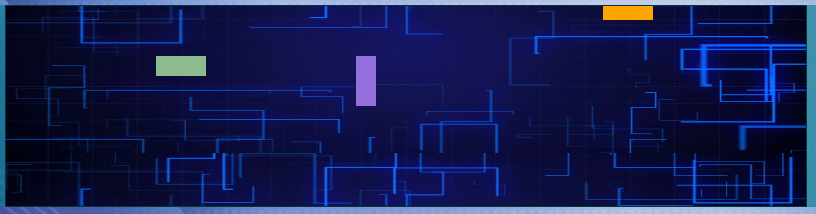
Intro to Canvas



**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:

Folders Files

* css

1. canvasDemo.css

* Js

1. canvasDemo.js
2. canvasDemoStudentVersion.js

* Media

1. Background.png
2. Bg2.png
3. digitalBG.jpg

* Canvas\_Demo\_Student\_Version.html
* Canvas\_Demo\_V1.html
* Canvas Overview and Learning Outcomes.docx
* CIT 255 Lab Procedure\_.docx

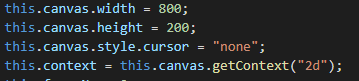
For this tutorial, you will be working with [canvasDemoStudentVersion.js] located in the js folder. This version is missing some sections of code that will be filled in over the course of this tutorial. A completed version is also included so that you can compare your progress with a working version.

**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.

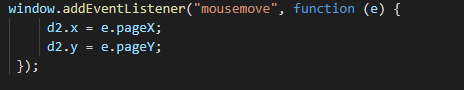


Within this function we will need to define a few characteristics:



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.

Context is used to dictate which Canvas methods are available to use in the application. Since we will be working with basic shapes, 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.



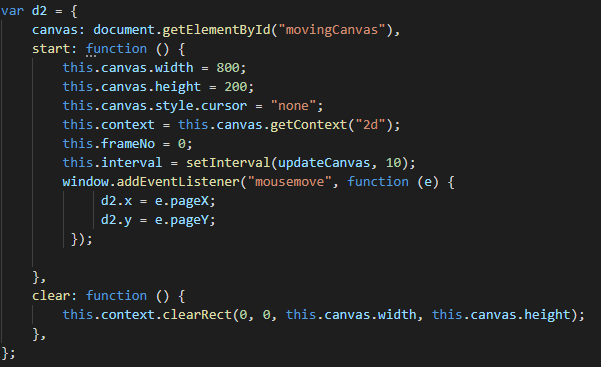
To update the Canvas we need to call an external function at regular intervals. To do this, create an interval which will call the updateCanvas function (we will go over this later) every 10 milliseconds.



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 the previously drawn graphics.

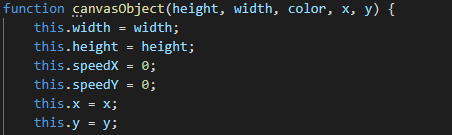


That was a lot! The final code should look something like this:



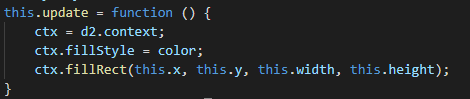
**Step 2.**

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 work around this by creating a JavaScript pseudo-class to hold the properties of graphics we wish to draw so that it can be reused.

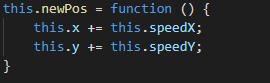


For this example, we are setting the graphic’s height, width, color, x position, and y position. The x and y properties control where the resulting graphic will appear on the canvas, while the speedX and speedY properties allow the position to gradually changed between updates.

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

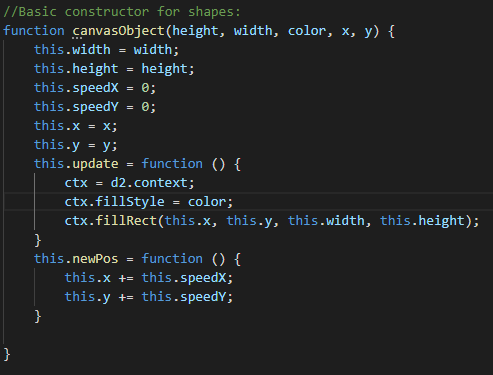


The update function is used to call the Canvas drawing method using the properties provided as arguments. This function can be modified to accommodate other shapes and even images, but that is beyond the scope of this tutorial.



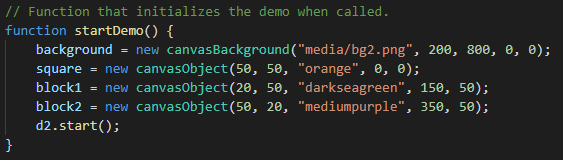
The newPos function serves as the mechanism that allows a graphic to move around the canvas. This also allows the position of a graphic to be updated in real time based on user input.

The final constructor should look like this:



**Step 3.**

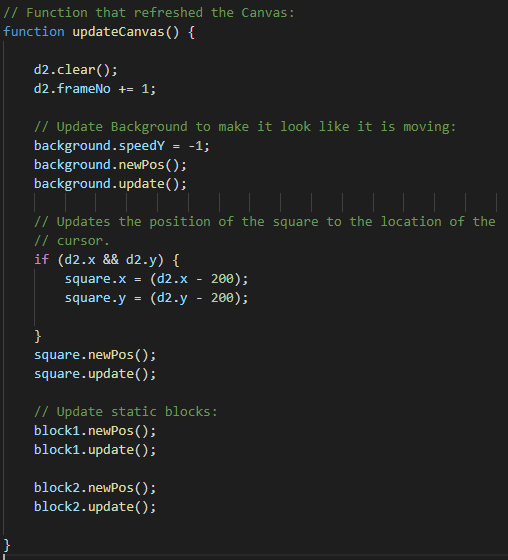
Find the startDemo function near the bottom of the starter code. This function is used to instantiate each graphic that will be drawn. It also initializes the Canvas itself.



Each item needs to be put before the start() function is called, as otherwise they won’t exist when said function 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 created are not appearing. This is because the update function for each graphic has not been called, meaning they have not been drawn. To fix this, we will need to call each graphic’s update function within the updateCanvas function. The new position function can also be put here.

The final result should look like this: 

We also can access our event listener and use it to update the position of a graphic.

**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**

Evaluation [PBJ2 Canvas Demo](https://www.surveymonkey.com/r/C75HSZC)



Results [Link](https://www.surveymonkey.com/results/SM-KSLRDVQJ7/)