IT@JCU MOBILE CODING

HOW TO USE THIS WORKSHOP

This is a self-paced workshop that guides you through the process of making your very own ***mobile web app***. Your app will work on most modern smartphones. If you have your own iPhone or Android phone, please let us know and we can help you put your game on your phone!

Please follow through the steps below in order. They will help you explore mobile web programming using HTML/CSS/JavaScript.

PART 1 STEPS:

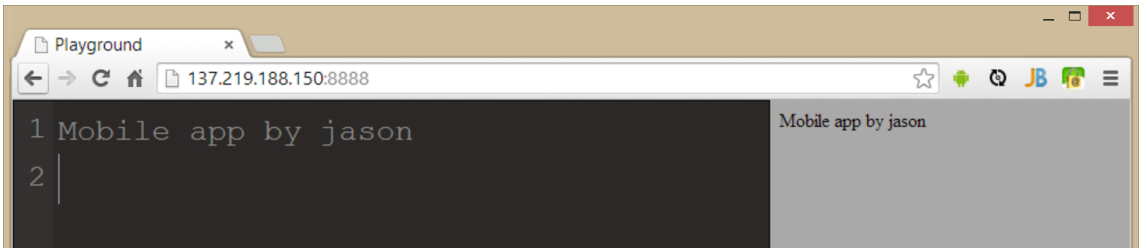
1. Make sure you can see the following on your computer screen:

   - This shows the "Playground IDE" – this is a browser-based code editor
   - The Playground IDE window is split into two parts:
     - The *code editor* (left-hand side) is where you type in your code
     - The *mobile emulator* (right-hand side) shows what your app would look like running on a mobile phone

2. Click your mouse cursor inside the black area and type in the following:

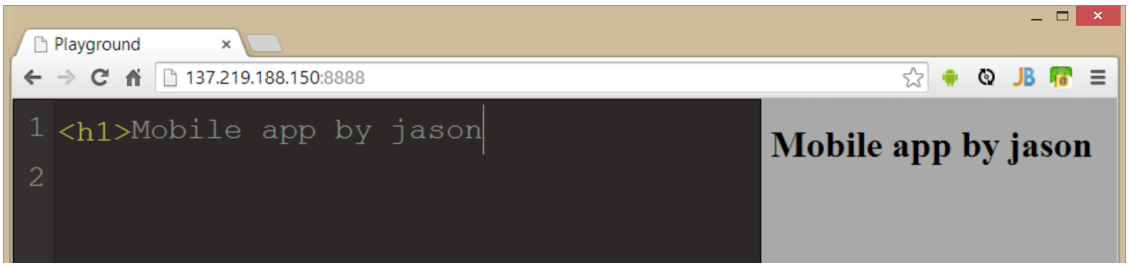
```
Mobile app by put your name here
```

3. Instantly, the mobile emulator shows what your program would look like running:



- Every time you type in text inside the code editor the mobile emulator will update immediately
- At the moment, the code editor is in **HTML mode**
    - You can type in plain text and HTML tags

4. Let's use a **HTML tag** called a **header tag**, adjust your code to look like this:

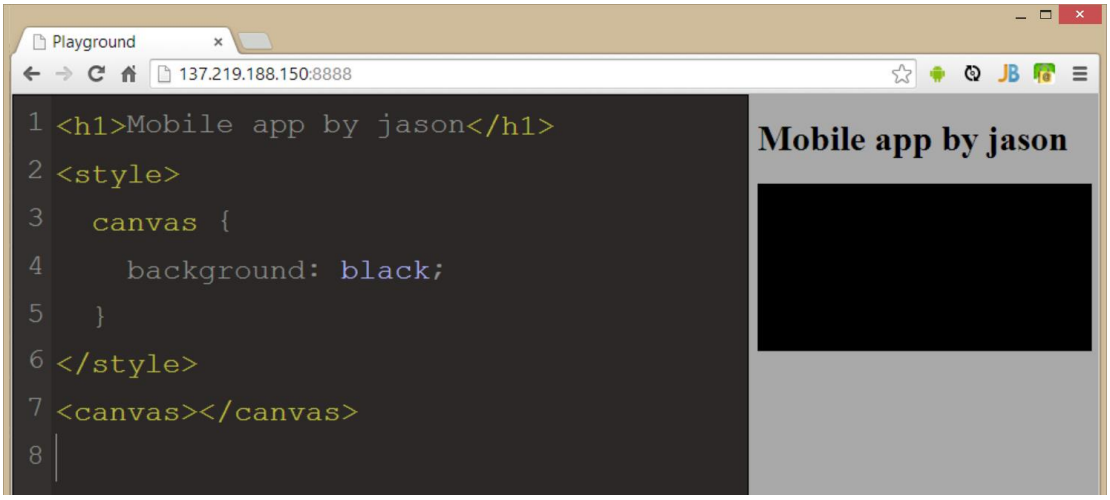


- Notice how this increases the size of the text in the running program
- The purpose of HTML is to **control the structure and layout of the text**

5. Next, we need a place to draw graphics – we add a HTML <canvas> tag pair, but to do this properly, we must also complete the header tag pair as follows:

- Many HTML tags to have two versions: a **start tag** and an **end tag**
- Notice that the Playground code editor will **automatically completed end tags for you** (this is a common feature found in many modern code IDEs)
- Notice that the `<canvas>` tag does not need anything in between the tag pair – that's fine it does not use text content
- The canvas did not appear to have changed the Playground mobile emulator… actually, it's just not visible yet!

6. It's easily make a canvas visible by changing its background colour, let's use some **CSS code** to do this:



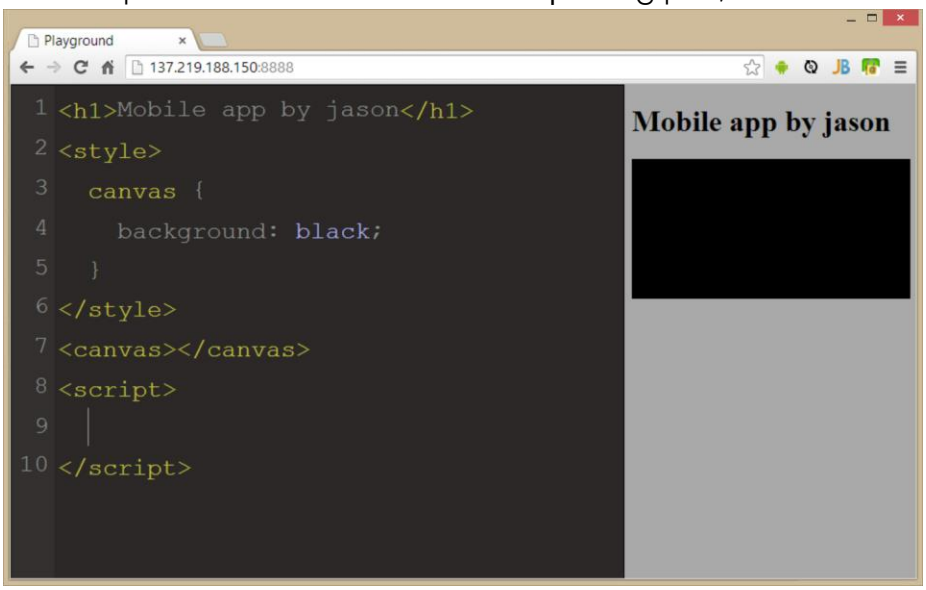
Things to note:

- Between the pair of <style> tags you are allowed to write CSS code
- As you can see, CSS **syntax** is very different to HTML syntax!
- Go ahead and try changing the colour of the background as you like!

PART 2 STEPS:

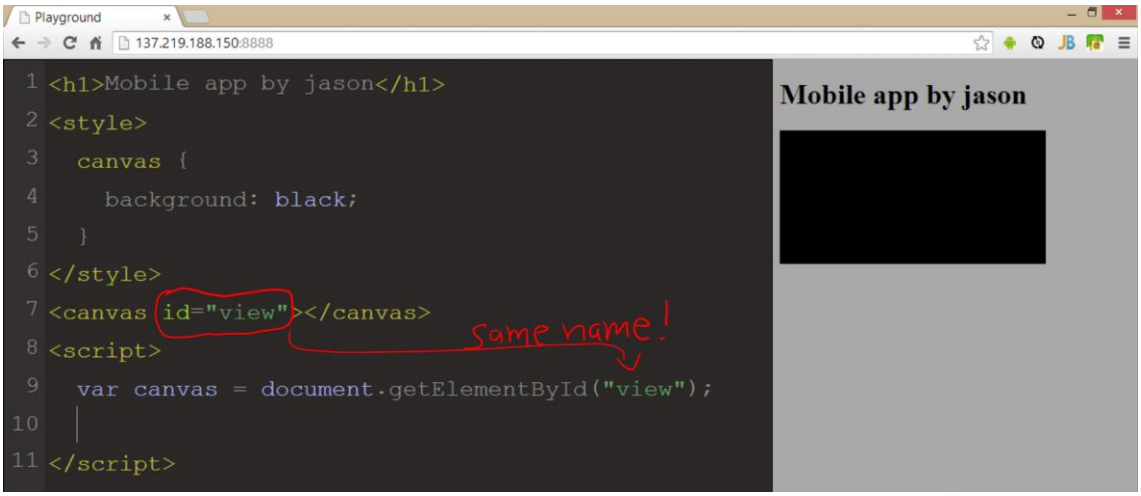
Next, we will use the canvas and some JavaScript code to draw some moving shapes and make our mobile web app come alive!

1. JavaScript code exists inside the `<script>` tag pair, as follows:



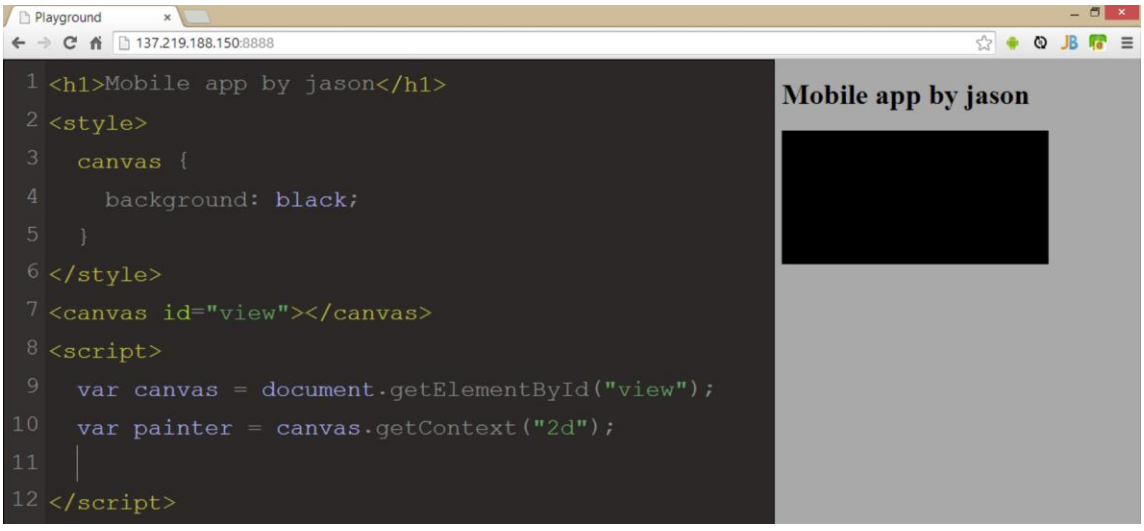
- Just like CSS code, JavaScript code only works inside a particular tag pair
- Each part of the program is separate from the other – the HTML code is separate from the CSS code, and so is the JavaScript code

2. To access and use the <canvas> tag inside your JavaScript code you need to create a **JavaScript variable** as follows:



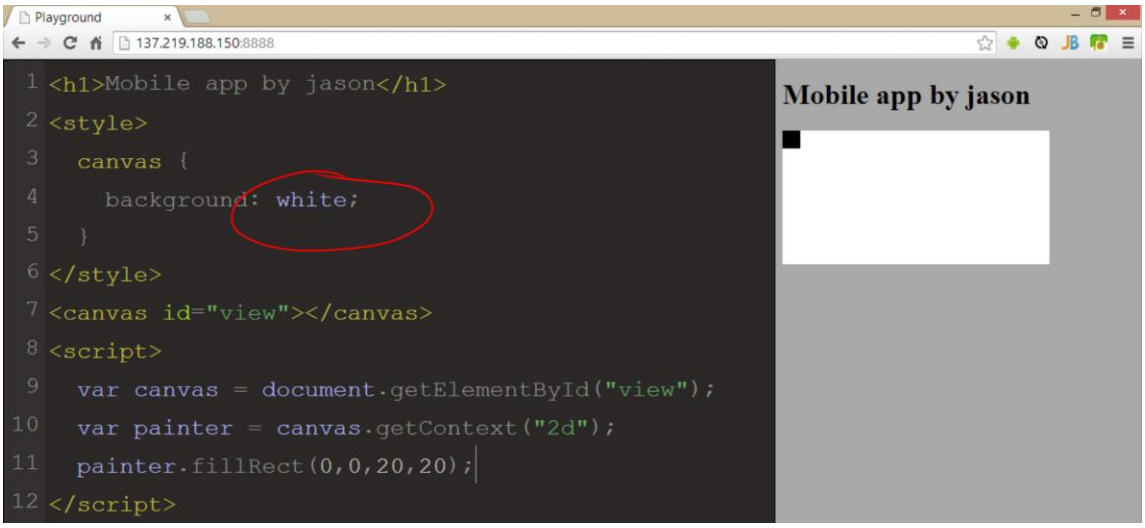
- A variable is like a bag - it can hold things for you
- Notice that we gave the canvas tag an **id attribute**
- The id attribute allows you to access the canvas using its **attribute value**

3. Now that we have access to the canvas, we can create the variable that allows you to draw onto the canvas:



- We call this drawing variable the "painter"
- Just like a real painter draws a painting onto a real canvas!
- In JavaScript different variables have access to different **functions**
- A function allows you to execute a task – the name of the function helps explain what the function will do

4. Adjust your code as follows:



- Note that you will need to change the CSS to make the canvas white!
- The painter variable has a fillRect() function that draws a rectangle at position x = 0, y = 0 of size width = 20, height = 20
- Try changing the position and size of your rectangle ☺

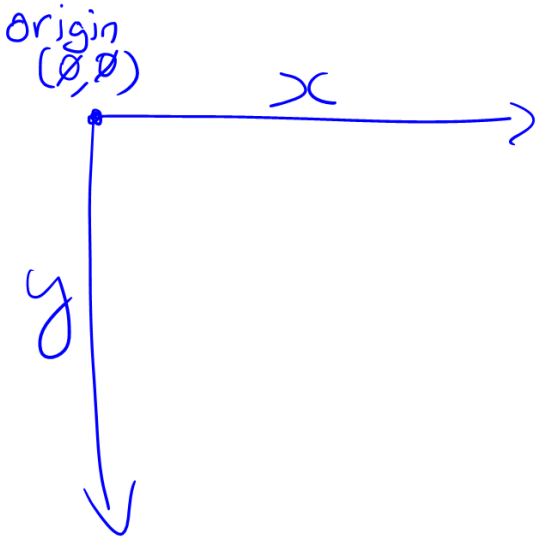
5. To change the colour of your rectangle adjust the code as follows:

```
8  <script>
9    var canvas = document.getElementById("view");
10   var painter = canvas.getContext("2d");
11   painter.fillStyle = "red";
12   painter.fillRect(0,0,20,20);
13 </script>
```

- `fillStyle` is not a function, it's a **property** of the painter variable
- Changing the value of a property will modify the variable

PART 3 STEPS:

To make the rectangle move around, we need to define some variables we can use to represent the position of the rectangle on the canvas. Here is how the canvas positions are defined:



1. First we create some new variables, and use them appropriately:

```
<script>
  var canvas = document.getElementById("view");
  var painter = canvas.getContext("2d");
  var x = 10, y = 20;
  painter.fillStyle = "red";
  painter.fillRect(x,y,20,20);
</script>
```

2. Next we need a function that will repeatedly draw the rectangle for us, in JavaScript we does this by using the `setInterval()` function:

```
<script>
  var canvas = document.getElementById("view");
  var painter = canvas.getContext("2d");
  var x = 10, y = 20;
  setInterval(function () {}, 1000);
  painter.fillStyle = "red";
  painter.fillRect(x,y,20,20);
</script>
```

3. Adjust the code by placing the two lines of painter code inside the curly braces:

```
<script>
  var canvas = document.getElementById("view");
  var painter = canvas.getContext("2d");
  var x = 10, y = 20;
  setInterval(function () {
    painter.fillStyle = "red";
    painter.fillRect(x,y,20,20);
  }, 1000);
</script>
```

   - This tells JavaScript to run the drawing code every 1000ms (milliseconds)
   - Be careful to place the painter code *inside the curly braces*!
   - How many seconds is 1000ms? Try different values and see what happens!

4. The rectangle will appear but not move yet. We need to change the values of the x and y variables inside the curly braces as well:

```
<script>
  var canvas = document.getElementById("view");
  var painter = canvas.getContext("2d");
  var x = 10, y = 20;
  setInterval(function () {
    x = x + 1;
    y = y + 1;
    painter.fillStyle = "red";
    painter.fillRect(x,y,20,20);
  }, 1000);
</script>
```

   - This code will now change the x and y variables by 1 and then draw the rectangle, every 1000ms
   - You'll notice that the rectangle leaves a trail – just like painter *moving a paint brush over the canvas*!
   - The painting will continue on forever – but we will only see what's drawn inside the boundary of the canvas

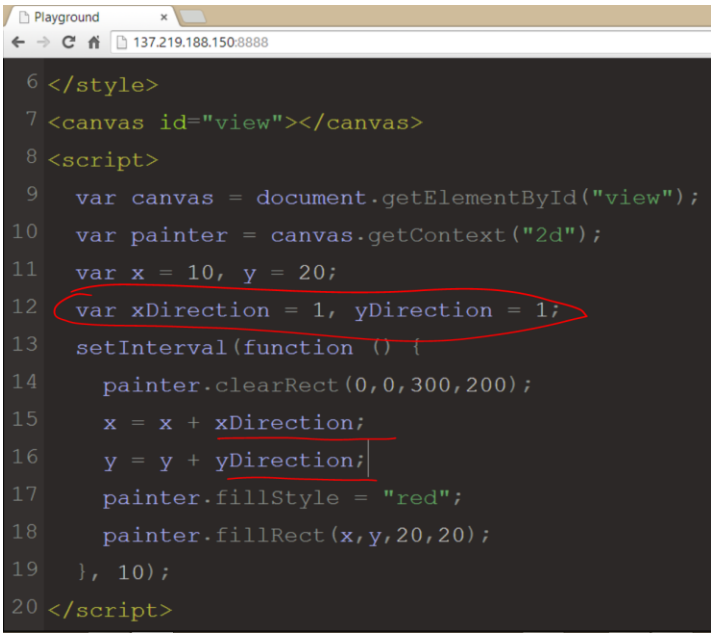
5. To complete the illusion of movement, adjust the code as follows:

```
setInterval(function () {
  painter.clearRect(0,0,300,200);
  x = x + 1;
  y = y + 1;
  painter.fillStyle = "red";
  painter.fillRect(x,y,20,20);
```

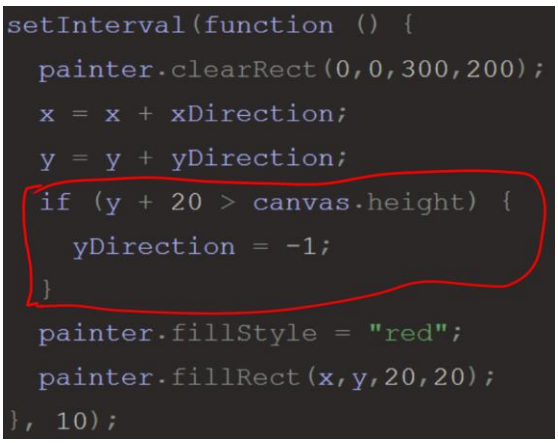
ADVANCED CODING

If you're interested, it is possible make the rectangle appear to bounce around the canvas and never disappear ☺

JavaScript has syntax for allowing the program to check when certain conditions occur. For this to work however, we need a few more variables:



You can make the rectangle appear to "bounce" off the bottom edge of the canvas by adjusting the code as follows:



The meaning of this code: if the bottom of the rectangle "hits" the bottom edge of the canvas reverse the y direction. Since the origin is at the top left corner of the canvas, positive y direction values move the rectangle down the screen, but negative values will move it up the screen! I wonder if you can figure out how to make the rectangle bounce off the remaining 3 edges of the canvas… ☺