**Lesson 8. Introducing Canvas**

Objective

Get to know the concept of canvas, Canvas in JavaScript. Draw emoticon using in Canvas.

Contents

[Introduction 1](#_Toc72753030)

[Index.html file 2](#_Toc72753031)

[Script.js file 3](#_Toc72753032)

[Setting up the canvas and context 3](#_Toc72753033)

[Rectangles, working with color 3](#_Toc72753034)

[Drawing lines, getting to know Path () 6](#_Toc72753035)

[Curves 7](#_Toc72753036)

[Summing up and drawing a smiley 10](#_Toc72753037)

[**Hometasks** 12](#_Toc72753038)

What we repeat

1. Basic tags
2. Differences between vector and raster graphics

What's new

1. Canvas basics
2. Working with rectangles - rect
3. Drawing curves (beginPath, closePath, etc.)

Links to materials and personal account

[Working materials](https://hwschool.bitrix24.ru/bitrix/tools/disk/focus.php?folderId=532365&action=openFolderList&ncc=1)(for the teacher).

[Materials (edit)](https://hwschool.bitrix24.ru/~lNlQF)(we send this link to the student at the beginning of the lesson).

Methodical material

Introduction

At the very beginning of learning **SVG**, we talked about two types of graphics - vector and raster. Do you remember how they differ?

1) Raster graphics - pictures are composed of colored dots (pixels). It is possible to make images of any complexity, but at the same time they weigh quite a lot and dots are noticeable when magnified.

2) Vector graphics - pictures from consist of geometric shapes (lines, circles, rectangles, etc.). They are lightweight, do not lose quality when enlarged, but using shapes it is impossible to make complex images.

Today we will take a look at the Canvas. This is an HTML5 tag, inside of which drawing is done using JavaScript.

Index.html file

We will write a standard template from the main tags, while do not forget to include scripts, CSS will not be useful to us today.

**<! DOCTYPE html>**

**<html>**

**<head>**

**<meta charset = "utf-8">**

**<title> Canvas </title>**

**</head>**

**<body>**

**<script src = "script.js"> </script>**

**</body>**

**</html>**

Now let's create a tag with the same name inside - canvas, specify its width and height and an identifier **(id)**

If the student is interested, then you can check that if you do not specify the height and width, then by default the canvas is 300 by 150.

**<! DOCTYPE html>**

**<html>**

**<head>**

**<meta charset = "utf-8">**

**<title> Canvas </title>**

**</head>**

**<body>**

**<canvas id = "canvas" width = "400" height = "400"> </canvas>**

**<script src = "script.js"> </script>**

**</body>**

**</html>**

Nothing appeared in the browser, but let's see the element code right away.

Graphical user interface, application

Description automatically generated

Script.js file

Setting up the canvas and context

Now we need to tell how our images should be drawn on the canvas, that is, how to render them. Perhaps you have heard this word in photo processing or video editing, if not, then we will assume that this is the transformation of something that we have done into an image or just a finished file.

Back to practice, we will be setting up the canvas to work with 2d graphics. Let's create a variable and pull out the tag by id. It is often called **ctx**, and we will pull it out using the **getElementById** method

**let ctx = document.getElementById ('canvas')**

This method is similar to **querySelector**, and it works in a similar way, but it searches only by **id**. In practice it can be found in older code, so you don't need to be intimidated by it. Let's configure our canvas to work in **2D** using the **getContext** function.

**let ctx = document.getElementById ('canvas'). getContext ('2d');**

As we already know **get** means get, and **context** is the content.

Rectangles, working with color

Let's learn how to draw rectangles inside the canvas, here we have 3 types of them

* fillRect - filled rectangle
* strokeRect - a rectangle with no fill (outline)
* clearReact - empty rectangle

Let's create a large rectangle for the whole canvas using fillRect, this method has 4 parameters

**fillRect (x0, y0, width, height)**

Note that our coordinate system is the same as in SVG.

Rectangle

Description automatically generated with low confidence

**let ctx = document.getElementById ('canvas'). getContext ('2d');  
ctx.fillRect (0,0,400,400)**

Shape, rectangle

Description automatically generated

Here is our rectangle, by default it is black, but we can change its color. The logic of working with canvas is built on live drawing, in order to draw a drawing of a different color, we must take a pencil of a different color before we start drawing. For this we have the **fillStyle** function

**let ctx = document.getElementById ('canvas'). getContext ('2d');**

**ctx.fillStyle = "# 7D95A4"**

**ctx.fillRect (0,0,400,400)**

We will also create a frame, for this we call the strokeRect (x0, y0, width, height) function which draws a rectangle without filling

**let ctx = document.getElementById ('canvas'). getContext ('2d');**

**ctx.fillStyle = "# 7D95A4"**

**ctx.fillRect (0,0,400,400)**

**ctx.strokeRect (0,0,400,300)**

Shape, square

Description automatically generated

There is a frame, but it is not very clearly visible, so I propose to start creating it not from the very origin, but, for example, from point (5; 5)

**ctx.strokeRect(5,5,390,290)**

Border color changes similar to fill color

**ctx.strokeStyle = "# DFEC14"**

**ctx.strokeRect (5,5,390,390)**

Shape, square

Description automatically generated

Let's also look at the empty rectangle, it is needed to clear any areas from the fill. I suggest creating a small one in the center

**ctx.strokeRect (5,5,390,390)**

**ctx.clearRect (200,200,50,50)**

Not quite centered, because we set the coordinates for the upper left corner, you already remember how to fix this yourself, because in **SVG** we often solved this issue.

**ctx.clearRect(175,175,50,50)**

Drawing lines, getting to know Path()

Now let's talk about drawing arbitrary shapes, for we have 4 methods

1. **beginPath ()**- start drawing
2. **closePath ()**- finish drawing
3. **stroke ()**- draw borders
4. **fill ()**- fill

We will write our base and all the other lines will be written between them.

**ctx.clearRect (172,175,50,50)**

**ctx.beginPath ();**

**ctx.closePath ();**

Let's draw a line to the lower right corner, the **lineTo** function will help us with this

**ctx.beginPath ();**

**ctx.lineTo (400,400)**

**ctx.closePath ();**

But the line did not appear, so that everything that we wrote is drawn, we need to execute **stroke ()**

**ctx.beginPath ();**

**ctx.lineTo (400,400)**

**ctx.stroke ()**

**ctx.closePath ();**

Again, there is no line, because we did not specify the initial coordinates, where the line should start from. To do this, we have the **moveTo** function - it moves our imaginary pencil to a given point, but does not draw

**ctx.beginPath ();**

**ctx.moveTo (0, 0);**

**ctx.lineTo (400,400)**

**ctx.stroke ()**

**ctx.closePath ();**

Let's draw another line, only on a different diagonal.

**I think the student can do it on his own**

**ctx.beginPath ();**

**ctx.moveTo (0, 0);**

**ctx.lineTo (400,400);**

**ctx.moveTo (400, 0);**

**ctx.lineTo (0, 400);**

**ctx.stroke ()**

**ctx.closePath ();**

Curves

To make a circle, arc or oval on canvas, use the same method, only with different arguments.

**arc (x0, y0, radius, startAngle, endAngle, direction)**

**x0** and **y0** denote the coordinates of the center

**startAngle, endAngle**- the initial and final angle of the arc (below we will consider a circle on which everything will be clearer)

**direction**- direction (**true** or **false**, clockwise or counterclockwise we go). Let's try to draw a circle inside our **Path()**

**ctx.beginPath ();**

**ctx.moveTo (0, 0);**

**ctx.lineTo (400,400);**

**ctx.moveTo (400, 0);**

**ctx.lineTo (0, 400);**

**ctx.arc (200, 200, 50, 0, 360, true);**

**ctx.stroke ()**

**ctx.closePath ();**

The result was not very good, because we forgot to do **moveTo** and move our pencil

**ctx.beginPath ();**

**ctx.moveTo (0, 0);**

**ctx.lineTo (400,400);**

**ctx.moveTo (400, 0);**

**ctx.lineTo (0, 400);**

**ctx.arc (200, 200, 50, 0, 360, true);**

**ctx.stroke ()**

**ctx.closePath ();**

The result was not very good, because we forgot to do **moveTo** and move our pencil

**ctx.moveTo (200,200)**

**ctx.arc (200, 200, 50, 0, 360, true);**

Now the result is even stranger

Icon

Description automatically generated

In fact, the arc function does not work with degrees, but radians, now comes a minute of math.

Chart, pie chart

Description automatically generated

Let's figure out what is shown on the circle. We see that 0 is on the right, counterclockwise direction. Degrees are red and radians are blue. That is, **90 degrees = Pi / 2, 180 degrees = Pi**, etc. Do not forget that a full circle is **360** degrees or **2 \* Pi**

With this knowledge, we can go scare the parents :)

Now to Canvas, where do we get this Pi? In fact, it is already declared in the Math library. Let's rewrite

**ctx.arc (200, 200, 50, 0, Math.PI \* 2, true);**

Great, there is a circle, but the extra line still remains

Diagram

Description automatically generated with low confidence

To get rid of it, you need to make such a movement using moveTo to get to the circle line, not to the center, but to the border. Let's think and indicate the necessary numbers

**ctx.moveTo (250,200)**

**ctx.arc (200, 200, 50, 0, Math.PI \* 2, true);**

A picture containing diagram

Description automatically generated

We also talked about the **fill()** function. Let's try to apply it too

**ctx.arc (200, 200, 50, 0, Math.PI \* 2, true);**

**ctx.stroke ();**

**ctx.fill ();**

Background pattern

Description automatically generated with medium confidence

Summing up and drawing a smiley

Let's draw a similar smiley based on the results of our knowledge on curved lines

Shape, circle

Description automatically generated

We can create a new document for this, but I suggest just creating another canvas in **HTML** with a different **ID** and working with it.

**let ctx2 = document.getElementById ('canvas2'). getContext ('2d');**

Let's make the background rectangle right away.

**ctx2.fillStyle = 'yellow';**

**ctx2.fillRect (0, 0, 150, 150);**

Now let's start drawing a smiley face, make a blank and a main circle.

**ctx2.beginPath ();**

**ctx2.arc (75, 75, 50, 0, Math.PI \* 2, true);**

**ctx2.stroke ();**

**ctx2.closePath ()**

Now we need to make a mouth, here we already need a semicircle, remember our radians and find out the final angle, just do not forget to move before drawing.

**ctx2.beginPath ();**

**ctx2.arc (75, 75, 50, 0, Math.PI \* 2, true); // Main circle**

**ctx2.moveTo (110, 75);**

**ctx2.arc (75, 75, 35, 0, Math.PI, false); // mouth**

I propose to comment on each of our actions, so as not to get confused later. Now let's move on to the eye circles.

**Hint: The eyes are at the same height, but they have different center x coordinates**

**ctx2.arc (75, 75, 35, 0, Math.PI, false); // mouth**

**ctx2.moveTo (65, 65);**

**ctx2.arc (60, 65, 5, 0, Math.PI \* 2, true); // Left eye**

**ctx2.moveTo (95, 65);**

**ctx2.arc (90, 65, 5, 0, Math.PI \* 2, true); // Right eye**

Done

Shape, circle

Description automatically generated

**Hometasks**

Make a house with a round window, like this, add other windows and doors at your discretion

A picture containing text, businesscard, envelope

Description automatically generated

**Teacher code**

**Let ctx = document.getElementById ('canvas'). getContext ('2d');**

**ctx.fillStyle = "# DBD3A6"**

**ctx.fillRect (0,150,400,250)**

**ctx.beginPath ()**

**ctx.moveTo (0, 150);**

**ctx.lineTo (200, 0)**

**ctx.lineTo (400, 150)**

**ctx.fillStyle = "# E6E793"**

**ctx.fill ()**

**ctx.moveTo (230.75)**

**ctx.arc (200,75,30,0, Math.PI \* 2, true)**

**ctx.stroke ()**

**ctx.closePath ()**