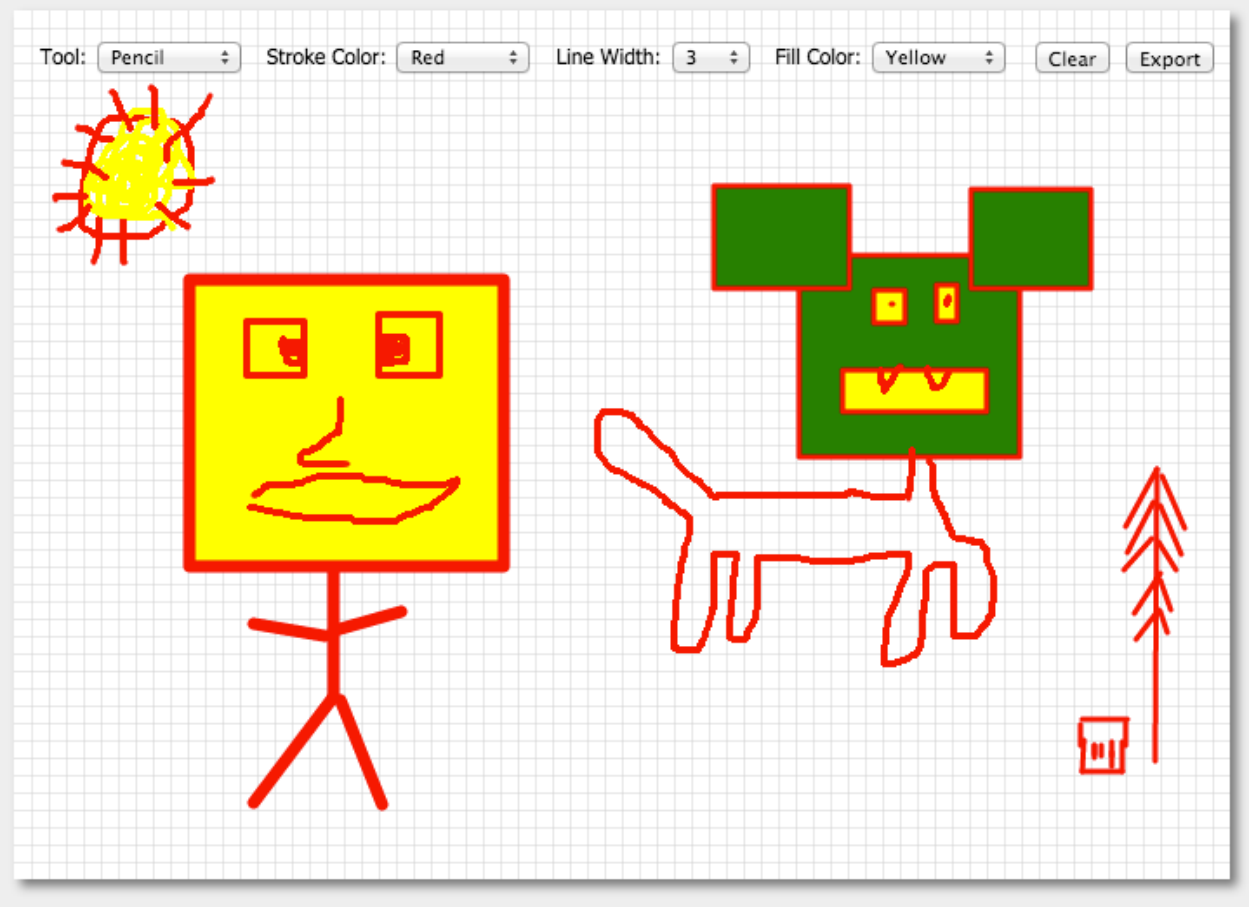
Canvas Painting App - Part I

In this 2-part exercise we are going to build a canvas painting app that has the following capabilities:

- The drawing will happen on a “graph paper” background.
- Pencil, Rectangle, and Line tools
- Controls for stroke color, fill color, and line width
- a button for clearing the canvas
- a button for exporting the contents as a JPEG
- Additionally, the Rectangle and line tools will allow the user to preview their drawing before it is committed to the canvas, and the operation will be cancelled if the user drags the pointer out of the canvas window.



Part A - Paint the Background Grid

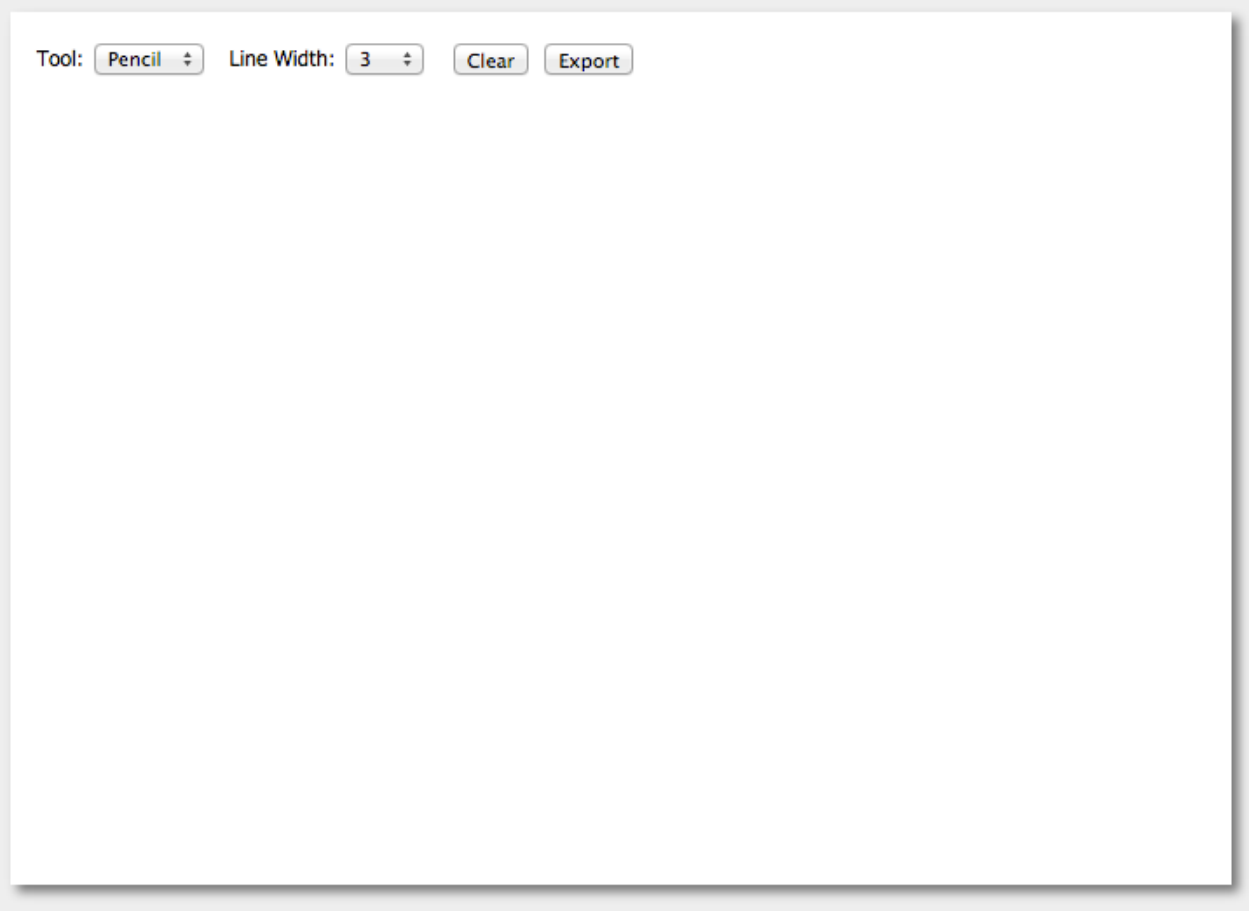
1) Download the start file, open it up in a text editor, and preview it in Chrome. There's not much going on yet as none of the controls work yet.

   Look over the HTML, CSS and JS:

   - note that the controls are *absolutely positioned* on top of the `<canvas>` element

   - there is an `init()` function in which we will initialize our `ctx` and `canvas` "module scoped" variables

   - there are many functions already stubbed in for you, and three of them - `doClear()`, `doExport()`, and `getMouse()` - are fully complete and will not need to be modified by you.



2) Let's get coding. Declare the following values in the `CONSTANTS` section:

```
// CONSTANTS
const defaultLineWidth = 3;
const defaultStrokeStyle = "red";
```

3) Make `init()` appear as follows

```
// FUNCTIONS
const init = () => {
  // initialize some variables
  canvas = document.querySelector("#main-canvas");
  ctx = canvas.getContext("2d");
  lineWidth = defaultLineWidth;
  strokeStyle = defaultStrokeStyle;

  // set initial properties of the graphics context
  ctx.lineWidth = lineWidth;
  ctx.strokeStyle = strokeStyle;
  ctx.lineCap = "round"; // "butt", "round", "square" (default "butt")
  ctx.lineJoin = "round"; // "round", "bevel", "miter" (default "miter")

  drawGrid(ctx, "lightgray", 10, 10);
};
```

After adding this code and refreshing the page, there shouldn't be any errors and you should see 1/2 of the light-gray grid being drawn on the canvas - just the vertical lines.

4) Look over the `drawGrid()` function to see how it works, but we only included the code for the vertical lines.

Add another `for` loop in `drawGrid()` that will draw the missing horizontal lines. When you are done the canvas will look something like graph paper with 10-pixel square cells.

Part B - Building the pencil tool

1) First, we'll hook up 4 mouse events to our 4 stubbed in callback functions. To do this, add the following code to the end of `init()`

```
// Hook up event listeners
canvas.onmousedown = doMousedown;
canvas.onmousemove = doMousemove;
canvas.onmouseup = doMouseup;
canvas.onmouseout = doMouseout;
```

Test the code by clicking the mouse and moving it outside of the canvas, you should see logs to the console as the applicable functions are fired. (Note: We didn't add a `console.log()` to `doMousemove()` because the large number of logs would have cluttered the console.)

2) To begin our pencil drawing, make `doMousedown()` appear as follows:

```
const doMousedown = (evt) => {
  console.log(evt.type);
  dragging = true;

  // get location of mouse in canvas coordinates
  const mouse = getMouse(evt);

  // PENCIL TOOL
  ctx.beginPath();

  // move pen to x,y of mouse
  ctx.moveTo(mouse.x, mouse.y);
};
```

3) Next we need to work on `doMouseMove()` - here's a start for you - including some helpful comments:

```
const doMousemove = (evt) => {
  // bail out if the mouse button is not down
  if(!dragging) return;
     // get location of mouse in canvas coordinates
     const mouse = getMouse(evt);

     // PENCIL TOOL
     // set ctx.strokeStyle and ctx.lineWidth to correct "module variable" values
     // YOUR CODE HERE

    // draw a line to x,y of mouse
    // YOUR CODE HERE

    // stroke the line
    // YOUR CODE HERE
};
```

Test the app. It should now draw when you click and drag. Unfortunately, it never stops drawing, so fix that by:

- heading to `doMouseup()`, and set `dragging` to `false` and close the path

- heading to `doMouseout()`, and set `dragging` to `false` and close the path

Now it should draw as you expect.

Part C – Enabling the GUI

1) First, we'll get the `#lineWidthChooser` drop down working. We'll go ahead and give you the full solution to this:

- add the following to the end of `init()`

```
document.querySelector("#linewidth-chooser").onchange = doLineWidthChange;
```

- add the following function:

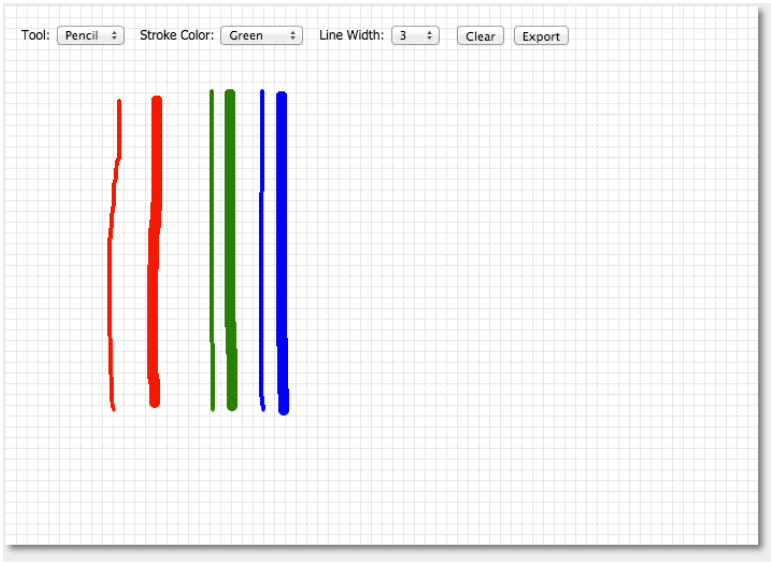
```
const doLineWidthChange = (evt) => {
  lineWidth = evt.target.value;
};
```

Test it - and you should see it function as expected. When you change the value of the line width chooser, the `lineWidth` module variable is then changed. And the next time you draw, inside of `doMousemove()` the `ctx.lineWidth` value is being changed to the current value of the `lineWidth` module variable.

2) Create a `<select id="strokestyle-chooser">` to allow the user to choose a `strokeStyle`. The values of the `<option>` tags should be CSS color keywords.

Now go ahead and hook up an `onchange` event handler as before, except this time it changes the value of the `strokeStyle` module variable.

When you are done it should function as shown →



3) Finally, get the **Clear** and **Export** buttons working. Both of the functions that need to be called have already been written for you - easy!