HTML5

<canvas> – 2D

Several new elements have been added to HTML5.

One of the new elements is canvas.

Functionally very similar to the canvas found in most graphics programs.

The main difference - in graphic programs we draw by default with an available tool (mouse, pen) in canvas HTML, we create graphic on the fly, with JavaScript.

The <canvas> element is only a container for graphics. You must use JavaScript to actually draw the graphics.

A canvas is a rectangular area on an HTML page. By default, a canvas has no border and no content.

Canvas element (tag):

<canvas width="300" height="200"></canvas>

HTML attributes provide additional information about HTML elements.

Default canvas width=300 pixels, hight=150 pixels high.

Incompatibility the height and width (e.g., by modifying it in CSS) can lead to a distortion in the aspect ratio of the rendered image.

If renderings seem distorted, try to specify width and height attributes in the <canvas> attributes, not in CSS.

We can also add a centering style attribute to the tag <body>.

<body style="text-align: center;">

<canvas id="Canvas1"

width="300" height="200"

style="border:1px solid #000000;

background-color: antiquewhite;">

</canvas>

</body>

Item <canvas> behaves on the page similar to an item <img>.

Using JavaScript, you can place images, draw shapes, read parameters, etc. on canvas.

We can also copy or save, just like with the img element.

We can dynamically change the content of canvas. So we can create various types of animations.

In this respect, canvas is a very convenient and useful element.

However, direct work with canvas is low-level work, often very difficult, tedious and laborious.

For intensive or advanced work, we often use various available libraries, such as: p5.js, Raphael, paper.js, chart.js.

There are also engines for creating games using the canvas element.

Element canvas:

<canvas> </canvas>

1. **We create a new HTML document and add an element <canvas>.**

We set the canvas size,

To make the canvas visible, set the background color and canvas color.

Here, these parameters are set directly in the HTML file.

However, we should remember that usually we should place the formatting externally in the CSS file.

**Example 1a**

**W\_2D\_01a.html**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>A blank canvas 1</title>

<style>

body{ background-color: grey; }

canvas{ background-color: white; }

</style>

</head>

<body>

<h1>Widzimy canvas</h1>

<canvas width="400" height="300">

Your browser does not support the HTML5 canvas element.

</canvas>

</body>

</html>

We place the replacement content inside the <canvas>... </canvas> element.

Browsers that do not support <canvas>, will ignore the tag, instead of the tag displaying what is inside the tag.

Browsers that support the <canvas> element will generate its content by ignoring what is inside the markup.

The tag construction requires a closing tag </canvas>.

The absence of a closing tag will cause everything after the <canvas> tag to be treated as replacement content.

We modify the example by changing the format saving location.

**Example 1b**

**W\_2D\_01b.html**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>A blank canvas 2</title>

<style>

body {

margin: 0px auto;

text-align: center;

background-color: grey;

}

canvas {background-color: white}

#my-canvas{

width: 400px;

height: 200px;

}

</style>

</head>

<body>

<h1>Welcome to the canvas</h1>

<h2>We see a white canvas on a gray background.</h2>

<canvas id = "my-canvas">

Your browser does not support the HTML5 canvas element.

</canvas>

</body>

</html>

**Example 1c**

**W\_2D\_01c.html**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>3 A blank canvas</title>

<link rel="stylesheet" href="W\_2d\_01c.css">

</head>

<body>

<h1>Welcome to the canvas</h1>

<h2>We see a white canvas on a light blue background.</h2>

<canvas id = "my-canvas">

Your browser does not support the HTML5 canvas element.

</canvas>

</body>

</html>

**W\_2D\_01c.css**

body {

margin: 0px auto;

text-align: center;

background-color: lightblue;

}

canvas {background-color: white}

#my-canvas{

width: 300px;

height: 300px;

}

**The rendering context**

The <canvas> element creates a fixed-size drawing surface.

It exposes two rendering contexts:

* 2D
* WebGL (3D context based on OpenGL ES)

which are used to create and manipulate the content shown.

To be able to perform canvas operations, we must have access to the canvas context.

HTML:

<canvas id = "my-canvas"> </canvas>

JavaScript:

const canvas = document.getElementById("my-canvas");

const context = canvas.getContext('2d');

We make a drawing using context methods.

* Point (0,0) is the upper left corner of the canvas.
* Fill color setting.
* Draw a filled rectangle (x,y,dx,dy).

1. **Drawing on the <canvas> element**
   1. **Determining the fill color**
   2. **Draw a filled rectangle**
   3. **Repetition three times**
   4. **Drawing made with the JS function**
   5. **The function performed when the page is loaded**

**Example 2**

In the <canvas> element, we overlap three colored squares.

**W\_2D\_02.html**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Rectangles 1</title>

<style>

body {

margin: 0px auto;

text-align: center;

}

</style>

<script>

window.onload = setup2d;

function setup2d() {

const canvas = document.getElementById("my-canvas");

const context = canvas.getContext("2d");

context.fillStyle = "rgb(200,0,0)";

context.fillRect(10,10,100,100);

context.fillStyle = "rgb(0,200,0)";

context.fillRect(20,20,100,100);

context.fillStyle = "rgb(0,0,200)";

context.fillRect(30,30,100,100);

}

</script>

</head>

<body>

<h1>Welcome to the canvas</h1>

<h2>We see a white canvas on a white background.</h2>

<canvas id = "my-canvas" width="150" height="150">

Your browser does not support the HTML5 canvas element.

</canvas>

</body>

</html>

**Task 1**

Build a page (modification of the previous example) displaying four equal rectangles, placed symmetrically in the four corners of the canvas.

The colors of the rectangles are to be random.

**Drawing a polygonal chain**

* Line color setting
* Line thickness setting
* Setting the beginning of the line
* Line to
* Draw

context.strokeStyle = "#00FF00";

context.lineWidth = 2;

context.beginPath();

context.moveTo(0, 40);

context.lineTo(30, 0);

context.lineTo(60, 40 );

context.lineTo(285, 40 );

context.stroke();

**Example 3**

W\_2D\_03a.html

In the <canvas> element we draw a polygonal chain.

Preparation for drawing the company logo.

<body>

<h3>Welcome to the company</h3>

<canvas id="logo" width="900" height="80">

<h1>Our Company</h1>

</canvas>

<script>

var drawLogo = function(){

var canvas = document.getElementById("logo");

var context = canvas.getContext("2d");

context.strokeStyle = "#00FF00";

context.lineWidth = 2;

context.beginPath();

context.moveTo(0, 40);

context.lineTo(30, 0);

context.lineTo(60, 40 );

context.lineTo(285, 40 );

context.stroke();

}

var canvas = document.getElementById("logo");

if (canvas.getContext){

drawLogo();

}

</script>

</body>

From the DOM object we get access to the <canvas> element, and then access to the "2d" context.

We set the drawing color (brush strokes) (strokeStyle) to green, line width to 2 points (lineWidth).

With the beginPath() method we start a new "path", reset the old settings.

The stroke() method makes a track drawing.

**Task 2**

Write the version of example 3 with the red path reflected symmetrically below.

**Example 3b**

W\_2D\_03b.html

In the <canvas> element we draw a polygonal chain.

We fill the area defined by a polygonal chain.

context.fillStyle = "#FF0000";

context.fill();

W\_2D\_03b.html

We close the polygonal chain, connecting the end of the last line segment with the beginning of the first.

context.closePath();

**Example 3c**

W\_2D\_03c.html

We add text to <canvas>.

We can add text in two ways:

* context.fillText("Nasza Firma", x, y) – text filled in position (x,y);
* context.strokeText("Nasza Firma", x, y) – text outlined in position (x,y);

context.font = "italic 40px 'Arial'";

context.fillText("Nasza Firma", 60, 36);

or

context.font = "italic 40px 'Arial'";

context.strokeText("Nasza Firma", 60, 36);

**Example 3d**

W\_2D\_03d.html

We will add next "drawings", but it will be more convenient to make them at a different origin of the coordinate system.

We will make the work easier by saving the current settings:

context.save();

By changing the <canvas> settings - moving the origin of the coordinate system:

context.translate(20,20);

Performing new "drawings":

context.lineWidth = 2;

context.beginPath();

context.moveTo(0, 20);

context.lineTo(10, 0);

context.lineTo(20, 20 );

context.lineTo(0, 20 );

context.fill();

context.closePath();

context.stroke();

Restoring the initial settings.

context.restore()

**Example 3e**

W\_2D\_03e.html

**Linear Gradient**

We will change the initial fill from red to a gradient going from red to black and red.

Instead of the initial:

context.fillStyle = "#FF0000";

context.strokeStyle = "#00FF00";

We introduce:

var gradient = context.createLinearGradient(0, 0, 0, 40);

gradient.addColorStop(0, "#110000");

gradient.addColorStop(1, "#FF0000");

context.fillStyle = gradient;

context.strokeStyle = gradient;

**Definitions:**

context.createLinearGradient(x0,y0,x1,y1);

(x0,y0), (x1,y1) – gradient start and end points.

**Example 3f**

W\_2D\_03f.html

**RadialGradient**

var c = document.getElementById("myCanvas");

var ctx = c.getContext("2d");

// Create gradient

var grd = ctx.createRadialGradient(75, 50, 5, 90, 60, 100);

grd.addColorStop(0, "red");

grd.addColorStop(1, "white");

// Fill with gradient

ctx.fillStyle = grd;

ctx.fillRect(10, 10, 150, 80);

**Draw a line**

var c = document.getElementById("myCanvas");

var ctx = c.getContext("2d");

ctx.moveTo(0, 0);

ctx.lineTo(200, 100);

ctx.stroke();

**Draw a Circle**

var c = document.getElementById("myCanvas");

var ctx = c.getContext("2d");

ctx.beginPath();

ctx.arc(95, 50, 40, 0, 2 \* Math.PI);

ctx.stroke();

arc(x, y, r, angle0, angle1)