Typical use of the <canvas> element

DETAILED EXPLANATION OF THE EXAMPLE SHOWN IN THE VIDEO

Here are the different steps, a little more detailed, of the example demonstrated in the above video:

1 - Add the <canvas> element into an HTML page

```
<canvas id="myCanvas" width="300" height="225">
  Fallback content that will be displayed in case the web browser
  does not support the canvas tag or in case scripting
  is disabled.
</canvas>
```

Place code similar to the above somewhere in an HTML page. This example defines an area of 300 by 225 pixels on which content can be rendered with JavaScript.

Normally you should see nothing as a result, by default canvases are "transparent".

Make it visible using CSS: A good practice when you learn using canvases is to use some CSS to visualize the shape of the canvas. This is not mandatory, just a good trick...

The three lines of CSS will create a border around the canvas with `id="myCanvas"`, of 1 pixel width, in black:

CSS code:

```
<style>
  #myCanvas {
     border:1px solid black;
  }
</style>
```

2 - Select the <canvas> element for use from JavaScript

We can have more than one `<canvas>` in a single page, and canvases will be manipulated with JavaScript like other elements in the DOM.

For example with:

```
var canvas = document.getElementById("mycanvas");
```

... or with the `querySelector()` method introduced by HTML5, that use the CSS selector syntax for selecting elements:

```
var canvas = document.querySelector("#mycanvas");
```

3 - get a "2D context" associated with the canvas, useful for drawing and setting drawing properties (color, etc.)

Once we have a pointer to the `<canvas>`, we can get a "context".

This particular object is the core of the canvas JavaScript API.

It provides methods for drawing, like `fillRect(w, y, width, height)` for example, that draws a filled rectangle, and properties for setting the color, shadows, gradients, etc.

Getting the context (do this only once):

```
var ctx=canvas.getContext('2d');
```

Set the color for drawing filled shapes:

```
ctx.fillStyle='red';
```

Draw a filled rectangle:

```
ctx.fillRect(0,0,80,100);
```

COMPLETE EXAMPLE THAT DRAWS A FILLED RECTANGLE IN RED

Try it online at JS Bin

Result:

Source code:

```
<!DOCTYPE html>
<html lang="en">
 <head>
 <style>
    #myCanvas {
       border: 1px solid black;
    }
 </style>
 <script>
  var canvas, ctx;

  function init() {
    // This function is called after the page is loaded
    // 1 - Get the canvas
    canvas = document.getElementById('myCanvas');
    // 2 - Get the context
    ctx=canvas.getContext('2d');
    // 3 - we can draw
    drawSomething();
  }
  function drawSomething() {
    // draw a red rectangle
    ctx.fillStyle='#FF0000';
    ctx.fillRect(0,0,80,100);
  }
 </script>
 </head>
<body onload="init();">
   <canvas id="myCanvas" width="200" height="200">
         Your browser does not support the canvas tag.
   </canvas>
</body>
</html>
```

Explanation

Only access elements when the DOM is ready:

Notice that we wrote an "init" function (line 12) that is called only when the page has been entirely loaded (we say "when the DOM is ready"). There are several ways to do that. In this example we used the `<body onload="init();">` method, at line 32.

It's a good practice to have such a function, as we cannot access the elements of the page before the page has been loaded entirely and before the DOM is ready.

Another way consists in putting the JavaScript code at the end of the document (between`<script>...</script>`), right before the `</body>`. In that case when the JavaScript code is executed, the DOM has already been constructed.

Start by getting the canvas and the context:

Before drawing or doing anything interesting with the canvas, we must first get its drawing "context". The drawing context defines the drawing methods and properties we can use.

Good practice is to get the canvas, the context, the width and height of the canvas and other global objects in this "init" function.

After the context is set, we can draw, but before, let's set the current color for filled shapes:

The example shows at line 27 the use of the `fillStyle` property - useful for specifying the way shapes will be filled. In our case this line indicates the color of all the filled shapes we are going to draw:

```
ctx.fillStyle='#FF0000';
```

The context property named `fillStyle` is used here. This property can be set with a color, a gradient, or a pattern. We will see examples of these later on in the course.

When we set it with a color, we use the CSS3 syntax.

The example says that all filled shapes will use the color "#FF0000", which corresponds to a pure red color using the CSS RGB hexadecimal encoding (we could also have used`ctx.fillStyle='red'`);

Then we can draw:

```
ctx.fillRect(0,0,80,100);
```

This line is a call to the method `fillRect(top left X coordinate, top left Y coordinate, width, height)`, which draws a filled rectangle.

The way the rectangle will be filled depends on the current value of several properties of the context, in particular the value of the `fillStyle` property. So, in our case, the rectangle will be red.

SUMMARY OF THE DIFFERENT STEPS

1. **Declare the canvas,** not forgetting to add an `id` attribute, and fallback content:
   ```
   <canvas id="myCanvas" width="200" height="200">
   ...fallback content...
   </canvas>
   ```
2. **Get a reference to the canvas in a JavaScript variable** using the DOM API:
   ```
   var canvas=document.getElementById('myCanvas');
   ```
3. **Get the context for drawing in that canvas**:
   ```
   var ctx=canvas.getContext('2d');
   ```
4. **Specify some drawing properties** (optional):
   ```
   ctx.fillStyle='#FF0000';
   ```
5. **Draw some shapes**:
   ```
   ctx.fillRect(0,0,80,100);
   ```

KNOWLEDGE CHECK 3.2.6 (NOT GRADED)

Drawing shapes in a canvas is only possible from JavaScript, and the canvas must be selected using the DOM API first, using `document.getElementById(...), document.querySelector(...)`, etc.

- True
- False