

# РИСОВАНИЕ на HTML 5

## Технология работы с Canvas

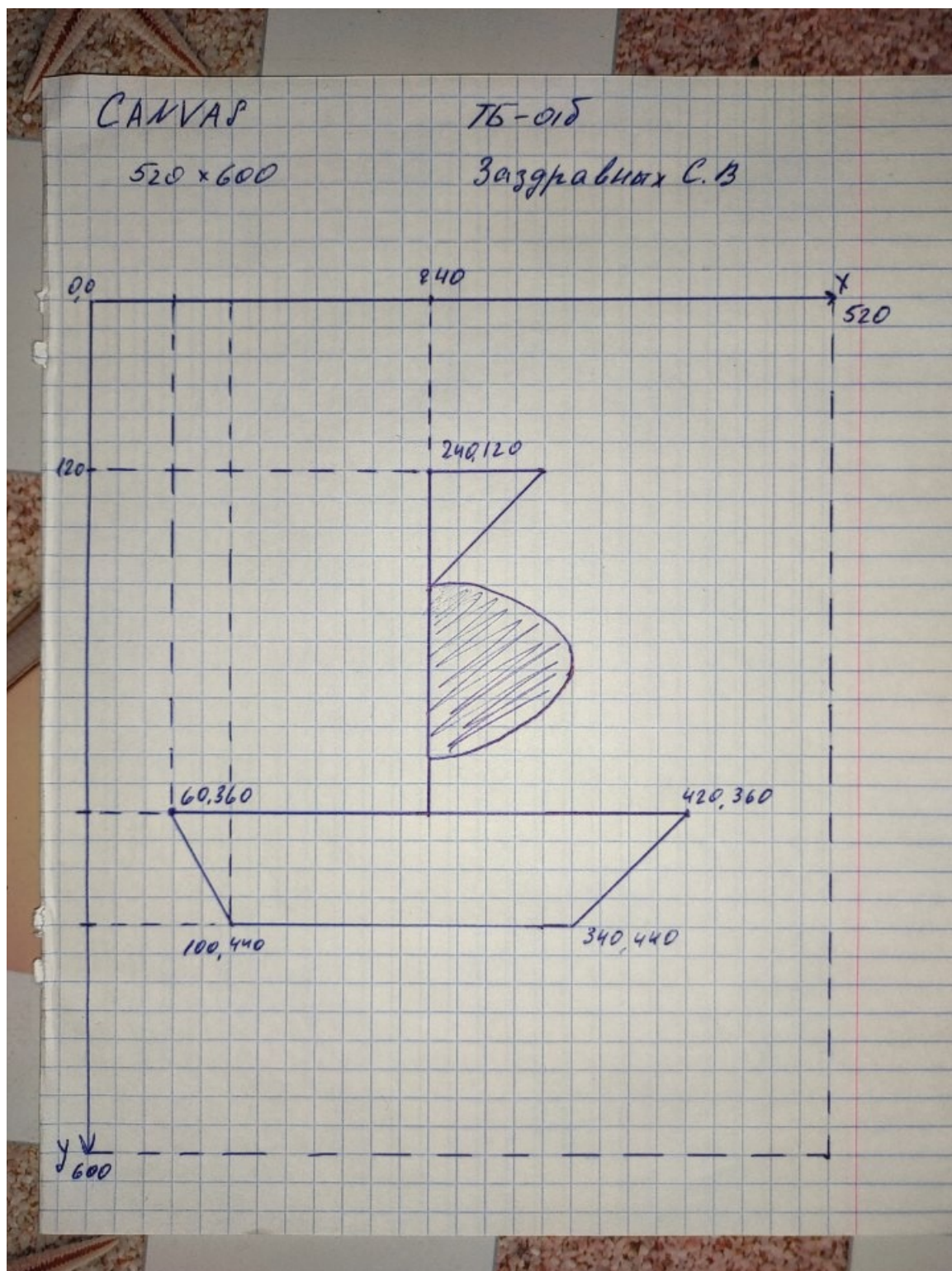
### Задание

1. Изучите приведенный ниже теоретический материал
2. Нарисуйте на бумаге ( на листе тетради) систему координат. Например, берём лист тетради «в клеточку», отступаем несколько клеточек сверху от края листа и проводим горизонтальную линию – это наша ось X. Отступаем слева от края листа несколько клеточек и проводим вертикальную линию – это наша ось Y.
3. По клеточкам аккуратно нарисуйте схематичное изображение некоторого узнаваемого по внешнему виду объекта (самолетик, машинка, игрушка и т. п.) **в натуральную величину**.
4. Решите, из каких графических примитивов можно построить картинку (отдельные линии, эллипсы/окружности, прямоугольники, дуги). Для каждого примитива надо найти базовые точки, по которым его можно построить.
5. Для каждой базовой точки надо рассчитать координаты X и Y **в пикселях**.
6. Напишите программу, которая строит картинку . **Требование к изображению: применить конструкции языка для построения линий, прямоугольников, дуг и окружностей разного цвета, использовать заливку и линии разной толщины.**

### РЕЗУЛЬТАТЫ РАБОТЫ

**1) фото рисунка на бумаге с координатами базовых точек в пикселях.**

Пример



## 2) HTML файл

Работа будет иметь продолжение – анимация объекта.

# ТЕОРИЯ

Для рисования на HTML 5 применяют специальный элемент **Canvas**.

С помощью Canvas можно рисовать как простейшие графические примитивы - линии, фигуры, текст, так и создавать сложные графические игры. В дополнение Canvas позволяет манипулировать изображениями и даже видео.

Определим на веб-странице элемент canvas:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Canvas в HTML5</title>
  </head>
  <body>
    <canvas id="myCanvas" width="300" height="200">

    </canvas>
  </body>
</html>
```

Также, как правило, для элемента canvas задаются ширина и высота с помощью атрибутов width и height. Если не указать эти атрибуты, то по умолчанию ширина будет составлять 300, а высота - 150 пикселей.

## Получение контекста рисования

Чтобы начать рисовать на canvas, нам надо получить его контекст:

```
var canvas = document.getElementById("myCanvas"),
    context = canvas.getContext("2d");
```

Для получения контекста используется функция `getContext("2d")`. В данном случае мы получаем двухмерный контекст для создания двухмерной графики. Также контекст элемента canvas позволяет создавать трехмерную графику. Подробнее об этом можно почитать в руководстве по [WebGL](#). В данном же случае мы затронем только двухмерную графику.

## Рисование прямоугольников

И теперь нам остается собственно нарисовать первую фигуру. Для рисования простейших фигур - прямоугольников нам могут понадобиться три метода:

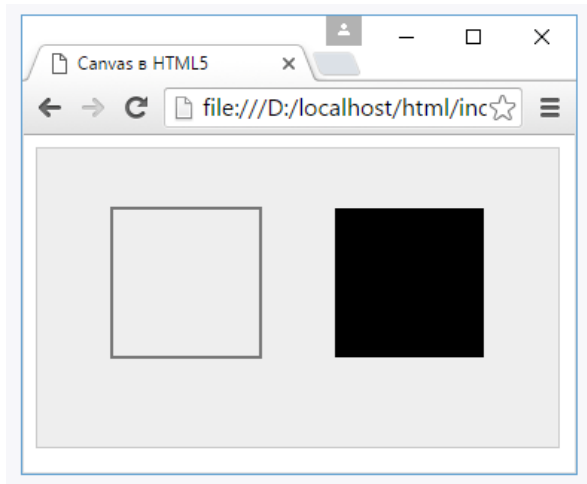
- `clearRect(x, y, w, h)`: очищает определенную прямоугольную область, верхний левый угол которой имеет координаты `x` и `y`, ширина равна `w`, а высота равна `h`
- `fillRect(x, y, w, h)`: заливает цветом прямоугольник, верхний левый угол которого имеет координаты `x` и `y`, ширина равна `w`, а высота равна `h`
- `strokeRect(x, y, w, h)`: рисует контур прямоугольника без заливки его каким-то определенным цветом

Теперь используем эти методы:

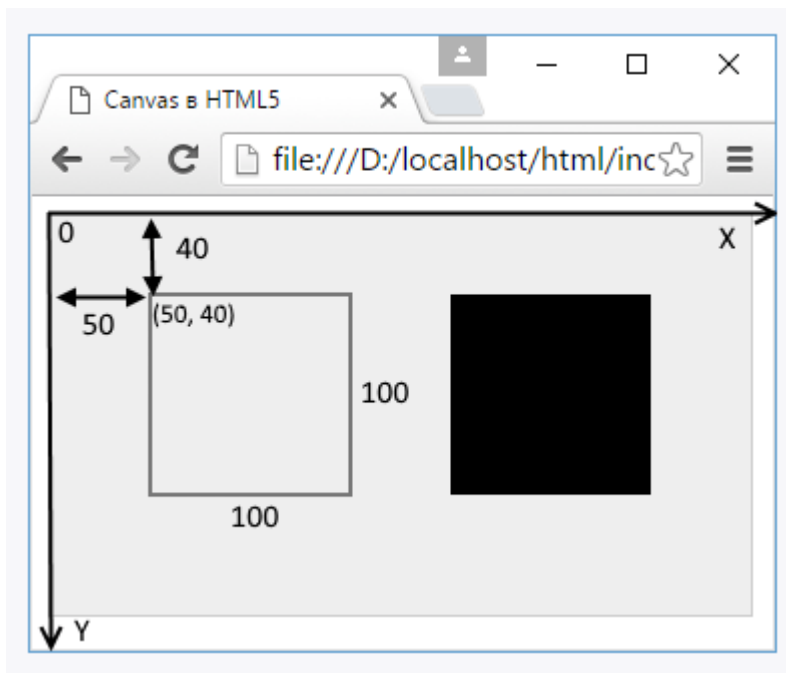
```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Canvas в HTML5</title>
  </head>
  <body>
    <canvas id="myCanvas" width="350" height="200"
      style="background-color:#eee; border: 1px solid #ccc;">

    </canvas>
    <script>
      var canvas = document.getElementById("myCanvas"),
          context = canvas.getContext("2d");

      context.strokeRect(50, 40, 100, 100);
      context.fillRect(200, 40, 100, 100);
    </script>
  </body>
</html>
```



При рисовании в функции `strokeRect` и `fillRect` передаются координаты `x` и `y` верхнего левого угла прямоугольника относительно элемента `canvas`, а также ширина и высота прямоугольника. При этом началом координат считается верхний левый угол элемента `canvas`, ось `X` направлена вправо, а ось `Y` направлена вниз:



### Система координат

Изображение на экране строится из отдельных светящихся точек (пиксель – pixel - picture element). Каждая точка канвы имеет координаты **X** и **Y**. Система координат имеет **начало в левом верхнем углу канвы**.



Координаты точек задаются в пикселях. Количество пикселей не может быть дробным и отрицательным, поэтому и **все координаты задаются целыми положительными числами**.

### Как рассчитать координаты точек для графического изображения

- 1) Надо нарисовать на бумаге ( на листе тетради) систему координат. Например, берём лист тетради «в клеточку», отступаем несколько клеточек сверху от края листа и проводим горизонтальную линию – это наша ось X. Отступаем слева от края листа несколько клеточек и проводим вертикальную линию – это наша ось Y.
- 2) По клеточкам аккуратно рисуем будущее изображение **в натуральную величину**.
- 3) Решаем, из каких графических примитивов можно построить нашу картинку (отдельные линии, эллипсы/окружности, прямоугольники, дуги). Для каждого примитива надо найти базовые точки, по которым его можно построить.

- 4) Для каждой базовой точки надо рассчитать координаты по X и по Y в пикселях. Для этого в нашей системе координат надо определить расстояние от начала координат (на бумаге) до изображения точки в миллиметрах (1 клеточка = 5 мм).
- 5) На каждом миллиметре изображения на экране располагается примерно 4 пикселя. Поэтому, чтобы **получить координаты точки в пикселях надо умножить каждую координату в миллиметрах на 4.**

## Настройка рисования

Контекст элемента canvas предоставляет ряд свойств, с помощью которых можно настроить отрисовку на canvas. К подобным свойствам относятся следующие:

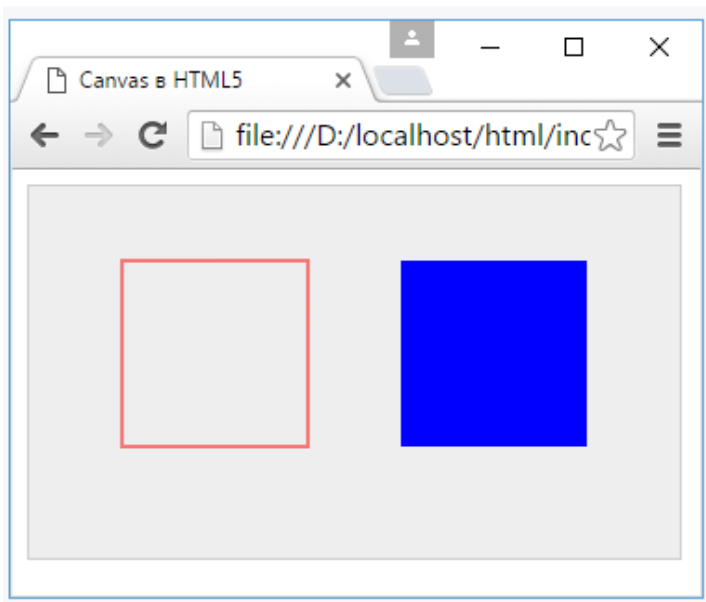
- `strokeStyle`: устанавливает цвет линий или цвет контура. По умолчанию установлен черный цвет
- `fillStyle`: устанавливает цвет заполнения фигур. По умолчанию установлен черный цвет
- `lineWidth`: устанавливает толщину линий. По умолчанию равно 1.0
- `lineJoin`: устанавливает стиль соединения линий
- `globalAlpha`: устанавливает прозрачность отрисовки на canvas
- `setLineDash`: создает линию из коротких черточек

Чтобы рисовать какие-то видимые фигуры, нам нужно задать цвет. Установить цвет можно разными способами. Во-первых, мы можем задать цвет контура или границы фигур с помощью свойства `strokeStyle`:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Canvas в HTML5</title>
  </head>
  <body>
    <canvas id="myCanvas" width="350" height="200"
      style="background-color:#eee; border: 1px solid #ccc;">
      Ваш браузер не поддерживает Canvas
    </canvas>
    <script>
      var canvas = document.getElementById("myCanvas"),
          context = canvas.getContext("2d");

      context.strokeStyle = "red";
      context.fillStyle = "blue";

      context.strokeRect(50, 40, 100, 100);
      context.fillRect(200, 40, 100, 100);
    </script>
  </body>
</html>
```



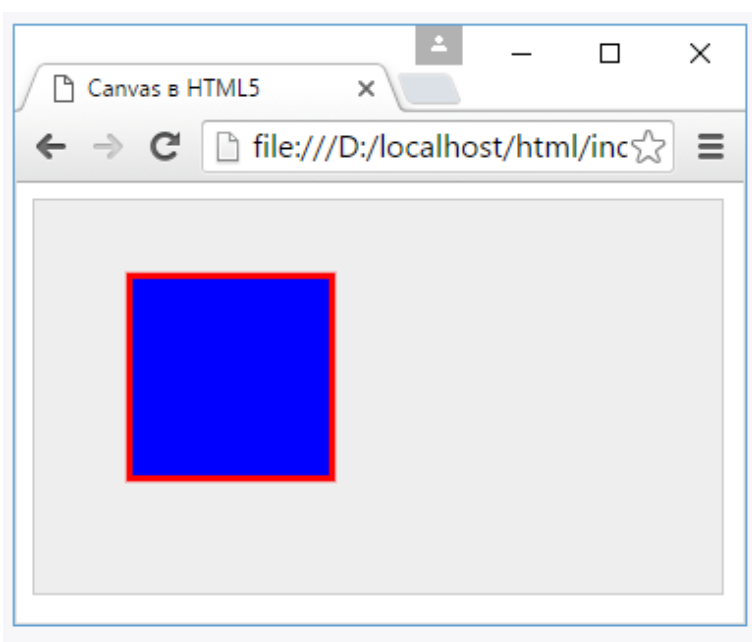
В качестве значения свойства `strokeStyle` и `fillStyle` получают название цвета в виде строки, либо в виде шестнадцатичного значения цвета (например, `"#00FFFF"`), либо в виде значений `rgb` (`"rgb(0, 0, 255)"`) и `rgba` (`"rgba(0, 0, 255, 0.5)"`).

Свойство `lineWidth` позволяет установить толщину линии:

```
var canvas = document.getElementById("myCanvas"),
    context = canvas.getContext("2d");

context.strokeStyle = "red";
context.fillStyle = "blue";
context.lineWidth = 6.5;

context.strokeRect(50, 40, 100, 100);
context.fillRect(50, 40, 100, 100);
```



Метод `setLineDash()` в качестве параметра принимает массив чисел, которые

устанавливают расстояния между линиями. Например:

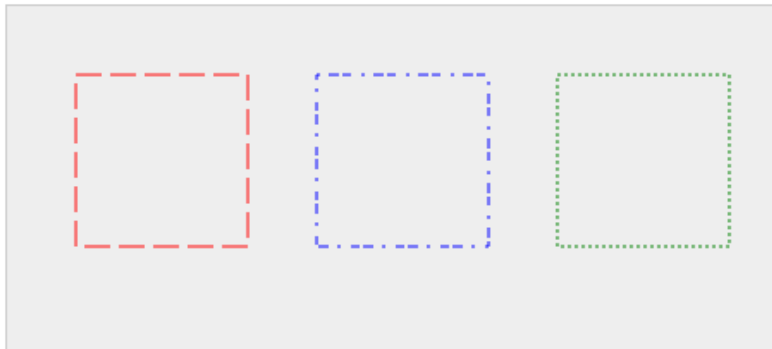
```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Canvas в HTML5</title>
  </head>
  <body>
    <canvas id="myCanvas" width="450" height="200"
      style="background-color:#eee; border: 1px solid #ccc;">
      Ваш браузер не поддерживает Canvas
    </canvas>
    <script>
      var canvas = document.getElementById("myCanvas"),
          context = canvas.getContext("2d");

      context.strokeStyle = "red";

      context.setLineDash([15,5]);
      context.strokeRect(40, 40, 100, 100);

      context.strokeStyle = "blue";
      context.setLineDash([2,5,6]);
      context.strokeRect(180, 40, 100, 100);

      context.strokeStyle = "green";
      context.setLineDash([2]);
      context.strokeRect(320, 40, 100, 100);
    </script>
  </body>
</html>
```



## Рисование текста

Наряду с геометрическими фигурами и изображениями canvas позволяет выводить текст. Для этого вначале надо установить у контекста canvas свойство font:

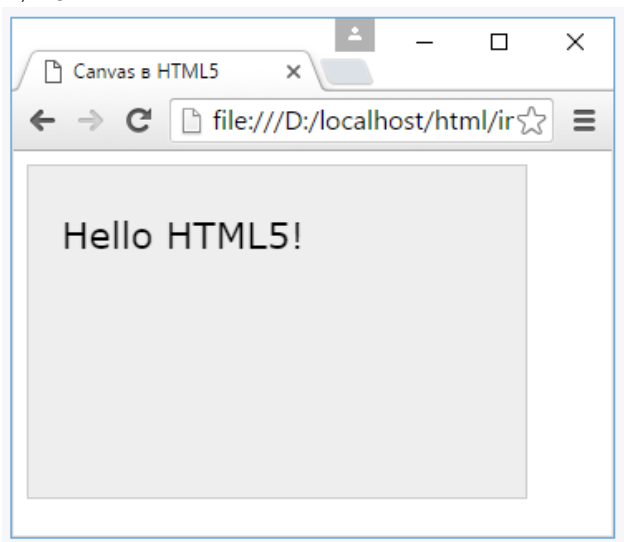
```
1 var canvas = document.getElementById("myCanvas"),
2   context = canvas.getContext("2d");
3 context.font = "22px Verdana";
```



Свойство `font` в качестве значения принимает определение шрифта. В данном случае это шрифт Verdana высотой 22 пикселя. В качестве шрифтов используются стандартные шрифты.

Далее мы можем вывести некоторый текст с помощью метода `fillText()`:

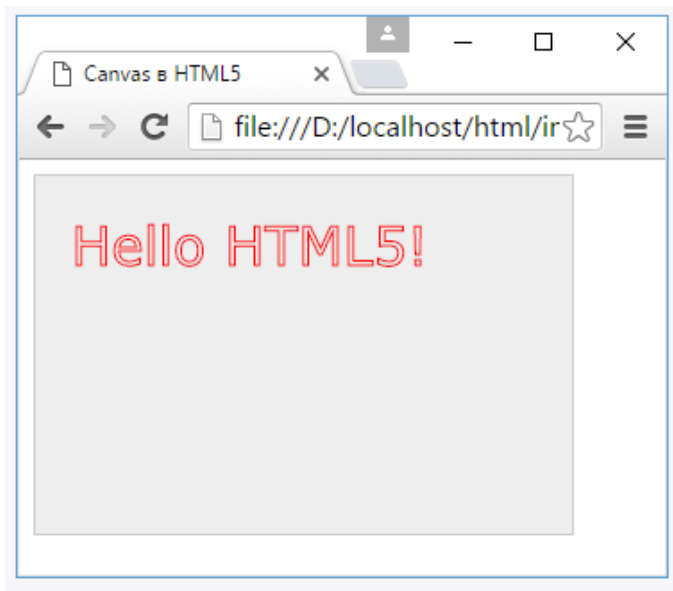
```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Canvas в HTML5</title>
  </head>
  <body>
    <canvas id="myCanvas" width="300" height="200"
      style="background-color:#eee; border:1px solid #ccc;">
      Ваш браузер не поддерживает Canvas
    </canvas>
    <script>
      var canvas = document.getElementById("myCanvas"),
          context = canvas.getContext("2d");
      context.font = "22px Verdana";
      context.fillText("Hello HTML5!", 20, 50);
    </script>
  </body>
</html>
```



Метод `fillText(text, x, y)` принимает три параметра: выводимый текст и `x` и `y` координаты точки, с которой выводится текст.

Для вывода текста можно также применять метод `strokeText()`, который создает границу для выводимых символов:

```
1 var canvas = document.getElementById("myCanvas"),
2   context = canvas.getContext("2d");
3 context.font = "30px Verdana";
4 context.strokeStyle = "red";
5 context.strokeText("Hello HTML5!", 20, 50);
```



## Рисование фигур

Кроме прямоугольников canvas позволяет рисовать и более сложные фигуры. Для оформления сложных фигур используется концепция геометрических путей, которые представляют набор линий, окружностей, прямоугольников и других более мелких деталей, необходимых для построения сложной фигуры.

Для создания нового пути надо вызвать метод `beginPath()`, а после завершения пути вызывается метод `closePath()`:

```
var canvas = document.getElementById("myCanvas"),
    context = canvas.getContext("2d");
context.beginPath();
// здесь инструкции по созданию фигур
context.closePath();
```

Между вызовами методов `beginPath()` и `closePath()` находятся методы, непосредственно создающие различные участки пути.

### Методы `moveTo()` и `lineTo()`

Для начала рисования пути нам надо зафиксировать начальную точку этого пути. Это можно сделать с помощью метода `moveTo()`, который имеет следующее определение:

```
1moveTo(x, y)
```

Метод перемещает нас на точку с координатами `x` и `y`.

Метод `lineTo()` рисует линию. Он имеет похожее определение:

```
1lineTo(x, y)
```

Метод рисует линию от текущей позиции до точки с координатами `x` и `y`.

Теперь нарисуем ряд линий:

```

var canvas = document.getElementById("myCanvas"),
    context = canvas.getContext("2d");
context.beginPath();
context.moveTo(30, 20);
context.lineTo(100, 80);
context.lineTo(150, 30);
context.closePath();

```

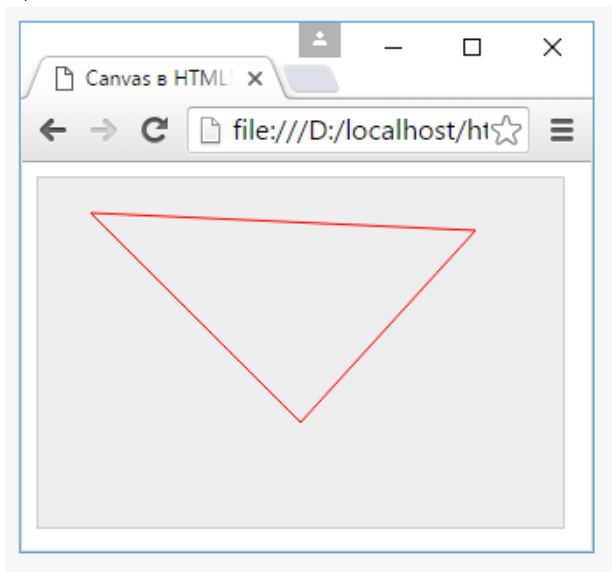
Здесь мы устанавливаем начало пути в точку (30, 20), затем от нее рисуем линию до точки (100, 80) и далее рисуем еще одну линию до точки (150, 30).

Хотя мы нарисовали несколько линий, пока мы их не увидим, потому что их надо отобразить на экране. Для отображения пути надо использовать метод `stroke()`:

```

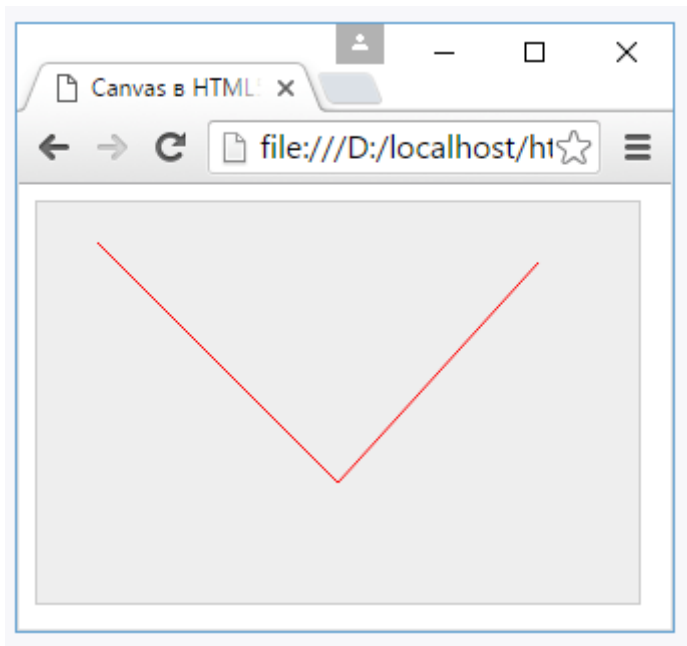
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Canvas в HTML5</title>
  </head>
  <body>
    <canvas id="myCanvas" width="300" height="200"
      style="background-color:#eee; border:1px solid #ccc;">
      Ваш браузер не поддерживает Canvas
    </canvas>
    <script>
      var canvas = document.getElementById("myCanvas"),
          context = canvas.getContext("2d");
      context.beginPath();
      context.moveTo(30, 20);
      context.lineTo(150, 140);
      context.lineTo(250, 30);
      context.closePath();
      context.strokeStyle = "red";
      context.stroke();
    </script>
  </body>
</html>

```



Хотя мы нарисовали все две линии, но по факту мы увидим три линии, которые оформляют треугольник. Дело в том, что вызов метода `context.closePath()` завершает путь, соединяя последнюю точку с первой. И в результате образуется замкнутый контур.

Если нам не надо замыкать путь, то мы можем удалить вызов метода `context.closePath()`:



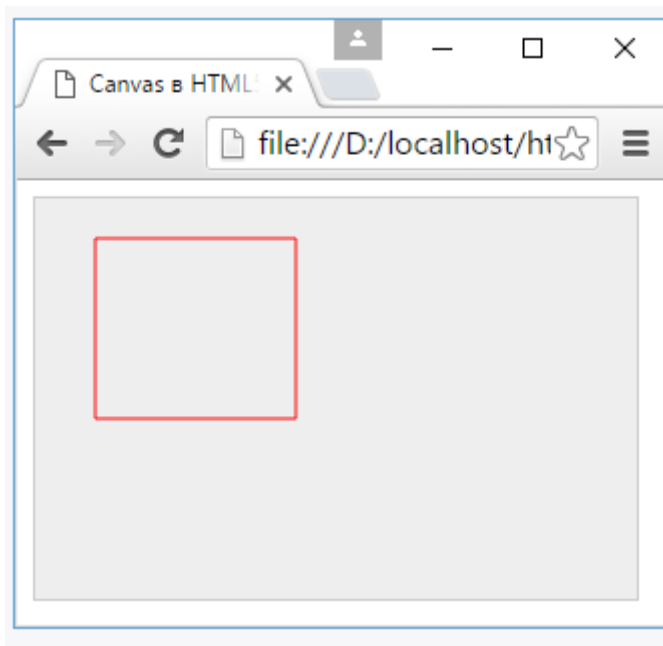
## Метод `rect`

Метод `rect()` создает прямоугольник. Он имеет следующее определение:

```
1 rect(x, y, width, height)
```

Где `x` и `y` - это координаты верхнего левого угла прямоугольника относительно `canvas`, а `width` и `height` - соответственно ширина и высота прямоугольника. Нарисуем, к примеру, следующий прямоугольник:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Canvas в HTML5</title>
  </head>
  <body>
    <canvas id="myCanvas" width="300" height="200"
      style="background-color:#eee; border:1px solid #ccc;">
      Ваш браузер не поддерживает Canvas
    </canvas>
    <script>
      var canvas = document.getElementById("myCanvas"),
          context = canvas.getContext("2d");
      context.beginPath();
      context.rect(30, 20, 100, 90);
      context.closePath();
      context.strokeStyle = "red";
      context.stroke();
    </script>
  </body>
</html>
```



Стоит отметить, что такой же прямоугольник мы могли бы создать из линий:

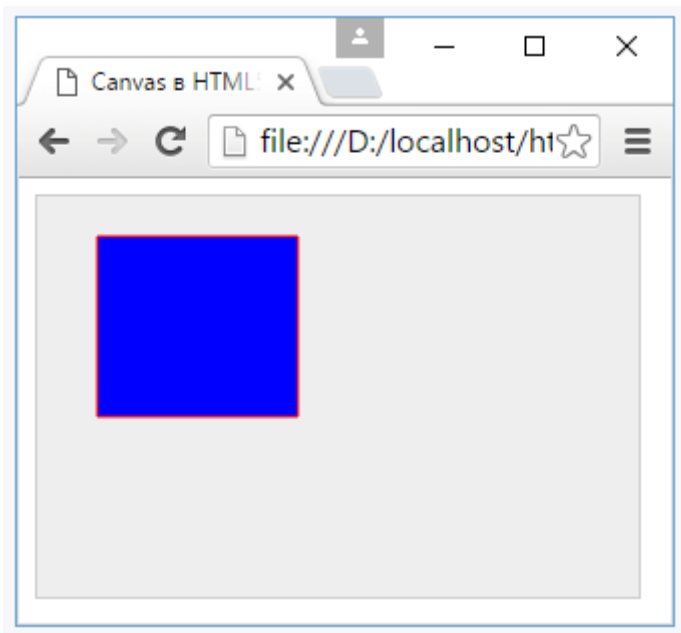
```
var canvas = document.getElementById("myCanvas"),
    context = canvas.getContext("2d");
context.beginPath();
context.moveTo(30, 20);
context.lineTo(130, 20);
context.lineTo(130, 110);
context.lineTo(30, 110);
context.closePath();
context.strokeStyle = "red";
context.stroke();
```

## Метод fill()

Метод fill() заполняет цветом все внутреннее пространство нарисованного пути:

```
var canvas = document.getElementById("myCanvas"),
    context = canvas.getContext("2d");
context.beginPath();
context.rect(30, 20, 100, 90);
context.closePath();
context.strokeStyle = "red";
context.fillStyle = "blue";
context.fill();
context.stroke();
```

С помощью свойства `fillStyle` опять же можно задать цвет заполнения фигуры. В данном случае это синий цвет.



## Метод arc()

Метод `arc()` добавляет к пути участок окружности или арку. Он имеет следующее определение:

```
1 arc(x, y, radius, startAngle, endAngle, anticlockwise)
```

Здесь используются следующие параметры:

- `x` и `y`: X- и y-координаты, в которых начинается арка
- `radius`: радиус окружности, по которой создается арка
- `startAngle` и `endAngle`: начальный и конечный угол, которые отсекают окружность до арки. В качестве единицы измерения для углов применяются радианы. Например, полная окружность - это  $2\pi$  радиан. Если, к примеру, нам надо нарисовать полный круг, то для параметра `endAngle` можно указать значение  $2\pi$ . В JavaScript эту величину можно получить с помощью выражения `Math.PI * 2`.
- `anticlockwise`: направление движения по окружности при отсечении ее части, ограниченной начальным и конечным углом. При значении `true` направление против часовой стрелки, а при значении `false` - по часовой стрелке.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Canvas в HTML5</title>
  </head>
  <body>
    <canvas id="myCanvas" width="420" height="200"
      style="background-color:#eee; border:1px solid #ccc;">
      Ваш браузер не поддерживает Canvas
    </canvas>
    <script>
      var canvas = document.getElementById("myCanvas"),
          context = canvas.getContext("2d");
```

```

context.strokeStyle = "red";

context.beginPath();
context.moveTo(20, 90);
context.arc(20, 90, 50, 0, Math.PI/2, false);
context.closePath();
context.stroke();

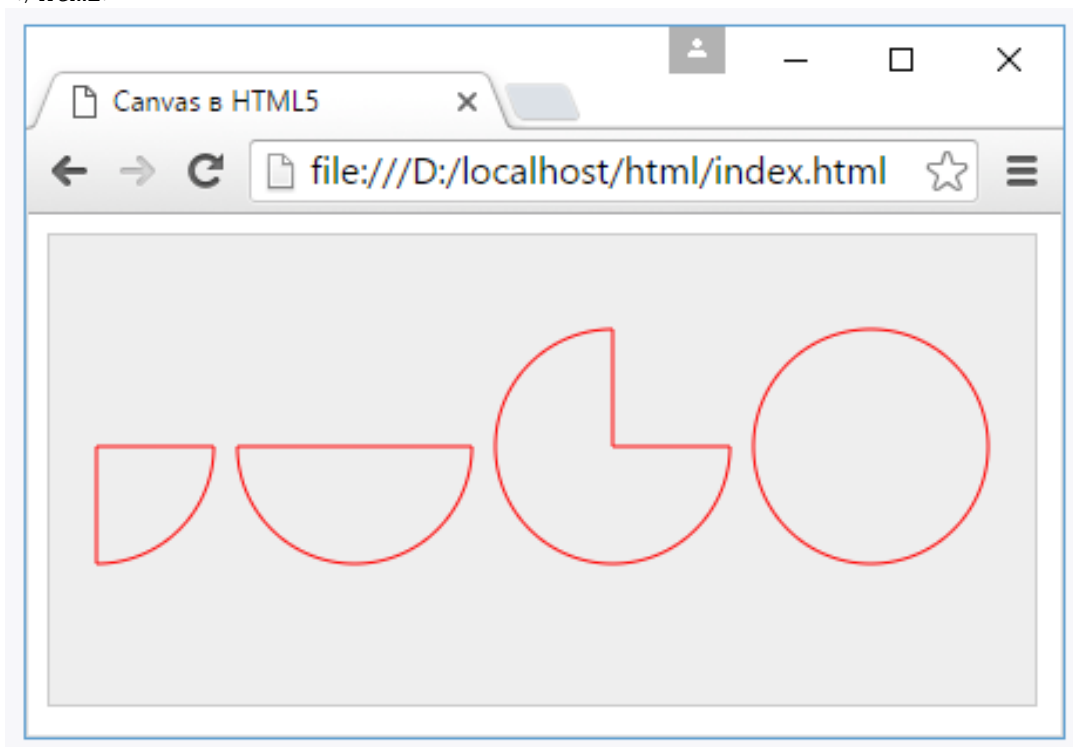
context.beginPath();
context.moveTo(130, 90);
context.arc(130, 90, 50, 0, Math.PI, false);
context.closePath();
context.stroke();

context.beginPath();
context.moveTo(240, 90);
context.arc(240, 90, 50, 0, Math.PI * 3 / 2, false);
context.closePath();
context.stroke();

context.beginPath();
context.arc(350, 90, 50, 0, Math.PI*2, false);
context.closePath();
context.stroke();

</script>
</body>
</html>

```



Последний параметр `anticlockwise` играет важную роль, так как определяет движение по окружности, и в случае изменения `true` на `false` и наоборот, мы можем получить совершенно разные фигуры:

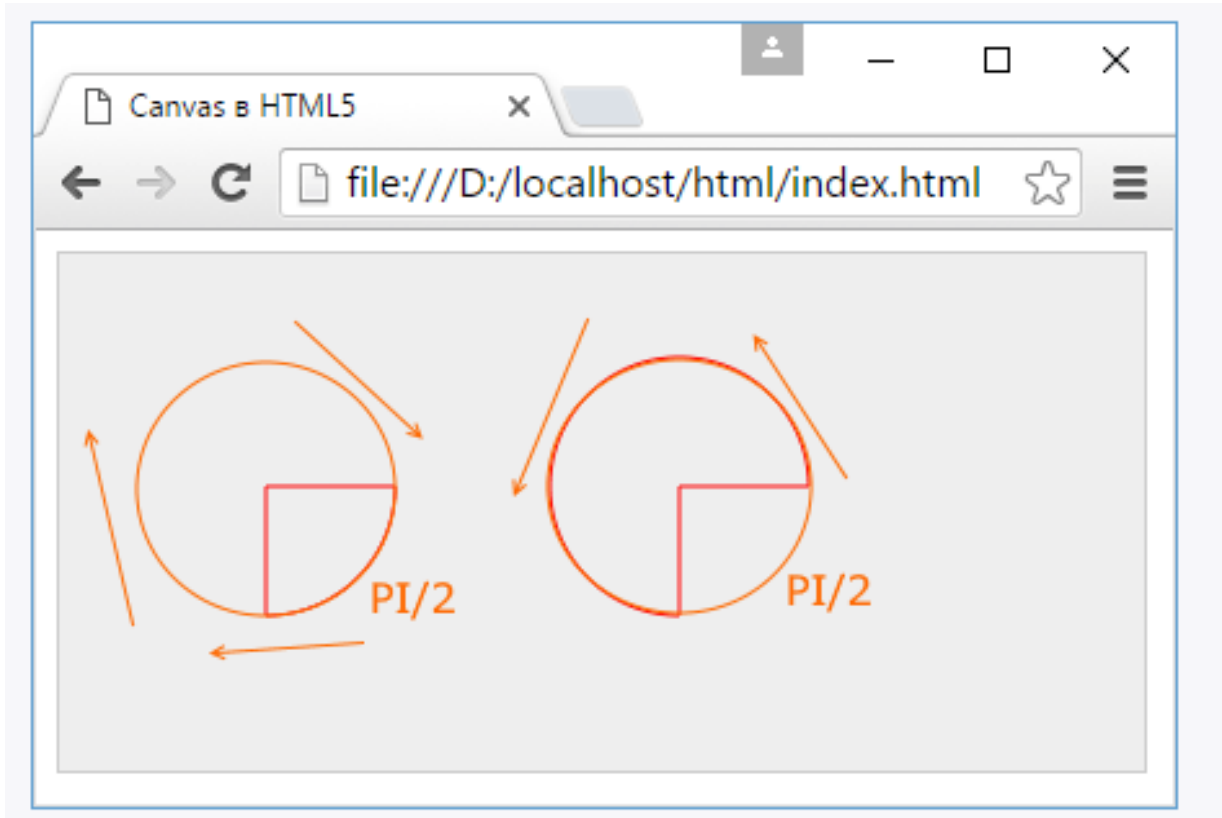
```

var canvas = document.getElementById("myCanvas"),
    context = canvas.getContext("2d");
context.strokeStyle = "red";

```

```
context.beginPath();
context.moveTo(80, 90);
context.arc(80, 90, 50, 0, Math.PI/2, false);
context.closePath();
context.stroke();
```

```
context.beginPath();
context.moveTo(240, 90);
context.arc(240, 90, 50, 0, Math.PI/2, true);
context.closePath();
context.stroke();
```



## Метод arcTo()

Метод arcTo() также рисует дугу. Он имеет следующее определение:

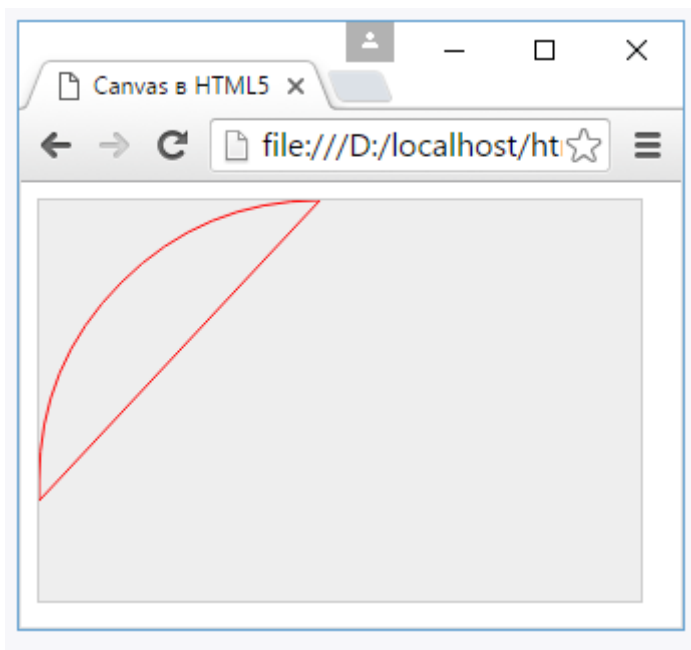
```
arcTo(x1, y1, x2, y2, radius)
```

Где  $x_1$  и  $y_1$  - координаты первой контрольной точки,  $x_2$  и  $y_2$  - координаты второй контрольной точки, а  $radius$  - радиус дуги.

```
var canvas = document.getElementById("myCanvas"),
    context = canvas.getContext("2d");
context.strokeStyle = "red";
```

```
context.beginPath();
context.moveTo(0, 150);
context.arcTo(0, 0, 150, 0, 140);
context.closePath();
context.stroke();
```





здесь мы перемещаемся вначале на точку  $(0, 150)$ , и от этой точки до первой контрольной точки  $(0, 0)$  будет проходить первая касательная. Далее от первой контрольной точки  $(0, 0)$  до второй  $(150, 0)$  будет проходить вторая касательная. Эти две касательные оформляют дугу, а 140 служит радиусом окружности, на которой отсекается дуга.