

# Лабораторная работа №1

## «Знакомство с контекстом рисования элемента <canvas>»

### Оглавление

Введение .....	1
Что такое <canvas>? .....	1
Содержимое файла 1.html .....	2
Содержимое файла 1.js .....	3
Получить ссылку на элемент <canvas> .....	3
Запросить контекст отображения двумерной графики .....	4
Нарисовать двумерное изображение, используя методы контекста .....	4
Вывод текста .....	6
Рисование сложных фигур .....	7
Как рисуются сложные фигуры .....	7
Перо. Перемещение пера .....	7
Прямые линии .....	7
Дуги .....	8
Кривые Безье .....	8
Прямоугольники .....	9
Задание стиля линий .....	9
Определение вхождения точки в состав контура .....	9
Задания для самостоятельной работы .....	10

### Введение

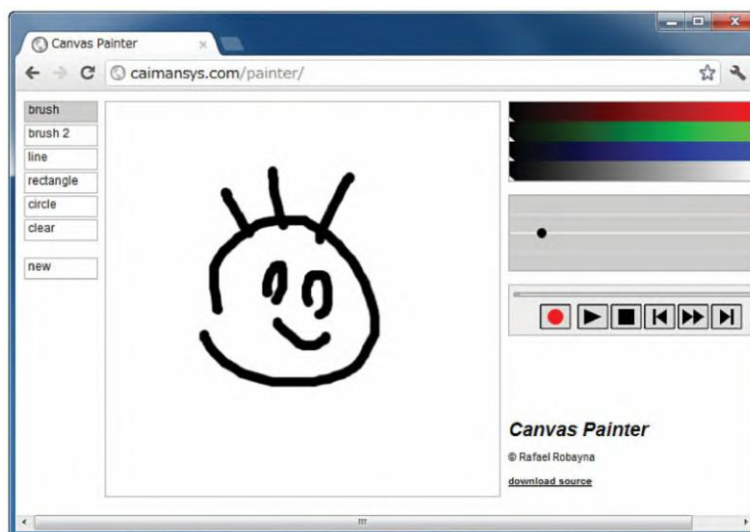
В лабораторной работе №1 мы познакомимся с элементом <canvas> исследовав процесс создания нескольких примеров программ.

### 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>`

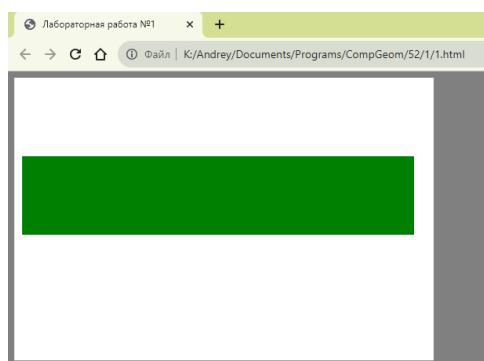
(<http://caimansys.com/painter/>)

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 300×150 пикселей. 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 двумерной графики.
3. Нарисовать двумерное 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>`.

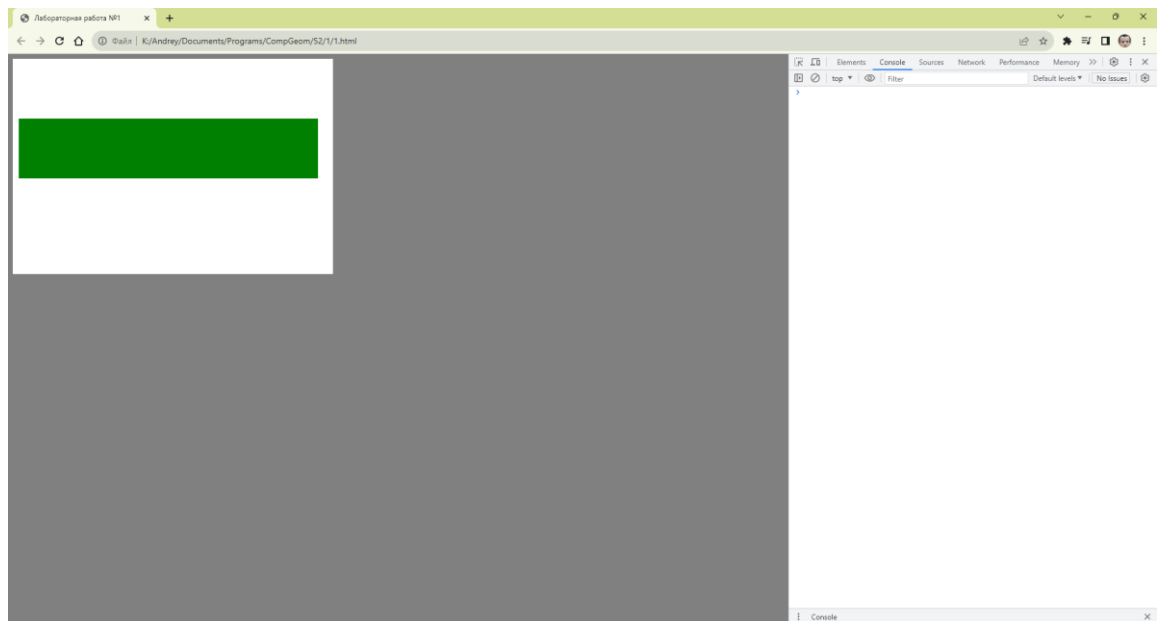


Рис. 3. Консоль в Chrome

## 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 нарисует, будут заполнены указанным цветом.

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

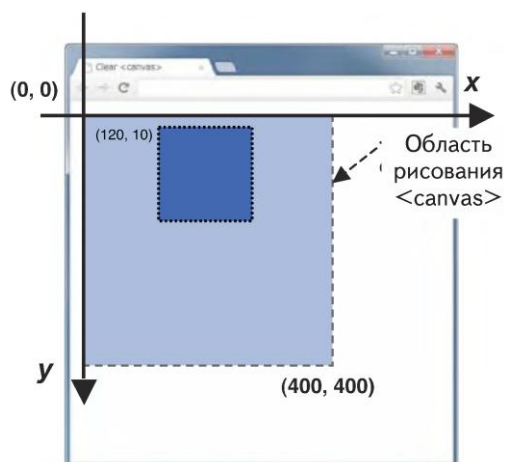


Рис. 4. Система координат элемента `<canvas>`

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

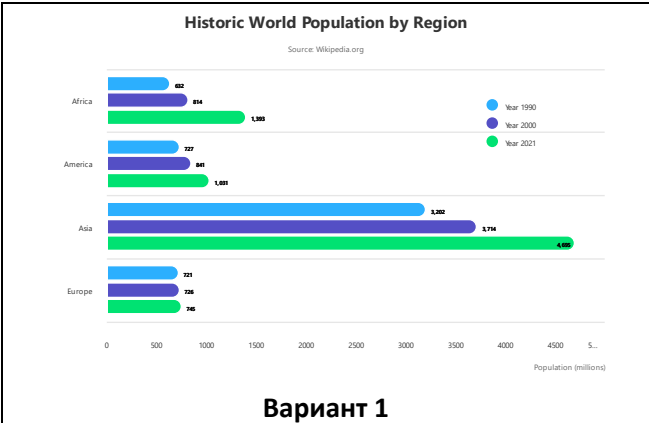
Иногда бывает необходимо выяснить, входит ли точка с заданными координатами в состав контура сложной фигуры. Это можно сделать с помощью метода `isPointInPath`:

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

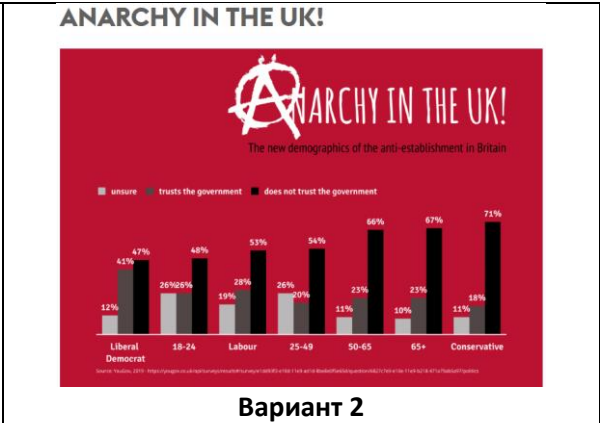
Обе координаты проверяемой точки указываются в виде чисел в пикселах. Метод возвращает `true`, если точка с такими координатами входит в состав контура, и `false` — в противном случае.

Задания для самостоятельной работы

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



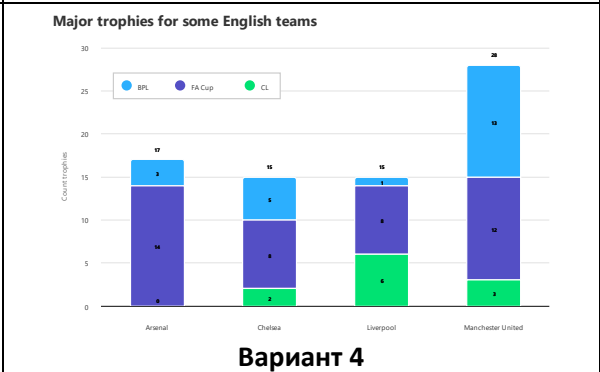
Вариант 1



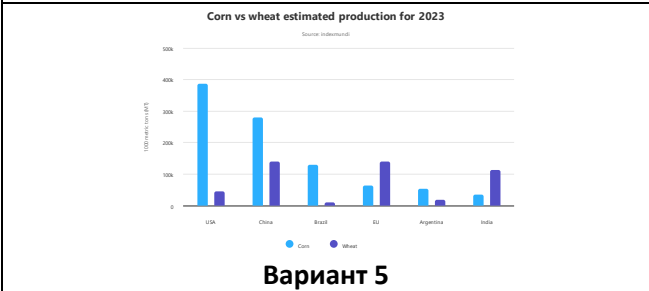
Вариант 2



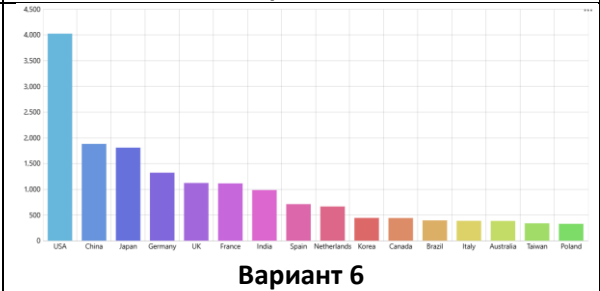
Вариант 3



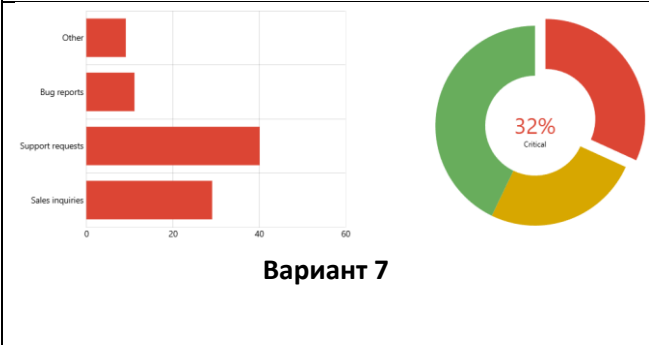
Вариант 4



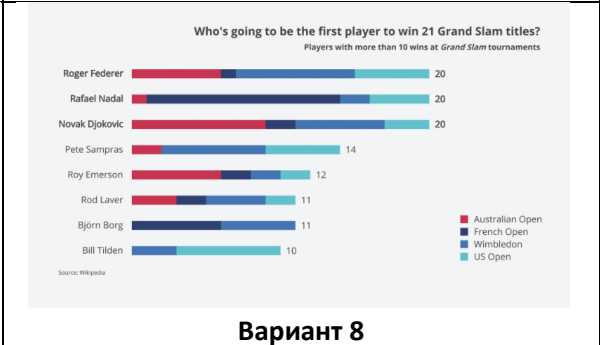
Вариант 5



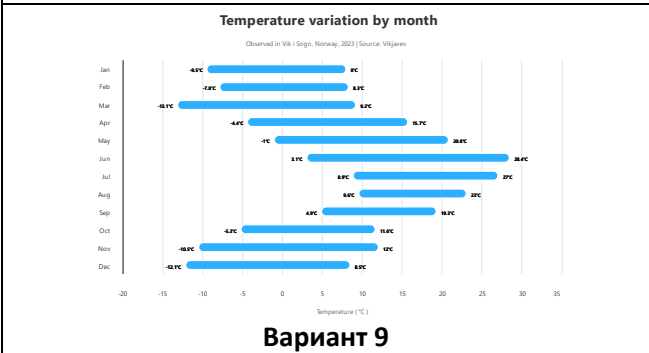
Вариант 6



Вариант 7



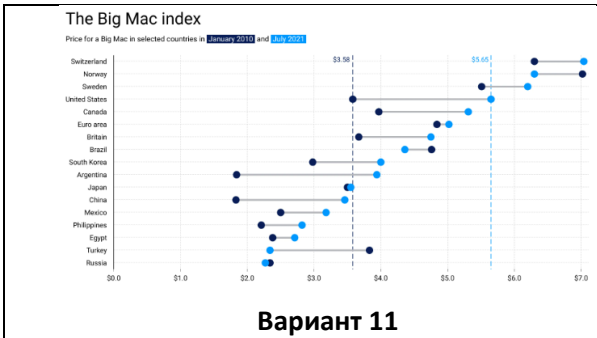
Вариант 8



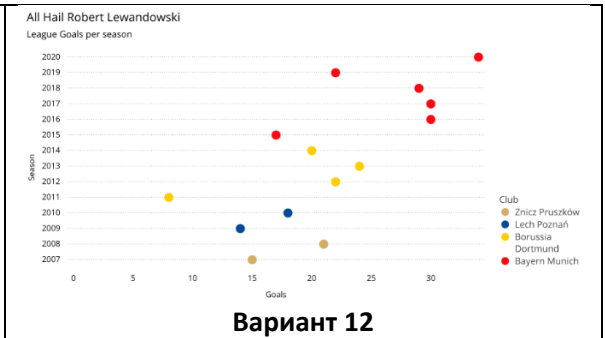
Вариант 9



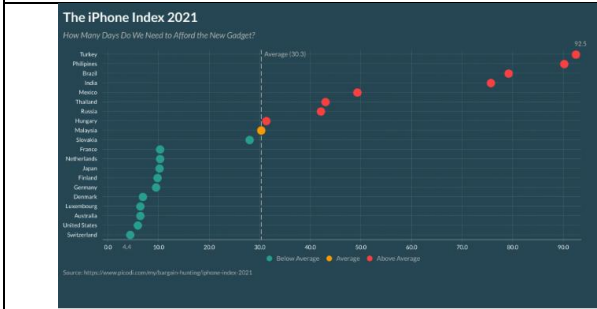
Вариант 10



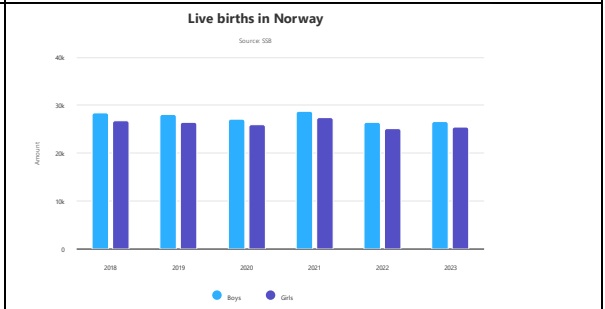
Вариант 11



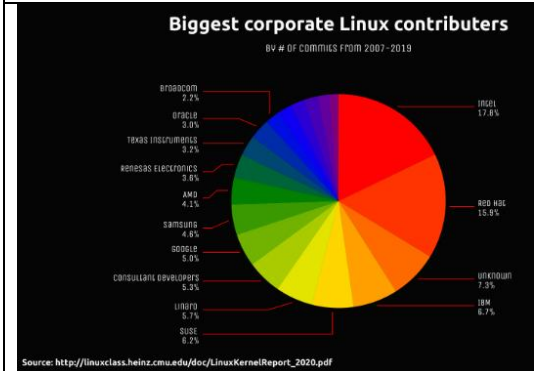
Вариант 12



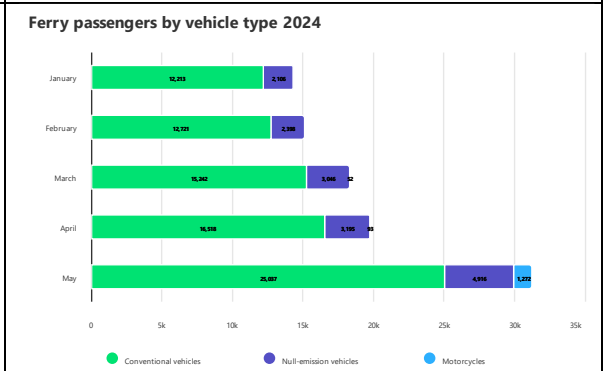
Вариант 13



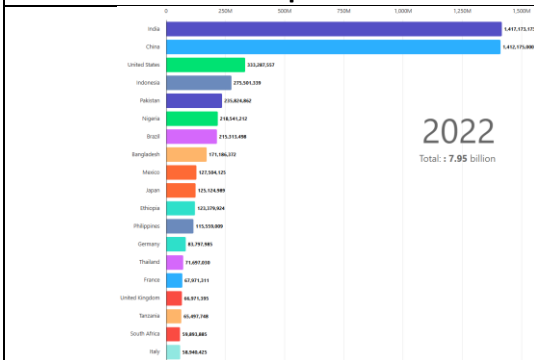
Вариант 14



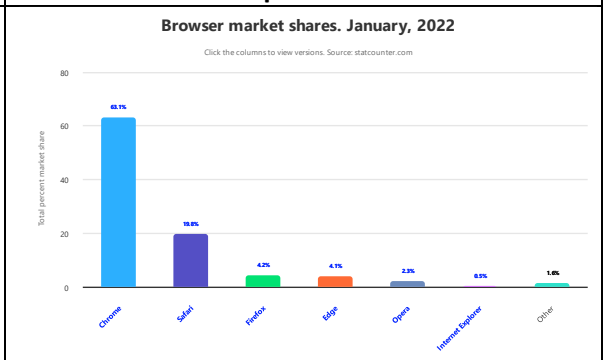
Вариант 15



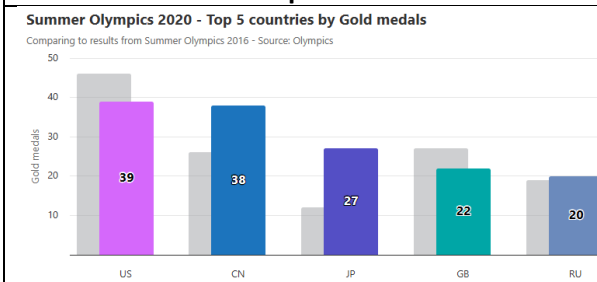
Вариант 16



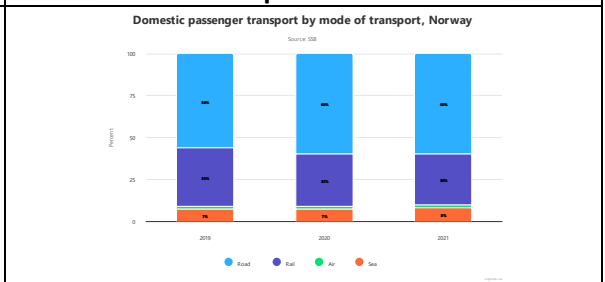
Вариант 17



Вариант 18



Вариант 19



Вариант 20

