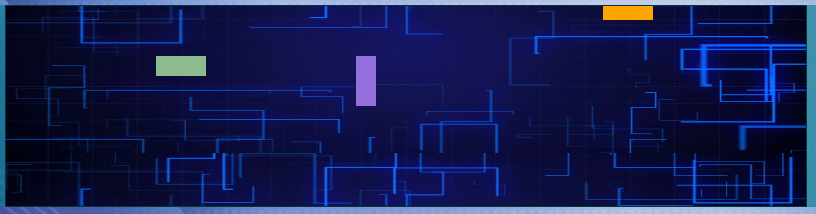
Intro to Canvas Graphics



**Introduction**:

Canvas is a new element type introduced with HTML5 that is designed to render graphics in real-time. This makes it highly desirable for making web-based games and similar applications where speed of rendering is critical. This tutorial is intended to provide a brief look at how to set up the Canvas to create and render graphics.

**Preparation**:

For this tutorial, you will need access to an application that can edit JavaScript and HTML files. The tutorial was created using Visual Studio Code, but any document editing application should suffice.

The starter code for this tutorial can be found at the following url:

<https://github.com/pipboy2/CIT-255-Canvas-Project>.

The tutorial package should contain the following folders and files:

* css
* canvasDemo.css
* js
* canvasDemo.js
* canvasDemoStudentVersion.js
* media
* Background.png
* Bg2.png
* digitalBG.jpg
* Canvas\_Demo\_Student\_Version.html
* Canvas\_Demo\_V1.html
* Canvas Overview and Learning Outcomes.docx
* CIT 255 Lab Procedure\_.docx

For this tutorial, you will be working with the canvas\_Demo\_Student\_Version.js file located in the js folder. This version is missing some sections of code that will be filled in over the course of this tutorial. A completed version, canvasDemo.js, is also included so you can compare your progress with a working version.

By the end of this tutorial, you should be able to:

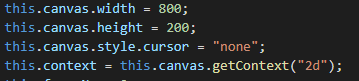
* Configure a Canvas element in JavaScript
* Create a JavaScript pseudo-class and methods for Canvas graphics
* Instantiate and update Canvas graphics
* Use events to manipulate the position of Canvas graphics

**Step 1.**

In order to use the Canvas, we first need to instantiate one to work with and store it in a variable (d2 in the example). The provided HTML documents have a Canvas element included with an ID of “**movingCanvas**”, so we can use *document.getElementById* to specify that canvas element as the one we will be using. Then, we create a “**start**” function to specify the characteristics of the canvas.

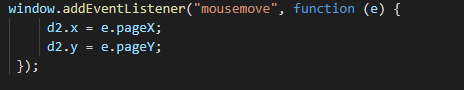


Within this function we will need to define a few characteristics:



The **Width** and **Height** characteristics dictate the size of the canvas’ boundaries in pixels. For this tutorial the size does not matter much, but it is generally recommended to choose dimensions that will look good on mobile devices since Canvas is not inherently responsive.

The Canvas’ **Context** is used to dictate which drawing methods are available to use in the application. Since we will be working with basic shapes, we will use "2d" as the argument for the ***getContext*** method. We also don’t want the mouse cursor to appear on the Canvas boundary since some of the graphics will be tied to the movements of the mouse later in the tutorial. Speaking of which, we will want to include an event listener to monitor the movement of the mouse and store its location as a variable that we can access later.



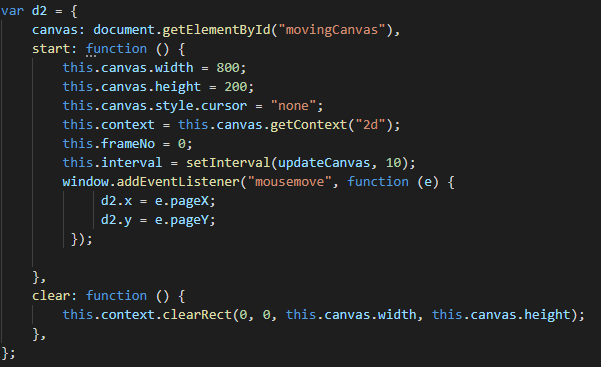
To update the Canvas we need to call an external function at regular intervals. To do this, create an interval which will call the **updateCanvas** function (we will go over this later) every 10 milliseconds.



We also need to create a function to clear the Canvas area. This is important for the updateCanvas function to provide fluid motion with the graphics. Otherwise, a new graphic will be drawn over what has previously been drawn and the result will be messy.

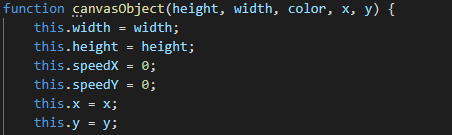


The final code for the Canvas should look something like this:



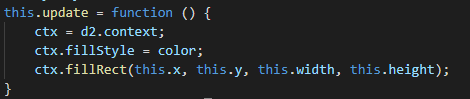
**Step 2.**

One of the main benefits of Canvas is that once it has drawn something it releases the memory used, making it very fast. However, this also means that graphics need to be redrawn each time something changes. We can make this process much easier by creating a JavaScript pseudo-class to hold the properties of graphics we wish to draw so that it can be reused.

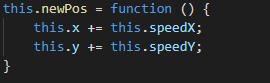


For this example, we are setting the graphic’s ***height***, ***width***, ***color***, ***x position***, and ***y position***. The x and y properties control where the resulting graphic will appear on the canvas, with 0, 0 being the top left corner of the canvas. Meanwhile, the speedX and speedY properties allow the position to gradually changed between updates.

We also need to create two functions: **update** and **newPos**.

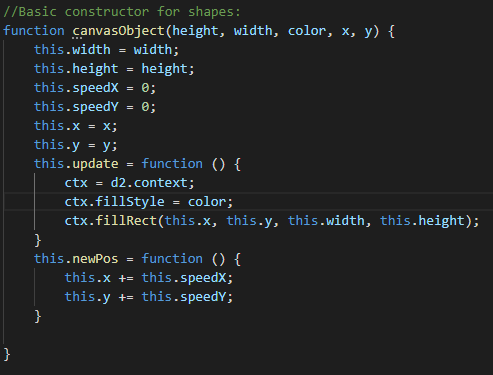


The **update** function is used to call the Canvas drawing method using the pseudo-class’ properties as arguments. For this tutorial we are using the basic **fillRect** method, but there are others that can be used to achieve different shapes.

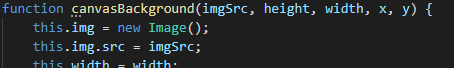


The **newPos** function serves as the mechanism that allows a graphic to move around the canvas. This also allows the position of a graphic to be updated in real time based on user input.

The final constructor should look like this:



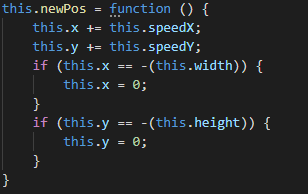
The starter code also includes a background constructor built off the basic one with a few modifications. The first notable difference is that the **update** function for the background graphic requires an image source path instead of color as a parameter.



This is because the background constructor is using the **drawImage** method in its update function, which does exactly what it the name implies and draws a specified image on the Canvas instead of lines or a shape. Another difference in the update method is the drawing of two instances of the background image rather than one.



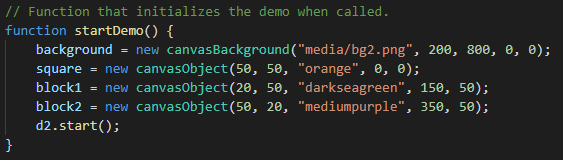
The background in this demo is designed to give the illusion of continual looping movement, so we need a duplicate to continue moving up the screen after the original resets. For this demo the background flows upward, so the second one has a y position equal to y + the height of the image. If the goal was to have the background move side-to-side, the x position and width properties would be used instead.

****

Lastly, the **newPos** function for the background is a bit more involved than that of the other graphics as it includes code to reset its position after reaching the edge of the Canvas.

**Step 3.**

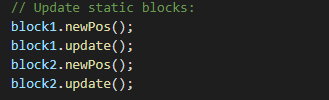
Now that we have a pseudo class to hold our information, it is time to instantiate some graphics! Find the **startDemo** function near the bottom of the starter code. This function is used to instantiate each graphic that will be drawn. It also initializes the Canvas itself.



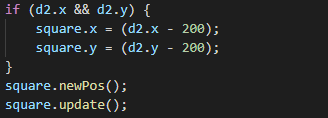
**NOTE:** Each item needs to be put before the start() function is called, as otherwise they won’t exist when startDemo calls the updateCanvas function and the web browser will throw an error.

**Step 4.**

If you attempt to launch the Canvas now you will find that the graphics you instantiated are not appearing. This is because the update function for each graphic has not been called, so they have not been drawn. To fix this, we will need to call each graphic’s update function within the **updateCanvas** function (located right after the constructors). You will need to make one newPos and update pair for each item you instantiated previously. The newPos function should be called *before* the update function if you have any moving graphics.



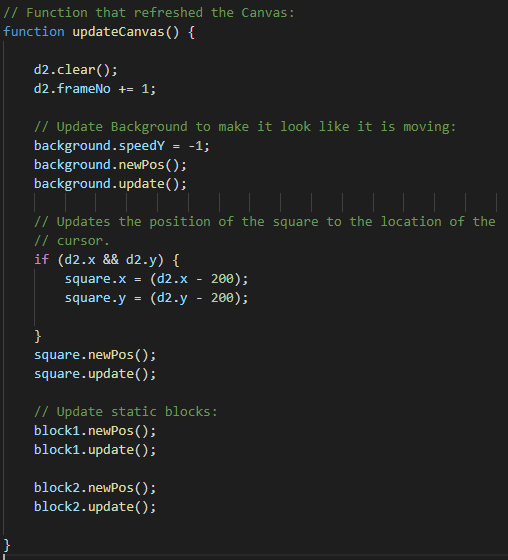
Remember that event listener we created at the start? We can finally put it to use here to move a graphic based on the position of the mouse.



A simple **if** statement ensures that the positional changes will only occur if the event handler is active. In this case we are directly copying the mouse’s position to the coordinates of the canvas, with a small offset so that mouse and graphic line up (this is unnecessary if the canvas encompasses the entire screen).

The section for the background has already been implemented and serves to demonstrate how the speedY (and speedX) work to move a graphic.



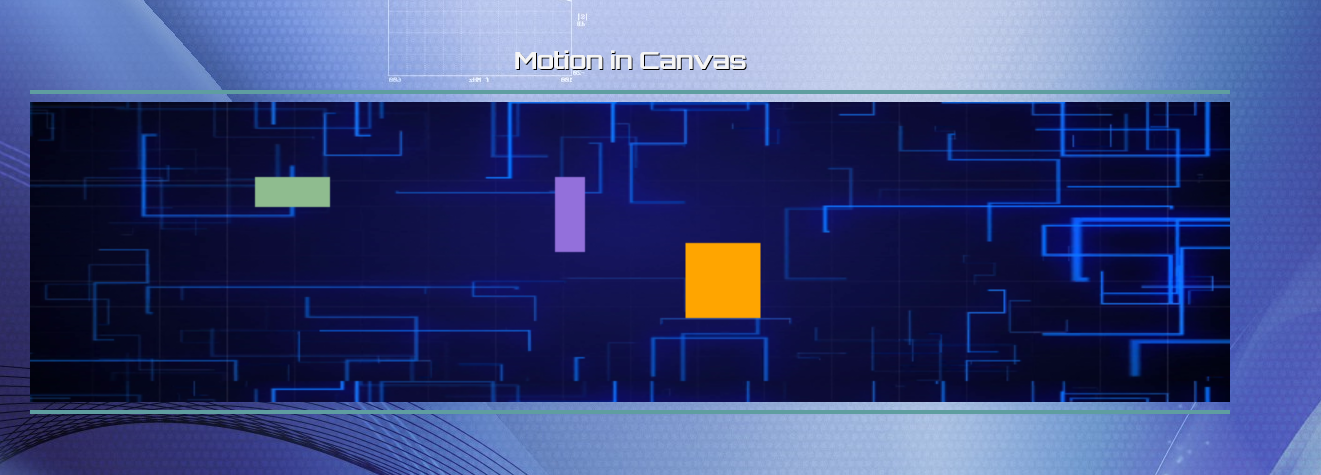
Since the background’s movement is hardcoded into its newPos method no further work is needed, but if you wanted other graphics to move you could set up a similar “if-else” statement before each newPos is called to control when (and if) the graphic resets to a starting position. The final updateCanvas function should look like this: 

**Conclusion.**

Congratulations! You have just made your first Canvas-based application. Open up the **canvas\_Demo\_Student\_Version.html** file in your web browser and click the “start demonstration” button to see the results of your efforts.



If all goes well, you should be greeted with something akin to the following:



We hope you found this brief tutorial useful and that you had fun playing with Canvas.

**Assessment**

Kahoot App Link: <https://www.microsoft.com/en-us/p/kahoot-play/9nzg3bqpgkj2>

Assessment Link: <https://kahoot.it/challenge/0697697>

Assessment Pin: 0697697

**Evaluation**

Click on the link to access the PBJ^2 Evaluation [PBJ2 Canvas Demo](https://www.surveymonkey.com/r/C75HSZC) or access it through the following QR code:

