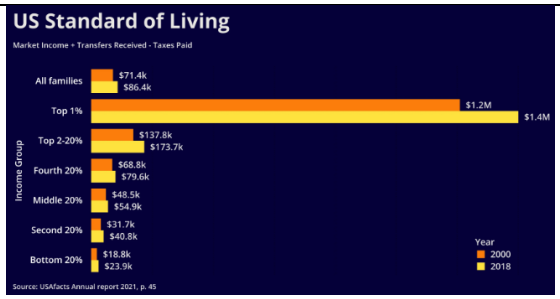
Обе координаты проверяемой точки указываются в виде чисел в пикселах. Метод возвращает `true`, если точка с такими координатами входит в состав контура, и `false` — в противном случае.

1.1 Задание для самостоятельной работы

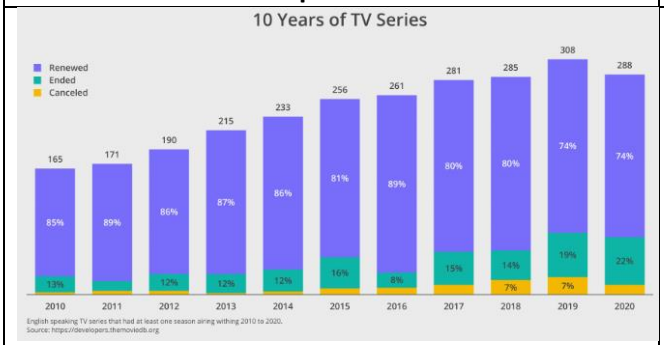
В соответствии с Вашим вариантом, воспроизведите основные геометрические элементы и текстовые надписи, показанные на рисунке.



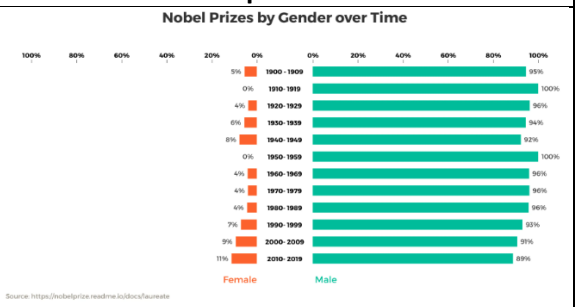
Вариант 1



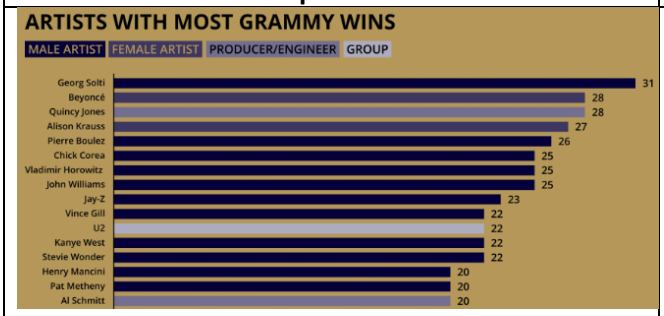
Вариант 2



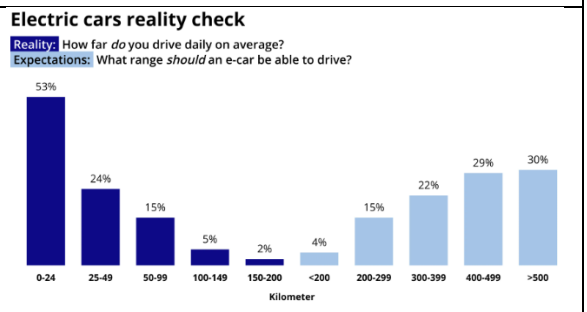
Вариант 3



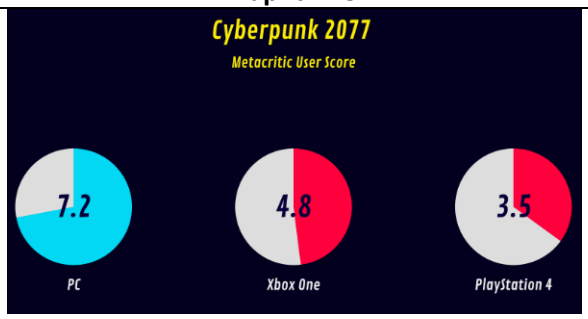
Вариант 4



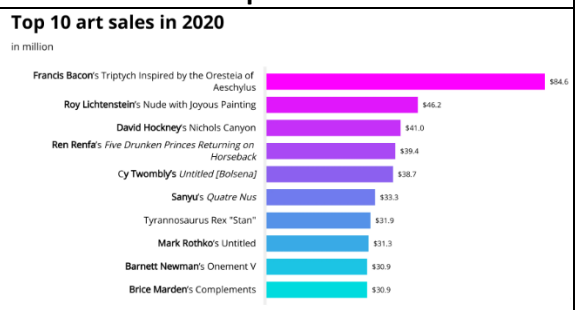
Вариант 5



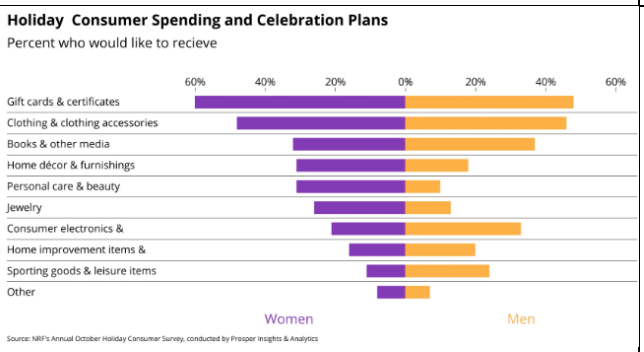
Вариант 6



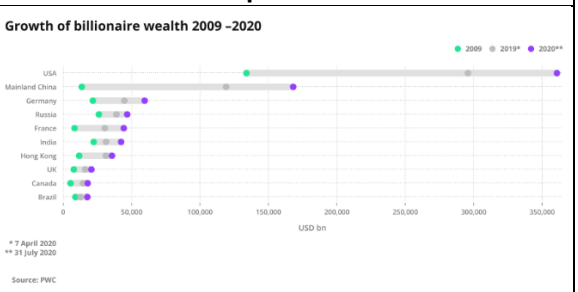
Вариант 7



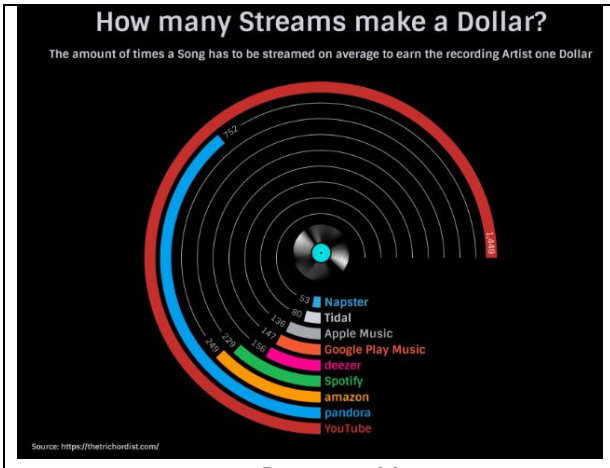
Вариант 8



Вариант 9

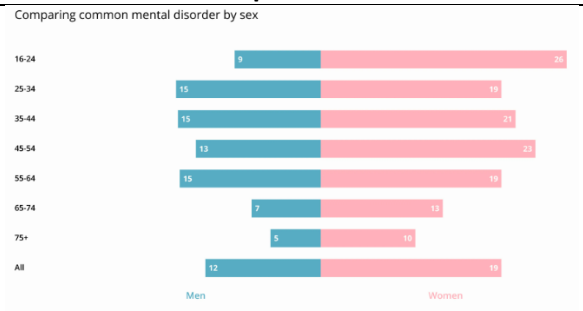


Вариант 10

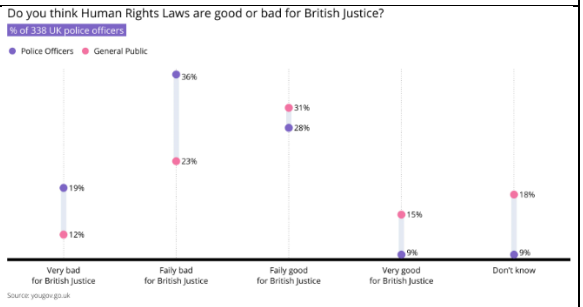


Вариант 11

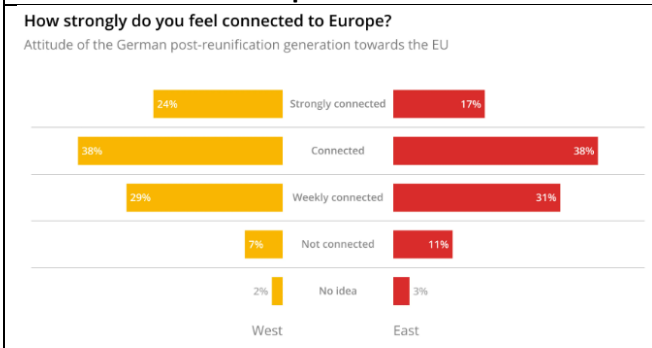
Вариант 12



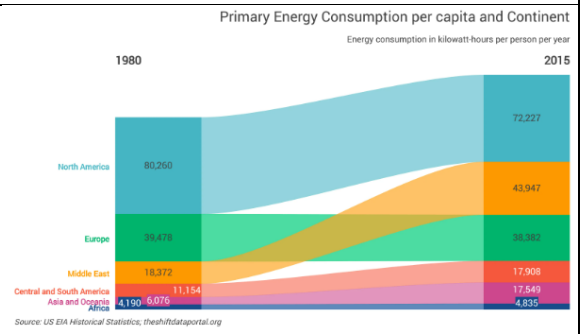
Вариант 13



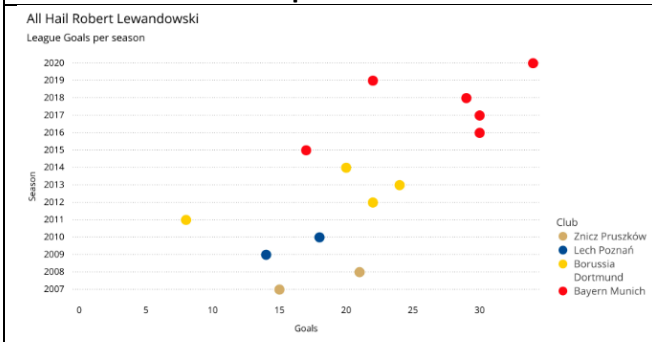
Вариант 14



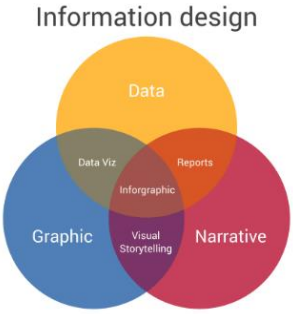
Вариант 15



Вариант 16



Вариант 17



Вариант 18

Использование сложных цветов

Помимо однотонных цветов, холст позволяет использовать для закрашивания линий и заливок градиенты и даже выполнять заливку текстурой.

Линейный градиент

Линейный градиент создают в три этапа. Первый этап — вызов метода `createLinearGradient()` — собственно создание линейного градиента:

```
<контекст рисования>.createLinearGradient
```

```
(<Горизонтальная координата начальной точки>,
 <вертикальная координата начальной точки>,
 <горизонтальная координата конечной точки>,
 <вертикальная координата конечной точки>)
```

Координаты начальной и конечной точек градиента отсчитываются относительно холста (а не заливаемого объекта).

Метод `createLinearGradient()` возвращает объект класса `CanvasGradient`, представляющий созданный линейный градиент.

Второй этап — расстановка ключевых точек градиента. Здесь нам понадобится метод `addColorStop()` класса `CanvasGradient`:

```
<градиент>.addColorStop(<положение ключевой точки>, <цвет>)
```

Первый параметр задается в виде числа от 0.0 (начало прямой) до 1.0 (конец прямой). Результат этот метод не возвращает.

В нашем примере определяются две ключевые точки, а именно, самым первым цветом градиента указывается белый, а самым последним — черный. Затем градиент будет плавно переходить между этими цветами по всему холсту слева направо.

Третий этап — использование готового линейного градиента. Для этого представляющий его объект класса `CanvasGradient` следует присвоить свойству `strokeStyle` или `fillStyle`.

Указывая для градиента различные стартовые и конечные позиции, можно придать ему наклон для распространения в любом направлении.

Создание промежуточных ключевых точек

В градиенте можно использовать сколько угодно ключевых точек, а не только стартовую и конечную. Это позволяет дать четкое описание почти что любому типу представляемого градиентного эффекта. Для этого нужно указать процент градиента, занимаемый каждым цветом. Для распределения по градиенту стартовой позиции в формате числа с плавающей точкой берется диапазон между 0 и 1. Конечную позицию цвета вводить не нужно, поскольку она выводится из стартовой позиции следующей ключевой точки или же градиент заканчивается, если позиция является последней указанной вами.

В следующем примере создаётся эффект радуги с помощью настройки промежуточных ключевых точек. Все цвета расположены примерно на одинаковом расстоянии друг от друга (каждому цвету выделено 14 % градиента, а последнему — 16 %) но придерживаться этого не обязательно. Какие-то цвета можно прижать друг к другу, а каким-то дать больше простора. Сколько цветов использовать и где в градиенте они должны стартовать и финишировать, отдается полностью на ваше усмотрение.

Радиальный градиент

Радиальный градиент также создают в три этапа. Первый этап — вызов метода `createRadialGradient()` — создание объекта радиального градиента:

```
<контекст рисования>.createRadialGradient
(<горизонтальная координата центра внутренней окружности>,
 <вертикальная координата центра внутренней окружности>,
```

```
<радиус внутренней окружности>,
<горизонтальная координата центра внешней окружности>,
<вертикальная координата центра внешней окружности>,
<радиус внешней окружности>)
```

Параметры этого метода определяют координаты центров и радиусы обеих окружностей, описывающих радиальный градиент. Они задаются в пикселах в виде чисел.

Метод `createRadialGradient()` возвращает объект класса `CanvasGradient`, представляющий созданный нами радиальный градиент.

Второй этап — расстановка ключевых точек — выполняется с помощью уже знакомого нам метода `addColorStop()` класса `CanvasGradient`. Только в данном случае первый параметр определит относительное положение создаваемой ключевой точки на промежутке между внутренней и внешней окружностями. Он задается в виде числа от 0.0 (начало промежутка, т. е. внутренняя окружность) до 1.0 (конец промежутка, т. е. внешняя окружность).

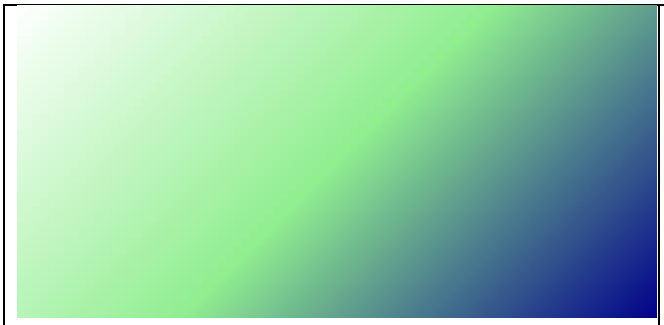
И третий этап — использование созданного градиента.

В первом примере градиент просто начинается в центре окружности, а затем распространяется наружу. Координаты стартовой и конечной позиций те же самые, но радиус задан нулевым для стартовой позиции и охватывает весь градиент для конечной позиции.

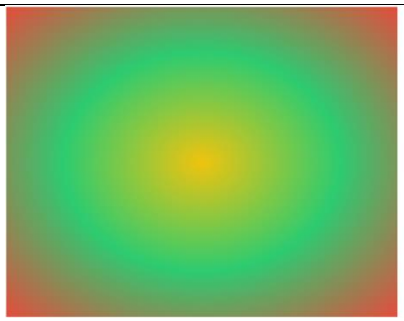
Во втором примере градиент начинается с центра с координатами (0; 120) и радиусом 0 пикселов, а заканчивается в центре с координатами (480; 120) и радиусом 480 пикселов.

1.2 Задание для самостоятельной работы

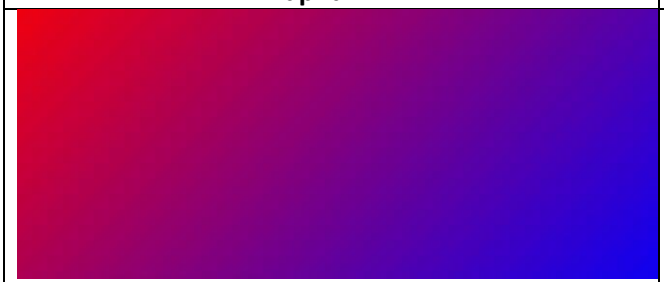
Нарисуйте фигуру и закрасьте её с помощью градиентной заливки в соответствие с вашим вариантом.



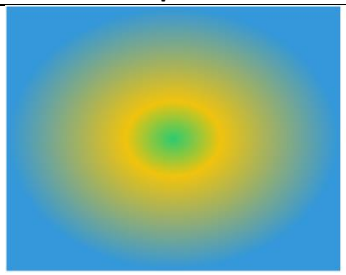
Вариант 1



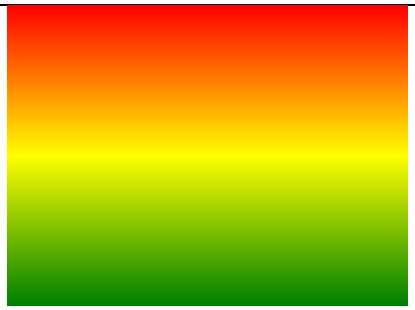
Вариант 2



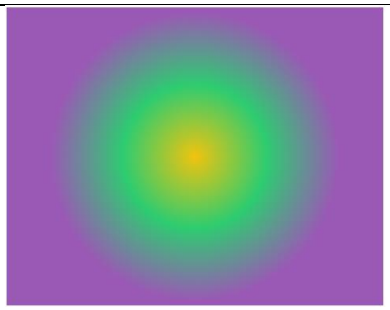
Вариант 3



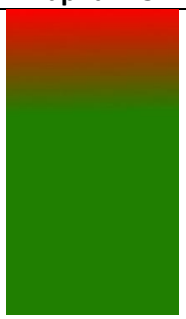
Вариант 4



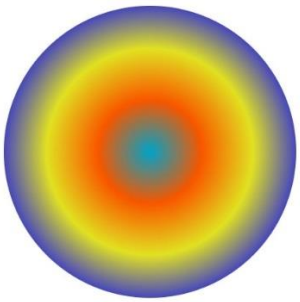
Вариант 5



Вариант 6

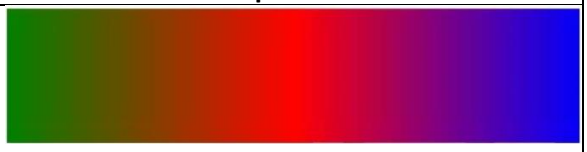


Вариант 7



Вариант 8

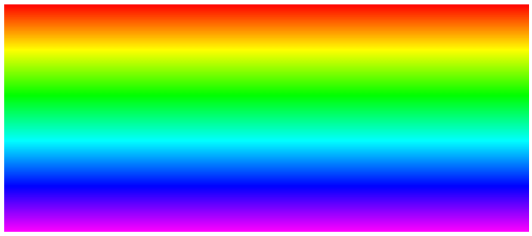
Вариант 9



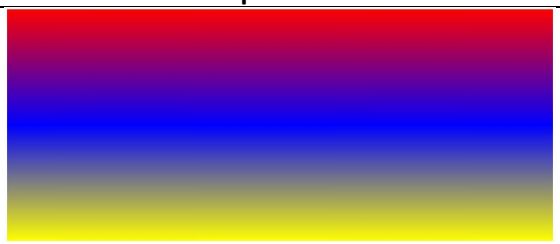
Вариант 10

Вариант 11

Вариант 12

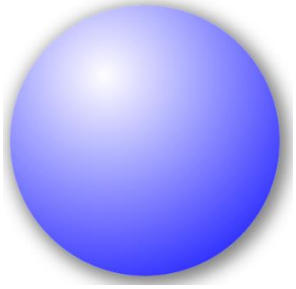


Вариант 13

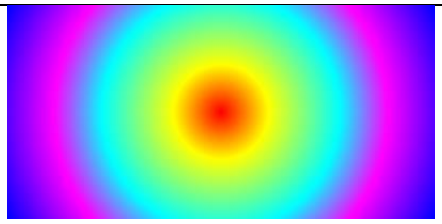
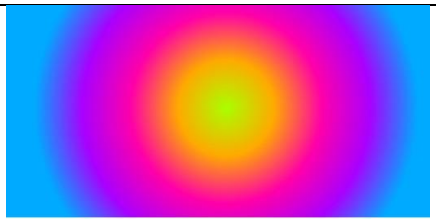


Вариант 14

Вариант 15



Вариант 16

	
Вариант 17	**Вариант 18**

Заливка текстурой

Текстура — это обычное изображение, которым закрашиваются линии или заливки. Таким изображением может быть содержимое как графического файла, так и другого холста.

Графический цвет создают в три этапа. Первый этап необходим только в том случае, если мы используем в качестве цвета содержимое графического файла. Такой файл нужно как-то загрузить, удобнее всего — с помощью объекта класса `Image`, который представляет графическое изображение, хранящееся в файле.

Сначала с помощью оператора `new` создаем объекта класса `Image`. Следующая инструкция прикрепляет к событию `onload` функцию, создающую повторяющийся узор для свойства `fillStyle`. Далее мы присваиваем свойству `src` этого объекта интернет-адрес загружаемого графического файла в виде строки. Все — теперь можно использовать данный экземпляр объекта `Image` для создания графического цвета.

Если бы в данном примере не использовалось событие `onload`, а вместо этого код просто выполнялся бы сразу, как только он попадался в сценарии, изображение могло быть к этому времени еще не загруженным и могло не выводиться на экран. Прикрепление к этому событию гарантирует доступность изображения для использования на холсте, поскольку событие наступает только в результате успешной загрузки изображения.

Второй этап — собственно создание графического цвета с помощью метода `createPattern`:

```
<контекст рисования>.createPattern(<графическое изображение или холст>, <режим повторения>);
```

Первый параметр задает графическое изображение в виде объекта класса `Image`, элемента `img` или объекта другого холста.

Часто бывает так, что размеры заданного графического изображения меньше, чем фигуры, к которой должен быть применен графический цвет. В этом случае изображение повторяется столько раз, чтобы полностью «вымостить» линию или заливку. Режим такого повторения задает второй параметр метода `createPattern()`. Его значение должно быть одной из следующих строк:

- `"repeat"` — изображение будет повторяться по горизонтали и вертикали;
- `"repeat-x"` — изображение будет повторяться только по горизонтали;
- `"repeat-y"` — изображение будет повторяться только по вертикали;
- `"no-repeat"` — изображение не будет повторяться никогда; в этом случае часть фигуры останется не занятой им.

Метод `createPattern()` возвращает объект класса `CanvasPattern`, представляющий созданный нами графический цвет.

Третий этап — использование готового графического цвета — выполняется так же, как для градиентов.

В нашем примере мы загружаем графический файл graphic_color.jpg, создаем на его основе повторяющийся по горизонтали и вертикали графический цвет и рисуем с его помощью прямоугольник.

Узор заливки применяется по отношению ко всей области холста, поэтому, когда команда заливки настроена на применение только к меньшей по размеру области внутри холста, изображения выводятся обрезанными.

Создание тени

Холст позволяет создавать тень у всех рисуемых фигур. Для задания ее параметров применяют четыре свойства:

- `shadowOffsetX` — смещение тени по горизонтали относительно фигуры (в виде числа в пикселах, значение по умолчанию — 0). При положительном значении тень смещается вправо, а при отрицательном – влево;
- `shadowOffsetY` — смещение тени по вертикали относительно фигуры (в виде числа в пикселах, значение по умолчанию — 0). При положительном значении тень смещается вниз, а при отрицательном – вверх;
- `shadowBlur` — степень размытия тени в виде числа: чем больше это число, тем сильнее размыта тень (значение по умолчанию — 0, т. е. отсутствие размытия);
- `shadowColor` — цвет тени (по умолчанию — черный непрозрачный). Если применяется размытие, этот цвет будет в размываемой области смешиваться с фоном.

1.3 Задание для самостоятельной работы

Добавьте тень к рисунку, который вы нарисовали в задании для самостоятельной работы № 1.2.

Управление наложением графики

Когда мы рисуем какую-либо фигуру поверх уже существующей, новая фигура накладывается на старую, перекрывая ее. Это поведение холста по умолчанию, которое мы можем изменить, указав другие значения для свойства `globalCompositeOperation`. Оно может содержать одно из следующих строковых значений:

1. `"source-over"` — новая фигура накладывается на старую, перекрывая ее (значение по умолчанию);
2. `"source-atop"` — отображается только та часть новой фигуры, которая накладывается на старую; остальная часть новой фигуры не выводится. Старая фигура выводится целиком и находится ниже новой;
3. `"source-in"` — отображается только та часть новой фигуры, которая накладывается на старую. Остальные части новой и старой фигур не выводятся;
4. `"source-out"` — отображается только та часть новой фигуры, которая не накладывается на старую. Остальные части новой фигуры и вся старая фигура не выводятся;
5. `"destination-over"` — новая фигура перекрывается старой;
6. `"destination-atop"` — отображается только та часть старой фигуры, которая накладывается на новую; остальная часть старой фигуры не выводится. Новая фигура выводится целиком и находится ниже старой;
7. `"destination-in"` — отображается только та часть старой фигуры, на которую накладывается новая. Остальные части новой и старой фигур не выводятся;

8. "destination-out" — отображается только та часть старой фигуры, на которую не накладывается новая. Остальные части новой фигуры и вся старая фигура не выводятся;
9. "lighter" — цвета накладывающихся частей старой и новой фигур складываются, результирующий цвет получается более светлым;
10. "xor" — отображаются только те части старой и новой фигур, которые не накладываются друг на друга;
11. "copy" — выводится только новая фигура; все старые фигуры удаляются с холста.

Заданный нами способ наложения действует только для графики, которую мы нарисуем после этого. На уже нарисованную графику он не влияет.

1.4 Задание для самостоятельной работы

Запустите код, который рисует два накладывающихся прямоугольника разных цветов и позволяет изучать поведение холста при разных значениях свойства globalCompositeOperation. Измените значение этого свойства в соответствии с вашим вариантом (1 – 11, последующие вновь берут 1,2 и т.д. варианты), перезагрузите страницу нажатием клавиши <F5> и посмотрите, что получится.

Использование масок

Маской называется особая фигура, задающая своего рода "окно", сквозь которое будет видна часть графики, нарисованной на холсте. Вся графика, не попадающая в это "окно", будет скрыта; при этом сама маска на хост не выводится.

Ниже перечислены действия, необходимые для создания маски.

1. Рисуем сложный контур, который станет маской.
2. Обязательно делаем его замкнутым.
3. Вместо вызова методов stroke или fill вызываем метод clip. Этот метод не принимает параметров и не возвращает результат.
4. Рисуем графику, которая будет находиться под маской.

После этого нарисованная нами на шаге 4 графика будет частично видна сквозь маску.

Приведенный код рисует маску в виде треугольника, а потом рисует прямоугольник. Часть этого прямоугольника будет видна сквозь маску.

Работа с отдельными пикселами

Мы можем работать с отдельными пикселами графики, нарисованной на холсте. Это может пригодиться при создании очень сложного изображения или при наложении на графику специальных эффектов наподобие размытия.

Получение массива пикселов

Получить массив пикселов, представляющий нарисованную на холсте графику или ее фрагмент, позволяет метод getImageData(). Формат метода:

```
<контекст рисования>.getImageData(<x>, <y>, <Ширина>, <Высота>)
```

Первые два параметра указывают координату левого верхнего угла фрагмента нарисованной графики, два последних — его ширину и высоту.

Метод getImageData() возвращает объект класса ImageData, представляющий массив пикселов, которые составляют часть сцены с указанными нами параметрами:

```
const idSample = ctxCanvas.getImageData(200, 150, 100, 100);
```

Создание пустого массива пикселов

Чтобы самостоятельно нарисовать какое-либо изображение, можно сначала создать пустой массив пикселов, вызвав метод `createImageData()`:

```
<контекст рисования>.createImageData(<Ширина>, <Высота>)
```

Метод возвращает объект класса `ImageData`:

```
const idEmpty = ctxCanvas.createImageData(400, 300);
```

Здесь мы создаем пустой массив пикселов тех же размеров, что и сам холст.

Манипуляция пикселами

Теперь мы можем начать работать с отдельными пикселами полученного массива, задавая для них цвет и значения полупрозрачности и тем самым формируя какое-либо изображение или изменяя уже существующее.

Класс `ImageData` поддерживает свойство `data`. Его значением является объект-коллекция, хранящая набор чисел:

- первое число представляет собой долю красного цвета в цвете первого пиксела массива;
- второе число — долю зеленого цвета в цвете первого пиксела;
- третье число — долю синего цвета в цвете первого пиксела;
- четвертое число — степень полупрозрачности цвета первого пиксела;
- пятое число — долю красного цвета в цвете второго пиксела;
- шестое число — долю зеленого цвета в цвете второго пиксела;
- седьмое число — долю синего цвета в цвете второго пиксела;
- восьмое число — степень полупрозрачности цвета второго пиксела;
- ...
- n-3-е число — долю красного цвета в цвете последнего пиксела;
- n-2-е число — долю зеленого цвета в цвете последнего пиксела;
- n-1-е число — долю синего цвета в цвете последнего пиксела;
- n-е число — степень полупрозрачности цвета последнего пиксела.

Все значения, включая и степень полупрозрачности, должны укладываться в диапазон от 0 до 255. Для степени полупрозрачности значение 0 задает полную прозрачность, а 255 — полную непрозрачность.

Нумерация пикселов в массиве идет слева направо и сверху вниз, т. е. по строкам.

Поскольку набор чисел, хранящийся в свойстве `data` класса `ImageData`, представляет собой коллекцию, для доступа к отдельным значениям мы можем использовать тот же синтаксис, что и в случае обычных массивов. Также мы можем воспользоваться поддерживаемым всеми коллекциями свойством `length`, возвращающим размер коллекции.

Вывод массива пикселов

Завершив формирование нового изображения в массиве пикселов, мы можем вывести его на холст, вызвав метод `putImageData()`:

```
<контекст рисования>.putImageData(<Массив пикселов>, <x1>, <y1>[, <x2>, <y2>,<Ширина>, <Высота>])
```

Второй и третий параметры задают координату точки, где будет находиться левый верхний угол выводимого массива пикселов. Четвертый и пятый параметры задают координаты точки, где находится левый верхний угол фрагмента массива пикселов, который должен быть выведен на холст, а шестой и седьмой— ширину и высоту выводимого фрагмента. Если эти параметры не указаны, будет выведен весь массив пикселов:

```
ctxCanvas.putImageData(idEmpty, 0, 0);
```

В примере приведен код, рисующий на странице прямоугольник с градиентной заливкой. При этом цвета становятся все более и более прозрачными.