Lecture 11
Bringing Out Your Inner Artist: The Canvas

Samkeun Kim <skim@hknu.ac.kr>

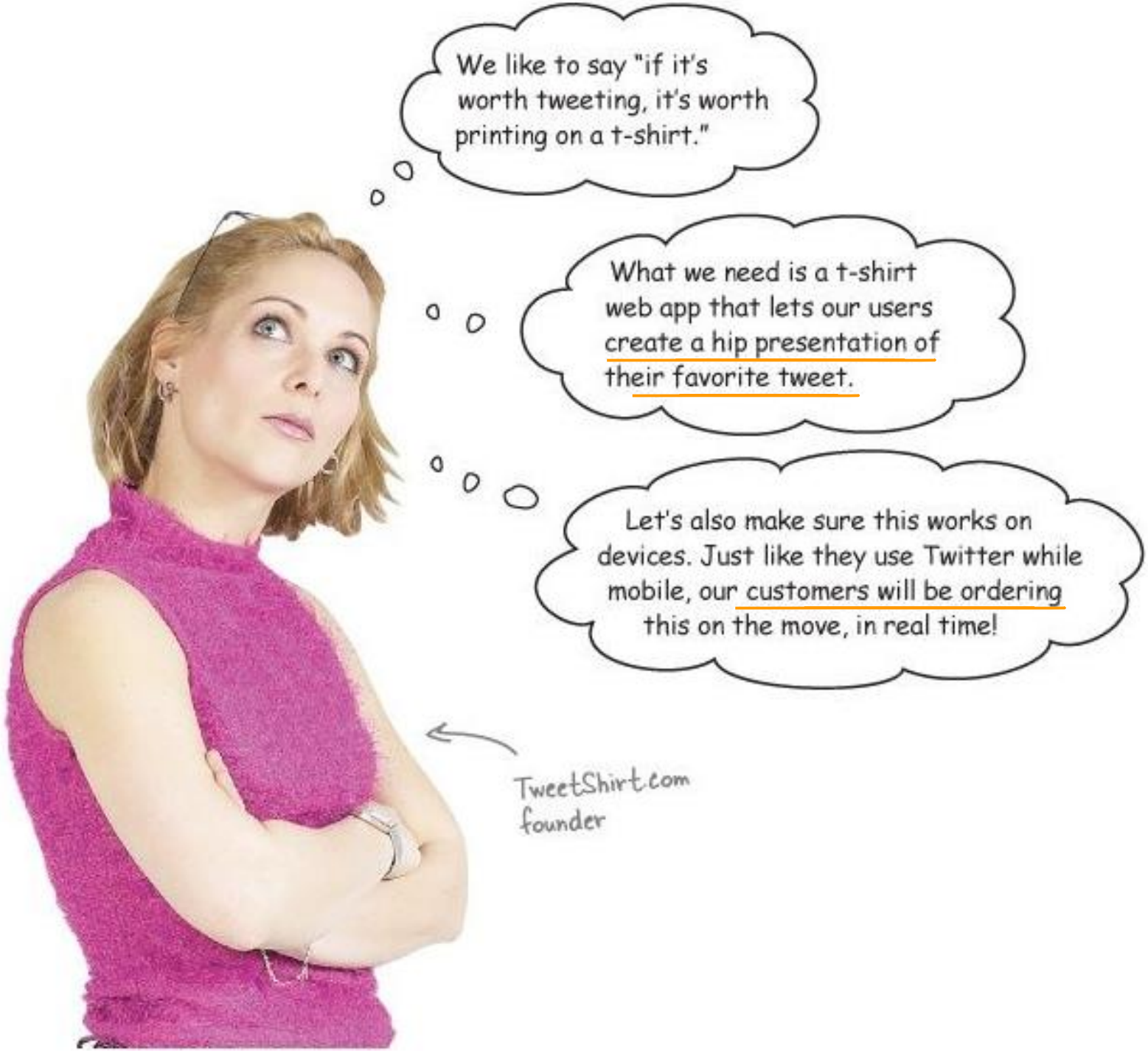
http://cyber.hknu.ac.kr/

국립한경대학교
HANKYONG NATIONAL UNIV.

Our new start-up: TweetShirt

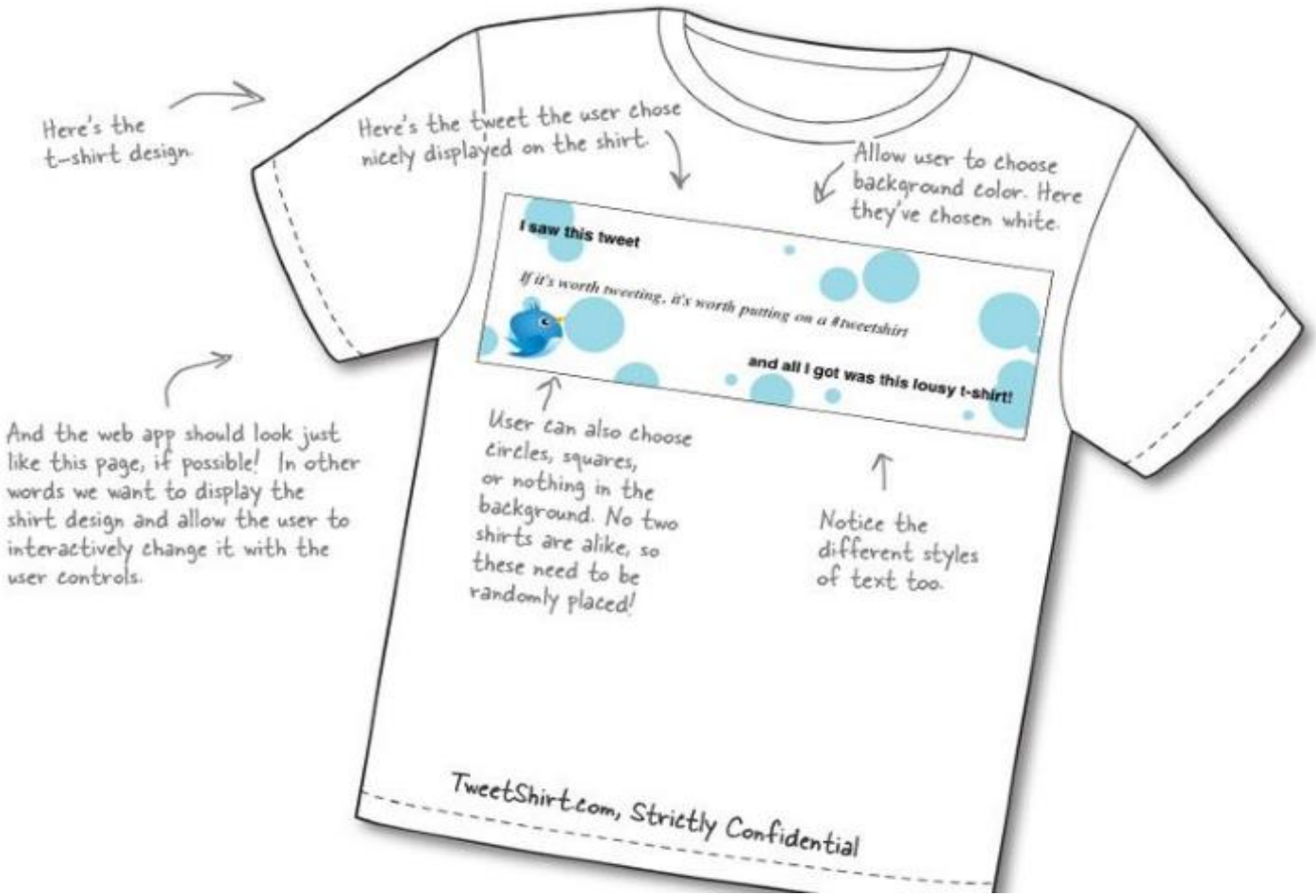
Our motto is "if it's worth tweeting on Twitter, it's worth wearing on a t-shirt."

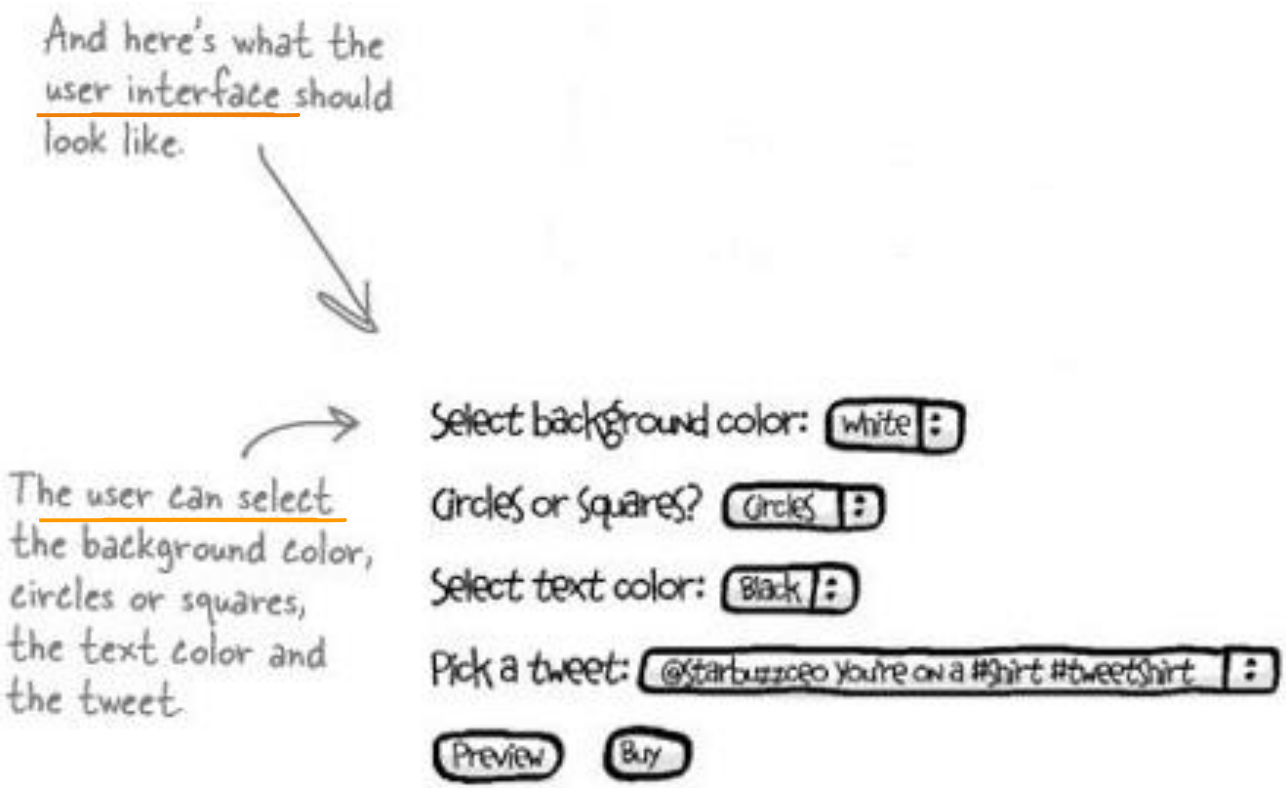
Now, there's only one thing that stands in the way of getting this start-up off the ground: we need **a nice web app that lets customers create a custom t-shirt design**, featuring one of their recent tweets.


We like to say "if it's worth tweeting, it's worth printing on a t-shirt."
What we need is a t-shirt web app that lets our users create a hip presentation of their favorite tweet.
Let's also make sure this works on devices. Just like they use Twitter while mobile, our customers will be ordering this on the move, in real time!
TweetShirt.com founder

Checking out the “comps”

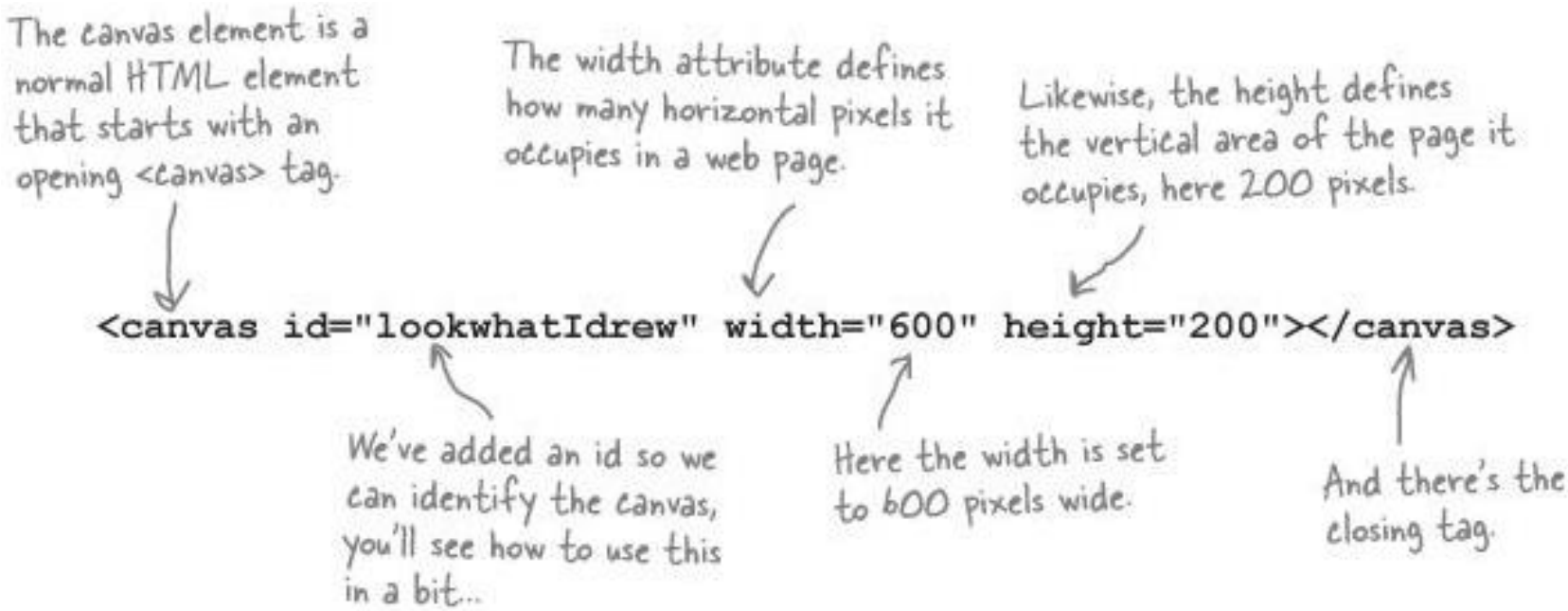
Initial visual design





How to get a canvas into your web page

In some ways a canvas is like an <img> element. You add a canvas like this:



How to see your canvas

If we use CSS to style the <canvas> element so we can see the border.
Let's add a simple style that adds a **1-pixel-wide black** border to the canvas.

```
<!doctype html>
<html lang="en">
<head>
    <title>Look What I Drew</title>
    <meta charset="utf-8">
    <style>
      canvas {
        border: 1px solid black;
      }
    </style>
</head>
<body>
<canvas id="lookwhatIdrew" width="600" height="200"></canvas>
</body>
</html>
```

We've added a style for the canvas that just puts a 1px black border on it, so we can see it in the page.

http://ksamkeun.dothome.co.kr/wp/hfhtml5/ch7/border.html

Drawing on the Canvas

Let's start with just our standard HTML5.

```
<!doctype html>
<html lang="en">
<head>
    <title>Look What I Drew</title>
    <meta charset="utf-8" />
    <style>
        canvas { border: 1px solid black; }
    </style>
    <script>
        window.onload = function() {
            var canvas = document.getElementById("tshirtCanvas");

            var context = canvas.getContext("2d");

            context.fillRect(10, 10, 100, 100);
        };
    </script>
</head>
<body>
    <canvas width="600" height="200" id="tshirtCanvas"></canvas>
</body>
</html>
```

We'll keep our CSS border in for now.

Here's our onload handler; we'll start drawing after the page is fully loaded.

To draw on the canvas we need a reference to it. Let's use getElementById to get it from the DOM.

Hmm, this is interesting, we apparently need a "2d" context from the canvas to actually draw....

We're using the 2d context to draw a filled rectangle on the canvas.

These numbers are the x, y position of the rectangle on the canvas.

And we've also got the width and height (in pixels).

Also interesting that a rectangle fill method doesn't take a fill color... more on that in a sec.

Ah, and we can't forget our canvas element. We're specifying a canvas that is 600 pixels wide and 200 pixels high, and has an id of "tshirtCanvas".

실습과제 11-1

Go ahead and type this code in and load it into your browser. Assuming you're using a modern browser you should see something like we do:

http://ksamkeun.dothome.co.kr/wp/hfhtml5/ch7/tshirtcanvas.html

A closer look at the code

1. 먼저 getElementById 메소드를 이용하여 canvas 객체에 대한 핸들을 얻는다.

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

2. 일종의 "protocol"인 컨텍스트를 얻어 온다. 여기서는 2D context.

```
var context = canvas.getContext("2d");
```

This is a bit of protocol we have to follow before we can start drawing on the canvas.

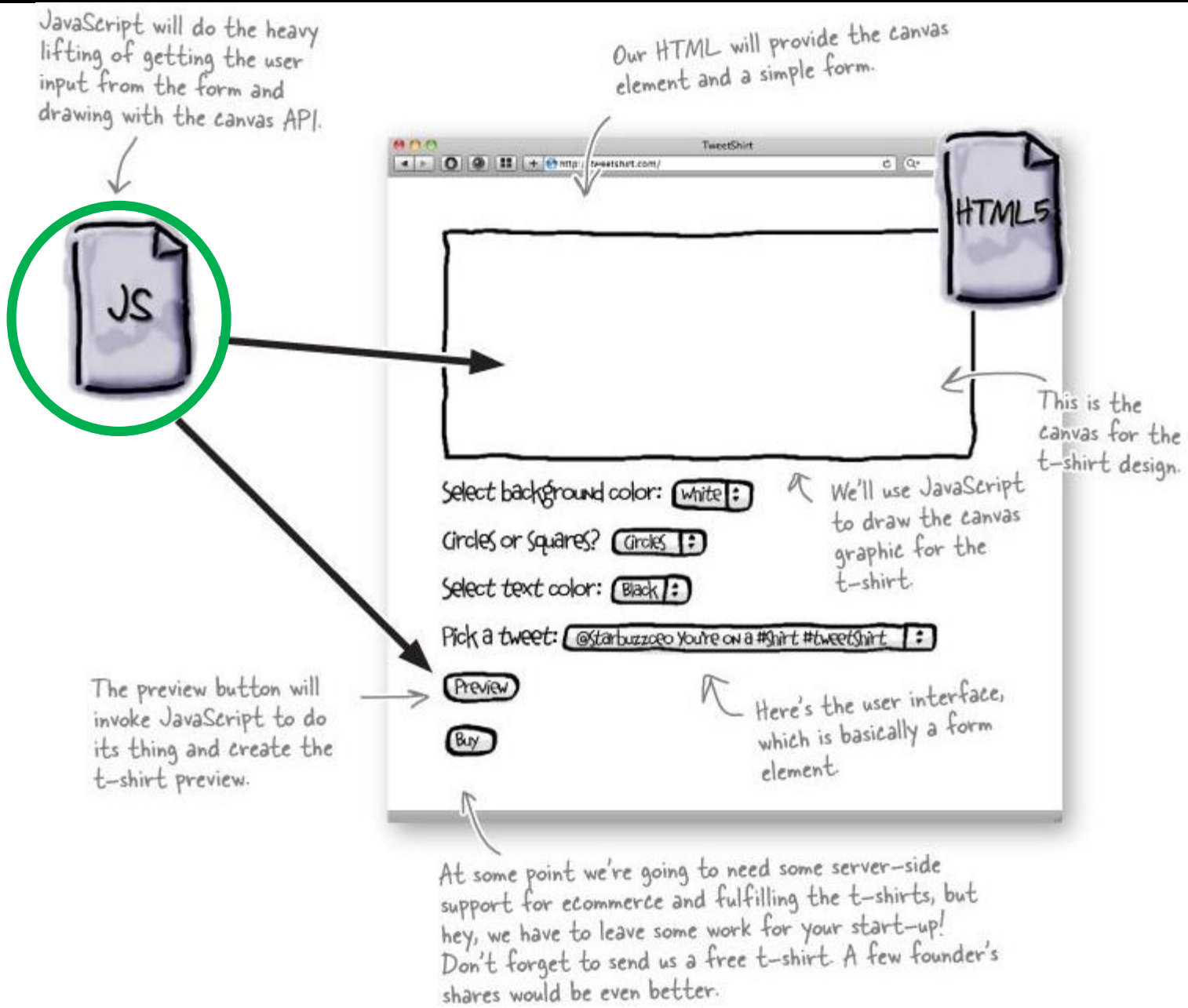
3. 이제 canvas에 fillRect 메소드를 이용하여 그림을 그릴 수 있다.
   (This method creates a rectangle starting at the x, y position of 10, 10 and that is 100 pixels wide and high.)

```
context.fillRect(10, 10, 100, 100);
```

Note, we're calling the fillRect method on the context, not the canvas itself.

Try this out and you should see a black rectangle appear. Try changing the values for x, y, width, and height and see what happens.

TweetShirt: the Big Picture



First, let's get the HTML in place

index.html

A nice HTML5-compliant file, yeah!

```
<!doctype html>
<html lang="en">
<head>
    <title>TweetShirt</title>
    <meta charset="utf-8" />
    <style>
        canvas {border: 1px solid black;}
    </style>
    <script src="tweetshirt.js"></script>
</head>
<body>
    <h1>TweetShirt</h1>
    <canvas width="600" height="200" id="tshirtCanvas">
        <p>Please upgrade your browser to use TweetShirt!</p>
    </canvas>
    <form>


    </form>
</body>
</html>
```

Notice, we changed the title to "TweetShirt"

Let's put all our JavaScript code in a separate file so it's a little easier to manage.

Here's the canvas!

And we've left a little message for users on old browsers.

And this is the form that will hold all the controls for the tweetshirt app. We'll get to this on the next page...

```
<form>
<p>
    <label for="backgroundColor">Select background color:</label>
    <select id="backgroundColor">
        <option value="white" selected="selected">White</option>
        <option value="black">Black</option>
    </select>
</p>
<p>
    <label for="shape">Circles or squares?</label>
    <select id="shape">
        <option value="none" selected="selected">Neither</option>
        <option value="circles">Circles</option>
        <option value="squares">Squares</option>
    </select>
</p>
<p>
    <label for="foregroundColor">Select text color:</label>
    <select id="foregroundColor">
        <option value="black" selected="selected">Black</option>
        <option value="white">White</option>
    </select>
</p>
<p>
    <label for="tweets">Pick a tweet:</label>
    <select id="tweets">
    </select>
</p>
<p>
    <input type="button" id="previewButton" value="Preview">
</p>
</form>
```

Here's where the user selects the background color for the tweet shirt design. The choices are black or white. Feel free to add your own colors.

We're using another selection control here for choosing circles or squares to customize the design. The user can also choose neither (which should result in a plain background).

Another selection for choosing the color of the text. Again, just black or white.

Here's where all the tweets go. So why's it empty? Ah, we'll be filling in that detail later (hint: we need to get them live from Twitter, after all this is a web app, right?!).

And last, a button to preview the shirt.

Time to get computational, with JavaScript

tweetshirt.js

Create a tweetshirt.js file and add this.

```
window.onload = function() {
        var button = document.getElementById("previewButton");
        button.onclick = previewHandler;
};
```

Start by getting the previewButton element.

Add a click handler to this button so that when it is clicked (or touched on a mobile device), the function previewHandler is called.

```
function previewHandler() {
    var canvas = document.getElementById("tshirtCanvas");
    var context = canvas.getContext("2d");

    var selectObj = document.getElementById("shape");
    var index = selectObj.selectedIndex;
    var shape = selectObj[index].value;

    if (shape == "squares") {
        for (var squares = 0; squares < 20; squares++) {
            drawSquare(canvas, context);
        }
    }
}
```

Start by getting the canvas element and asking for its 2d drawing context.

Now we need to see what shape you chose in the interface. First we get the element with the id of "shape".

Then we find out which item is selected (squares or circles) by getting the index of the selected item, and assigning its value to the variable shape.

And if the value of shape is "squares", then we need to draw some squares. How about 20 of them?

To draw each square we're relying on a new function named drawSquare, which we're going to have to write. Notice that we're passing both the canvas and the context to drawSquare. You'll see in a bit how we make use of those.

Writing the drawSquare function

Here's our function, which has two parameters: the canvas and the context.

We're using Math.random() to create random numbers for the width and the x,y position of the square. More on this in a moment...

```
function drawSquare(canvas, context) {
    var w = Math.floor(Math.random() * 40);

    var x = Math.floor(Math.random() * canvas.width);

    var y = Math.floor(Math.random() * canvas.height);

    context.fillStyle = "lightblue";
    context.fillRect(x, y, w, w);
}
```

We chose 40 as the largest square size so the squares don't get too big.

Here we need a random width, and x and y position for the square.

The x & y coordinates are based on the width and height of the canvas. We choose a random number between 0 and the width and height respectively.

We're going to make the squares a nice light blue using the fillStyle method, we'll look at this method more closely in a sec...

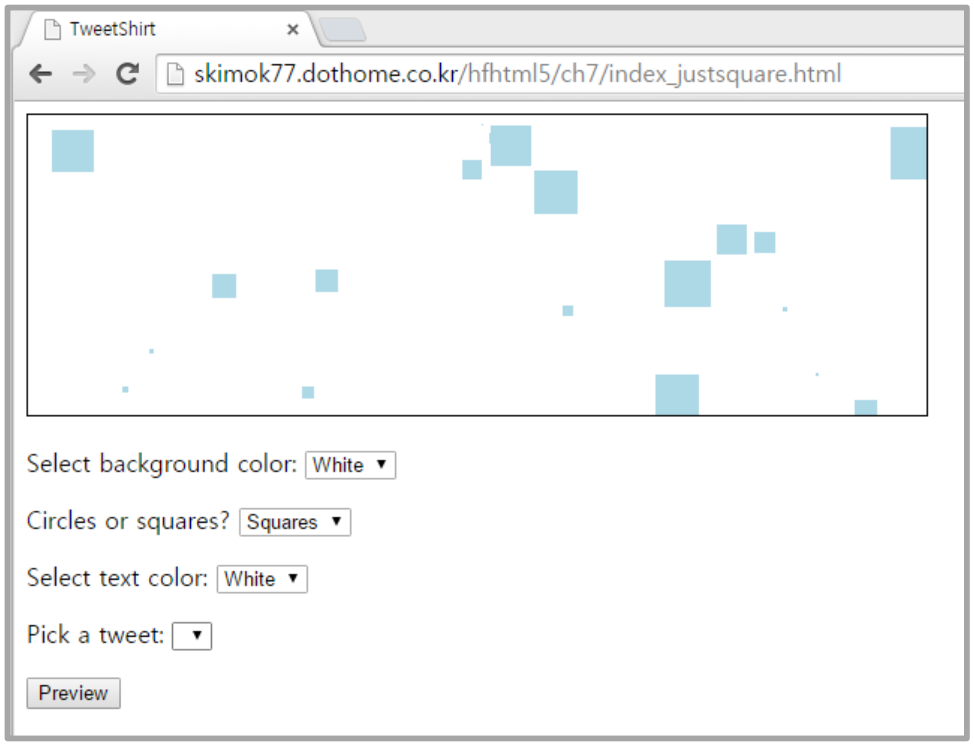
And finally, we draw the actual square with fillRect.

Head First HTML with CSS & XHTML has a good chapter on color if you need a refresher.

실습과제 11-2 Just squares

Okay, after all that typing, let's give all this code a test run. Go ahead and open your TweetShirt **index.html** file in your browser. Press preview and you should see random blue squares.

Here's what we see:



http://ksamkeun.dothome.co.kr/wp/hfhtml5/ch7/index_justsquare.html

Why are we seeing the old squares and the new squares when we preview?

http://ksamkeun.dothome.co.kr/wp/hfhtml5/ch7/index_justsquare.html

Add the call to fillBackgroundColor

fillBackgroundColor function:

- ✓ need to make sure we call it from previewHandler
- ✓ clean background before we start adding anything else to the canvas

```
function previewHandler() {
    var canvas = document.getElementById("tshirtCanvas");
    var context = canvas.getContext("2d");
    fillBackgroundColor(canvas, context);

    var selectObj = document.getElementById("shape");
    var index = selectObj.selectedIndex;
    var shape = selectObj[index].value;

    if (shape == "squares") {
        for (var squares = 0; squares < 20; squares++) {
            drawSquare(canvas, context);
        }
    }
}
```

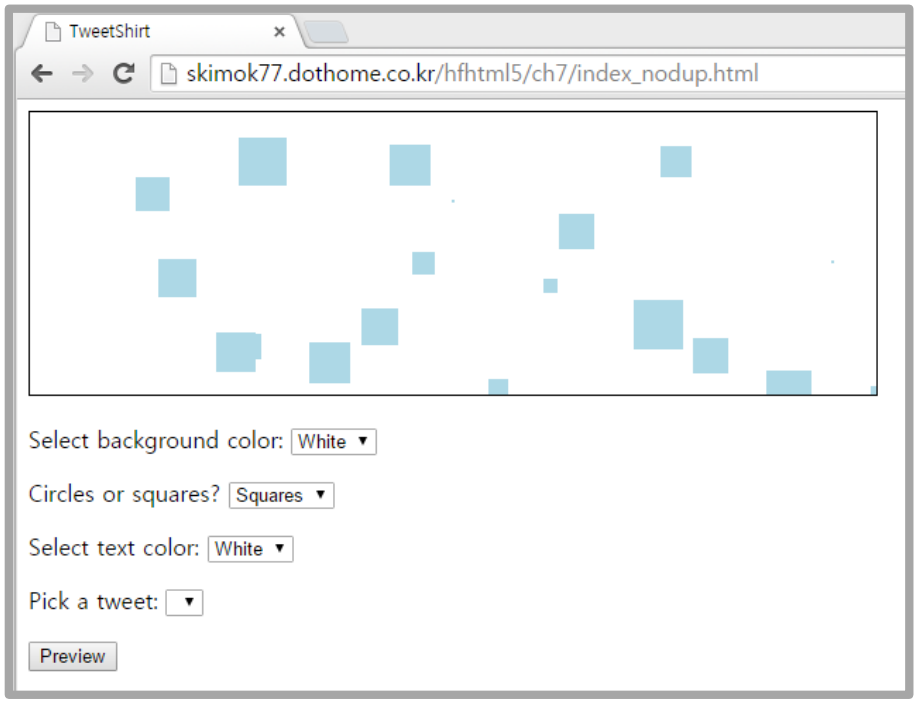
We're adding the call to fillBackgroundColor before we draw our squares so it covers up the previous drawing, and gives us a clean background for our new drawing.

```
// This is where we'll set the background color
function fillBackgroundColor(canvas, context) {
        var selectObj = document.getElementById("backgroundColor");
        var index = selectObj.selectedIndex;
        var bgColor = selectObj[index].value;

        context.fillStyle = bgColor;
        context.fillRect(0, 0, canvas.width, canvas.height);

}
```

실습과제 11-3

Another quick test drive to make sure our new fillBackgroundColor function works...

Add the new code to your tweetshirt.js file, reload your browser, select a background color, select squares, and click preview. Click it again. This time you should see only new squares each time you preview.



http://ksamkeun.dothome.co.kr/wp/hfhtml5/ch7/index_nodup.html

Drawing with Geeks

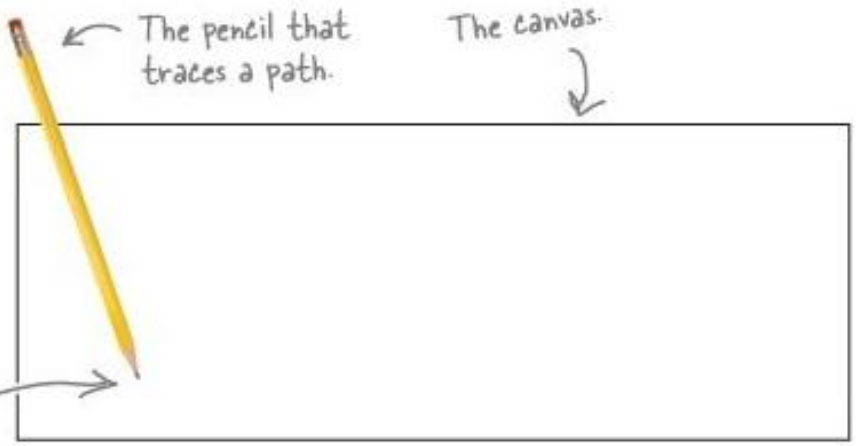
paths and arcs:
캔바스에 삼각형을 그려주는 fillTriangle 같은 메소드는 없다!

We use the beginPath method to tell the canvas we're starting a new path.

```
context.beginPath();
context.moveTo(100, 150);
```

We use the moveTo method to move the "pencil" to a specific point on the canvas. You can think of the pencil as being put down at this point.

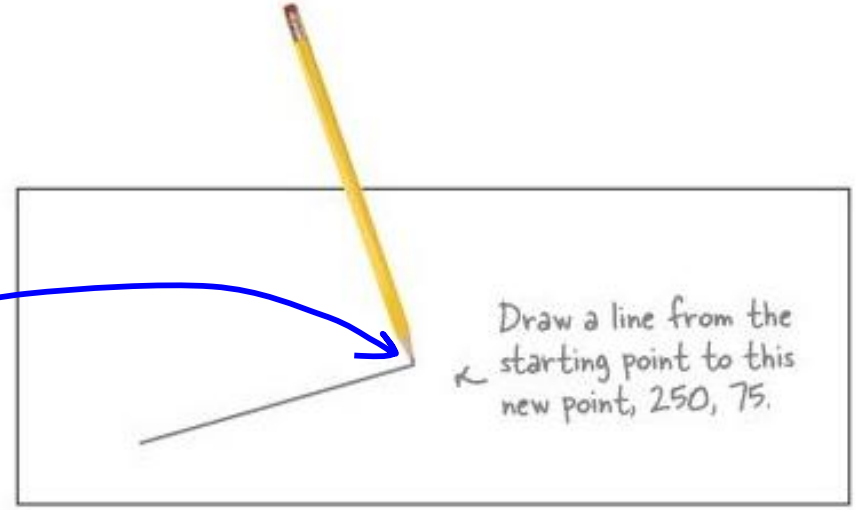
Here we're putting the pencil down at x = 100 and y = 150. This is the first point on the path.



The lineTo method traces a path from the pencil's current location to another point on the canvas.

```
context.lineTo(250, 75);
```

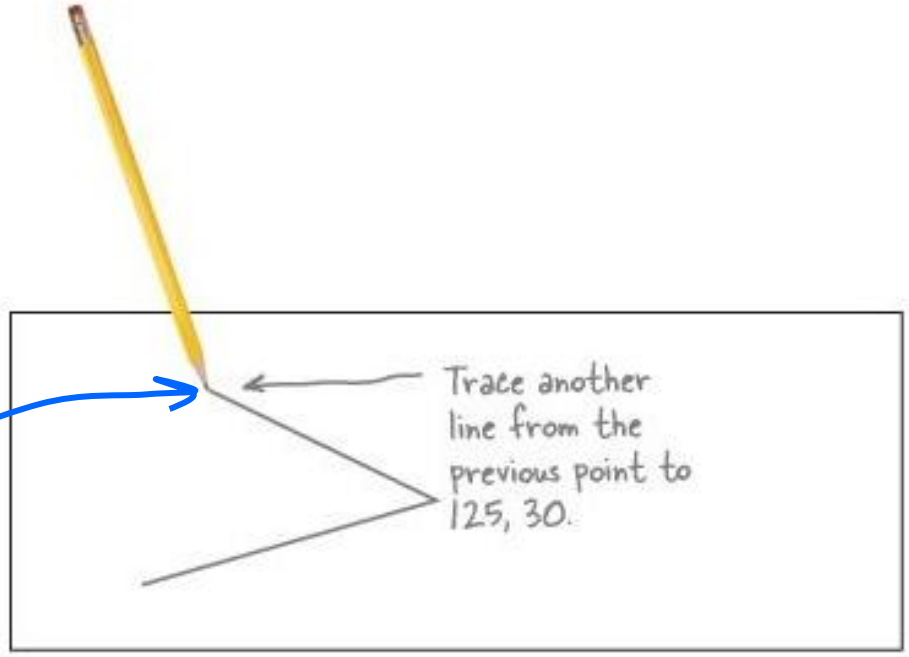
The pencil was at 100, 150, and here we're extending the path from there to the point x=250, y=75.



We've got the first side of the triangle, now we need two more. Let's use lineTo again for the second side:

```
context.lineTo(125, 30);
```

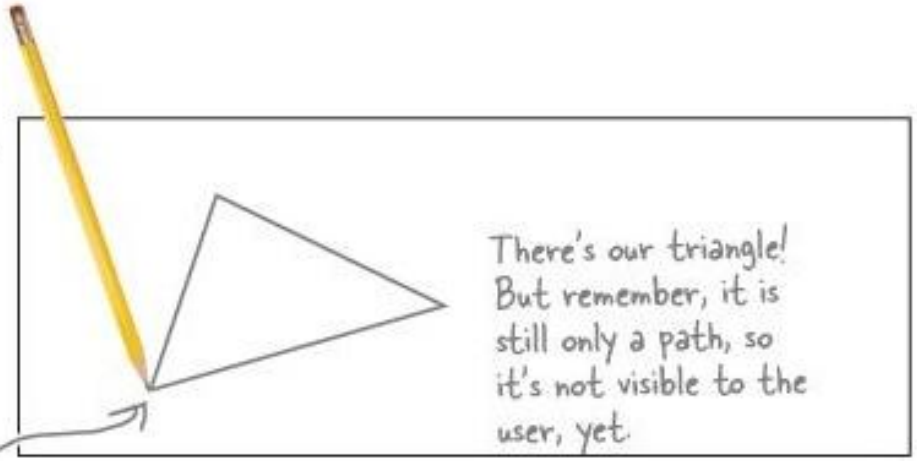
Here we're tracing from the current pencil position (250, 75) to a new position, x = 125, y = 30. That completes our second line.



We're almost there! All we need to do now is to trace one more line to finish the triangle. And to do that, we're just going to close the path with the closePath method.

```
context.closePath();
```

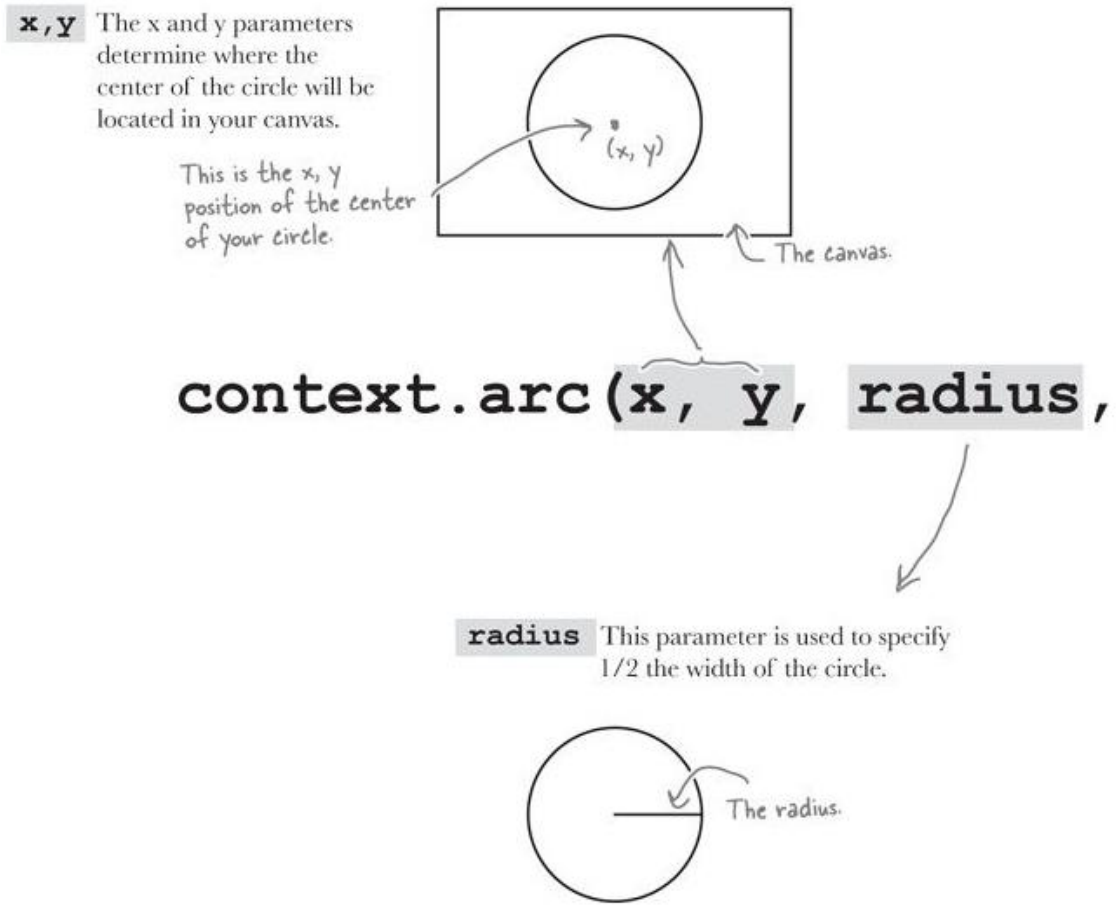
The closePath method connects the starting point of the path (100, 150) to the last point in the current path (125, 30).



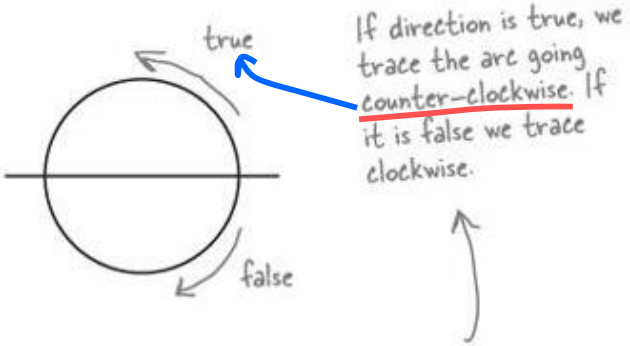
Breaking down the arc method

context.arc(x, y, radius, startangle, endangle, direction)

The whole point of the arc method is to specify how we want to trace a path along a circle.

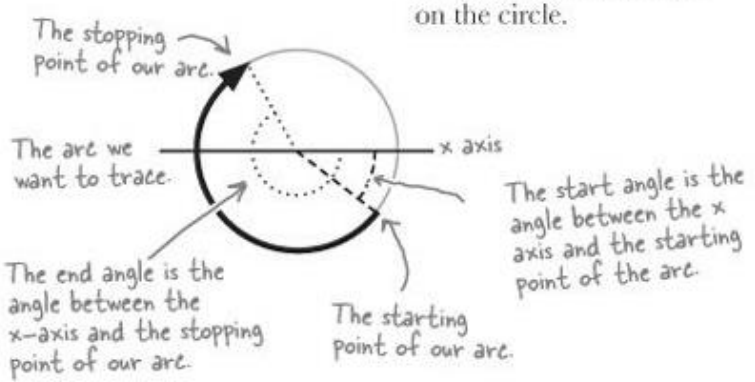


`direction` determines if we are creating the arc path in a counterclockwise or clockwise direction. If direction is true, we go counterclockwise; if it's false, we go clockwise.



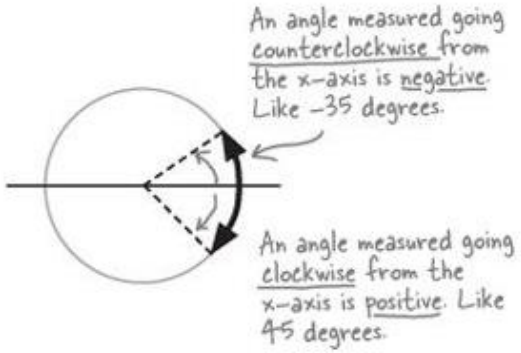
`startAngle, endAngle, direction)`

`startAngle, endAngle` The start angle and end angle of the arc determine where your arc path starts and stops on the circle.



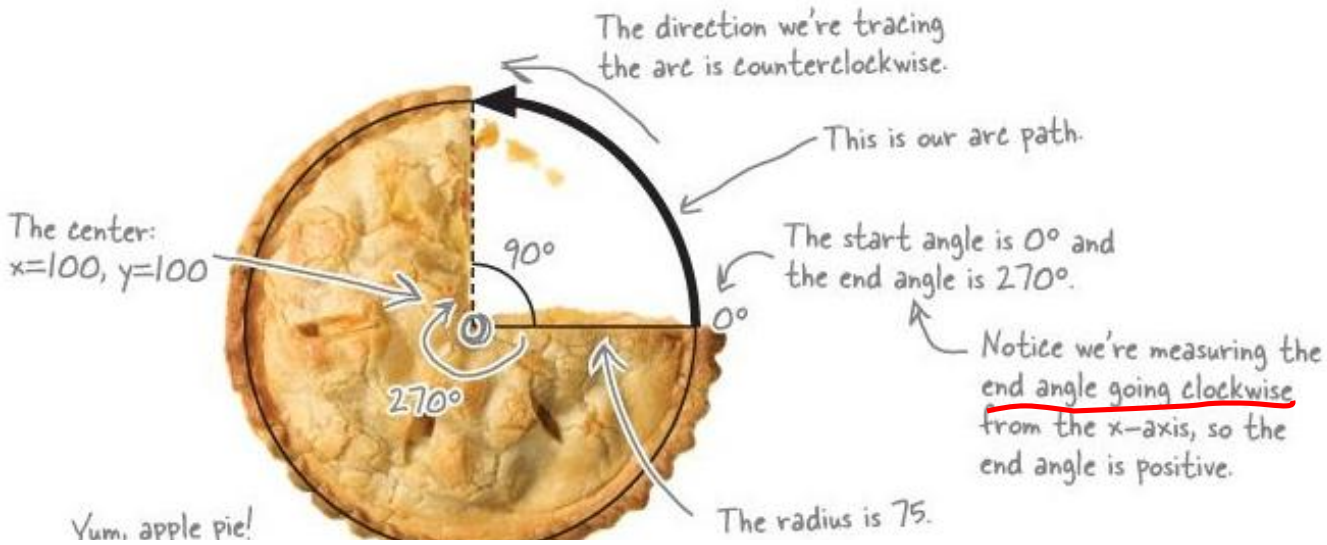
Important Point Below!

Don't skip this. Angles can be measured in the negative direction (counterclockwise from the x-axis) or in the positive direction (clockwise from the x-axis). This is *not* the same as the direction parameter for the arc path! (You'll see on the next page.)



A little taste of using the arc

Let's say that you want to trace an arc over a circle that is centered at x = 100, and y = 100 and you want the circle to be 150 pixels wide (or, a radius of 75). And, the path you want to trace is just 1/4 of the circle, like this:




```
context.arc(100, 100, 75, 0, degreesToRadians(270), true);
```

We'll come back to this in a sec. It just converts degrees (which we're used to), into radians (which the context seems to prefer).

we're tracing the arc in a *counterclockwise* direction, so we use *true*.

I say degree, you say radian

We think in **degrees**, but the canvas context thinks in **radians**.

```
360 degrees = 2Pi radians
```

```
function degreesToRadians(degrees) {
    return (degrees * Math.PI)/180;
}
```

Use this function whenever you want to think in degrees, but get radians for drawing an arc.

You might remember seeing this briefly in the Geolocation chapter.

To get radians from degrees, you multiply by π and divide by 180.

On page 313, you saw us use 2*Math.PI to specify the end angle of a circle. You could do that... or just use degreesToRadians(360).

Back to writing the TweetShirt circle code

Edit your **tweetshirt.js** file and add the new code below.

```
function previewHandler() {
    var canvas = document.getElementById("tshirtCanvas");
    var context = canvas.getContext("2d");
    fillBackgroundColor(canvas, context);

    var selectObj = document.getElementById("shape");
    var index = selectObj.selectedIndex;
    var shape = selectObj[index].value;

    if (shape == "squares") {
        for (var squares = 0; squares < 20; squares++) {
            drawSquare(canvas, context);
        }
    } else if (shape == "circles") {
        for (var circles = 0; circles < 20; circles++) {
            drawCircle(canvas, context);
        }
    }
}
```

This code looks almost identical to the code to test for squares. If the user has chosen circles rather than squares then we draw 20 circles with the drawCircle function (which we now need to write).

We're passing the canvas and context to the drawCircle function, just like we did with drawSquares.

Writing the drawCircle function...

Now let's write the drawCircle function. Remember, here we just need to draw one random circle. The other code is already handling calling this function 20 times.

```
function drawCircle(canvas, context) {
    var radius = Math.floor(Math.random() * 40);
    var x = Math.floor(Math.random() * canvas.width);
    var y = Math.floor(Math.random() * canvas.height);

    context.beginPath();
    context.arc(x, y, radius, 0, degreesToRadians(360), true);

    context.fillStyle = "lightblue";
    context.fill();
}
```

Just like we did for squares, we're using 40 for the maximum radius size to keep our circles from getting too big.

And, again, the x & y coordinates of the center of the circle are based on the width and height of the canvas. We choose random numbers between 0 and the width and height respectively.

We use an end angle of 360° to get a full circle. We draw counterclockwise around the circle, but for a circle, it doesn't matter which direction we use.

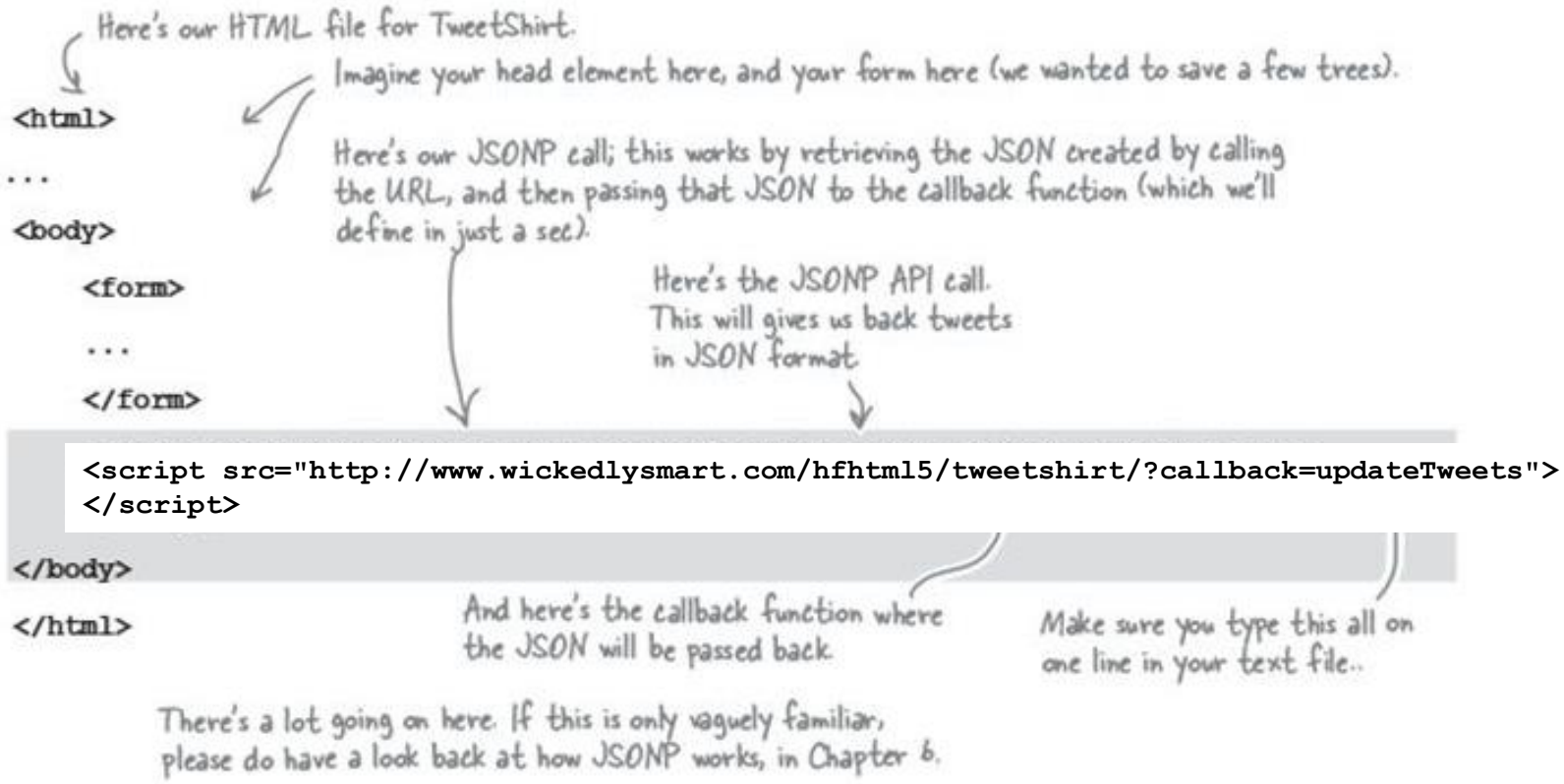
We're using "lightblue" as our fillStyle again, and then filling the path with context.fill().

실습과제 11-4

http://ksamkeun.dothome.co.kr/wp/hfhtml5/ch7/index_pluscircle.html

Welcome back...

1. Add a <script> at the bottom of the **tweetshirt.html** file to make a call to the Twitter JSONP API.
2. Implement a callback to get the tweets that Twitter sends back. We'll use the name of this callback in the URL for the <script> in Step 1.



https://www.wickedlysmart.com/hfhtml5/tweetshirt/?callback=updateTweets

Getting your tweets

Edit your **tweetshirt.js** file and add the updateTweets function at the bottom. Here's the code:

Here's our callback.

Which is passed a response containing the tweets as an array of tweets.

We grab a reference to the tweets selection from the form.

```
function updateTweets(tweets) {
    var tweetsSelection = document.getElementById("tweets");

    for (var i = 0; i < tweets.length; i++) {
        tweet = tweets[i];
        var option = document.createElement("option");
        option.text = tweet.text;
        option.value = tweet.text.replace("\"", "'");

        tweetsSelection.options.add(option);
    }

    tweetsSelection.selectedIndex = 0;
}
```

For each tweet in the array of tweets, we:

Get a tweet from the array.

Create a new option element.

Set its text to the tweet.

And set its value to the same text, only we've processed the string a little to replace double quotes with single quotes (so we can avoid formatting issues in the HTML).

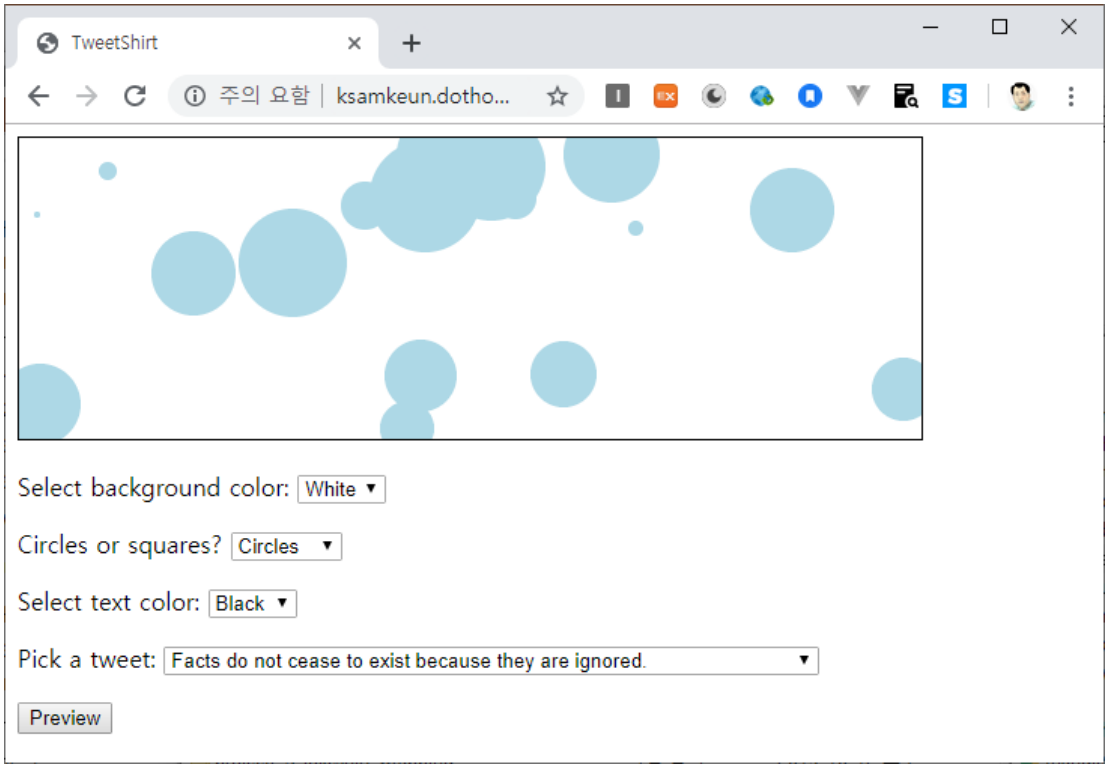
We then take the new option, and add it to the tweet selection in the form.

After we've done this for each tweet, we have a <select> element that contains an option for each tweet.

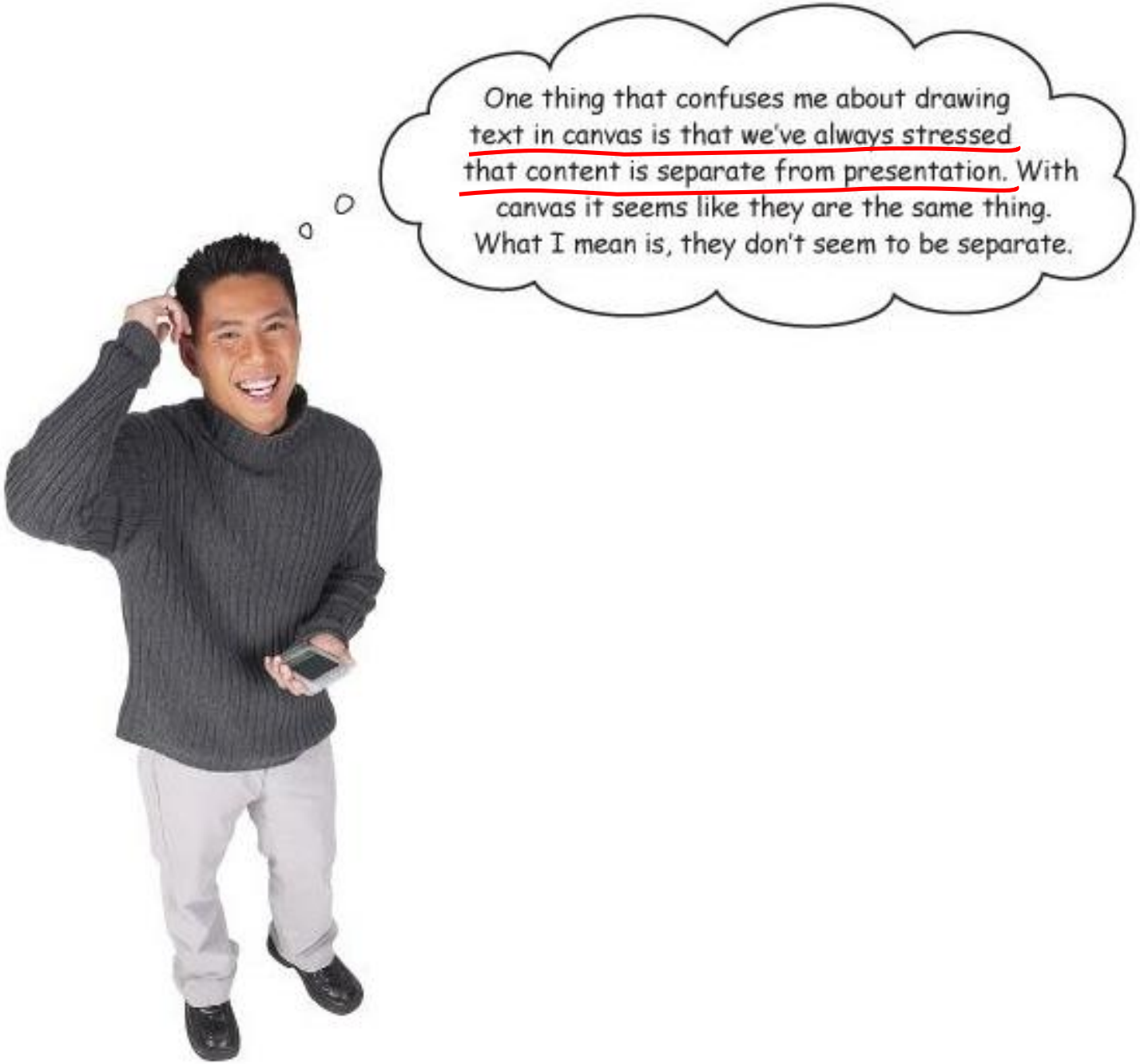
And, finally, we make sure the first tweet is the selected tweet by setting the selectedIndex of the <select> to 0 (the first option element contained within it).

실습과제 11-5

Let's do a quick test drive. Make sure you've got all the code added to **tweetshirt.js** and index.html. Also make sure you're using a Twitter username that has recent tweets in your script src URL (so you'll be sure you see some tweets!). Load the page and click on the tweets selection. Here's what we see:



http://ksamkeun.dothome.co.kr/wp/hfhtml5/ch7/index_jsonp.html


One thing that confuses me about drawing text in canvas is that we've always stressed that content is separate from presentation. With canvas it seems like they are the same thing. What I mean is, they don't seem to be separate.

That's a really good point.

Now let's work through why it's set up this way. Remember that **canvas is designed to give you a way to present graphics within the browser. Everything in the canvas is considered presentation, not content.** So while you usually think of text — and certainly tweets — as content, in this case, you've got to think of it as presentation. It's part of the design. Like an artist who uses letterforms as part of her artwork, you're using tweets as part of the artwork of your t-shirt design.

One of the main reasons that separating presentation and content is a good idea is so that the browser can be smart about how it presents the content in different situations: for example, an article from a news web site is presented one way on a big screen and a different way on your phone.

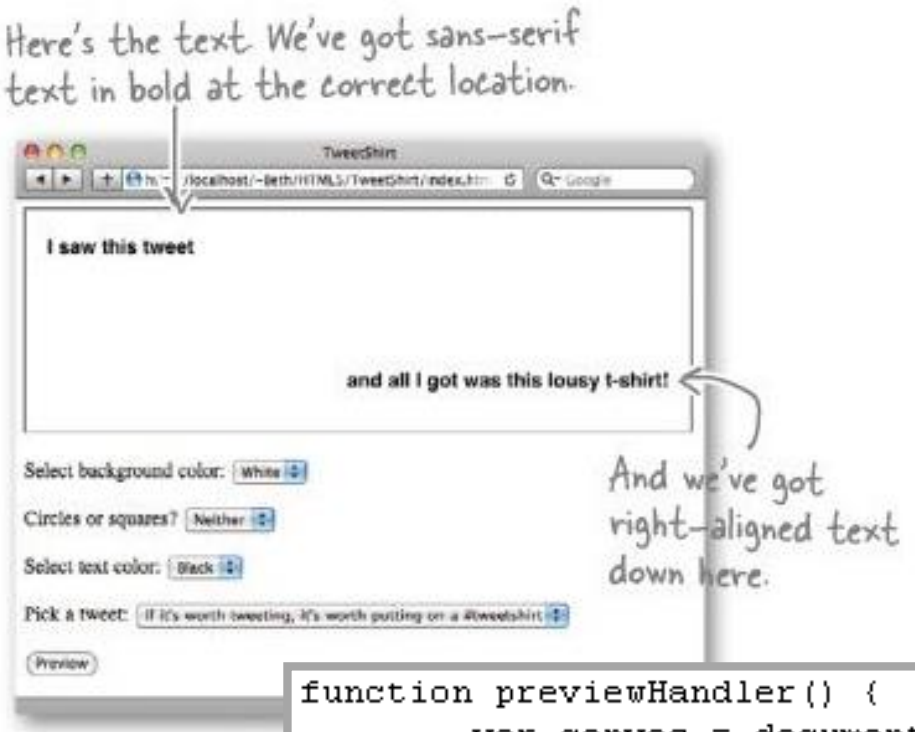
Giving drawText a spin

Now that you've got more of the API in your head, go ahead and get the code you created in the Magnet Code exercise typed in — here it is with the magnets nicely translated to code:

```
function drawText(canvas, context) {
    var selectObj = document.getElementById("foregroundColor");
    var index = selectObj.selectedIndex;
    var fgColor = selectObj[index].value;

    context.fillStyle = fgColor;
    context.font = "bold 1em sans-serif";
    context.textAlign = "left";
    context.fillText("I saw this tweet", 20, 40);

    context.font = "bold 1em sans-serif";
    context.textAlign = "right";
    context.fillText("and all I got was this lousy t-shirt!",
        canvas.width-20, canvas.height-40);
}
```

We're going to put the code that draws the tweet text here in a sec...

After you've got it typed in, update your previewHandler function to call the drawText function, and give it a test drive by loading it in your browser. You should see something like we do:




```
function previewHandler() {
        var canvas = document.getElementById("tshirtCanvas");
        var context = canvas.getContext("2d");
        .
        .
        .

        drawText(canvas, context);
}
```

Completing the drawText function

How does it compare to yours? If you haven't already typed your code in, go ahead and type in the code below (or your version if you prefer), and reload your **index.html**.

```
function drawText(canvas, context) {
    var selectObj = document.getElementById("foregroundColor");
    var index = selectObj.selectedIndex;
    var fgColor = selectObj[index].value;

    context.fillStyle = fgColor;
    context.font = "bold 1em sans-serif";
    context.textAlign = "left";
    context.fillText("I saw this tweet", 20, 40);

    selectObj = document.getElementById("tweets");
    index = selectObj.selectedIndex;
    var tweet = selectObj[index].value;
    context.font = "italic 1.2em serif";
    context.fillText(tweet, 30, 100);

    context.font = "bold 1em sans-serif";
    context.textAlign = "right";
    context.fillText("and all I got was this lousy t-shirt!",
        canvas.width-20, canvas.height-40);
}
```

We don't need to align the tweet text to the left; the alignment is still set from up here.

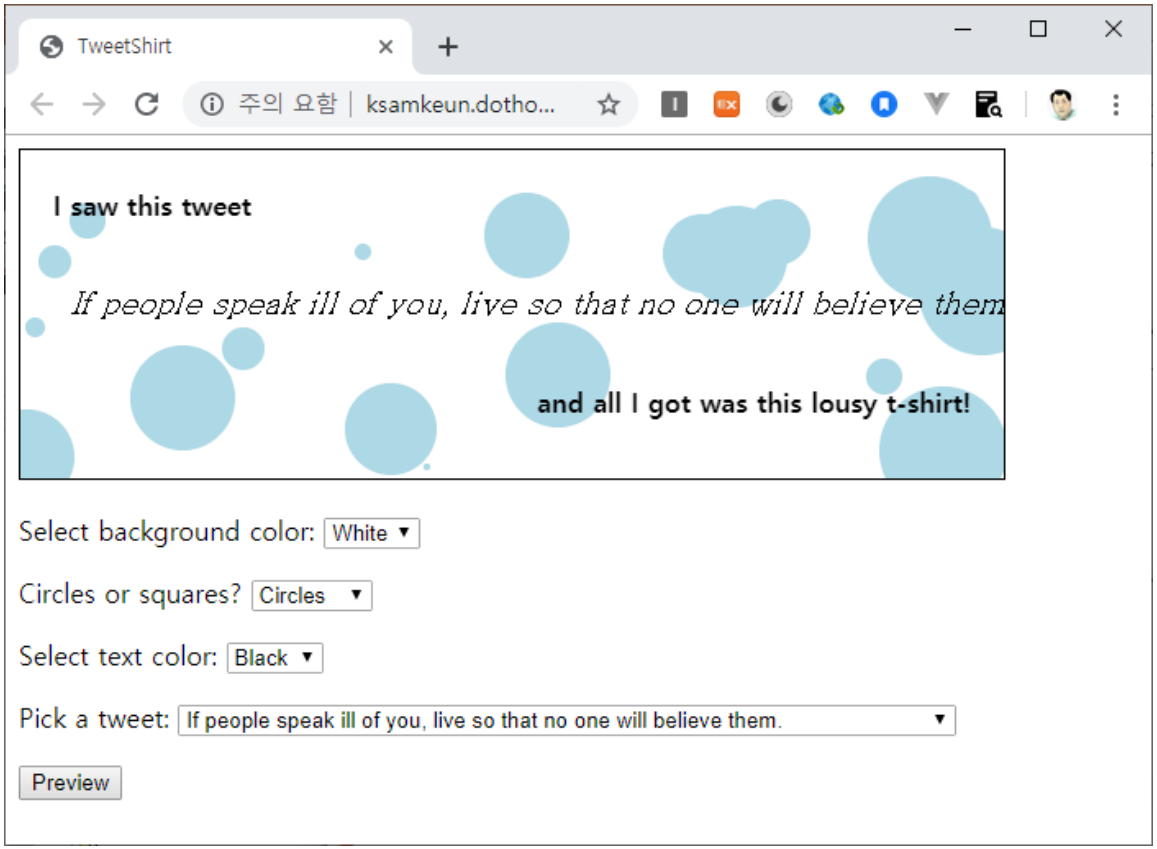
We grab the selected option from the tweet menu.

Set the font to italic serif, just a tad bigger...

... and draw it at position 30, 100.

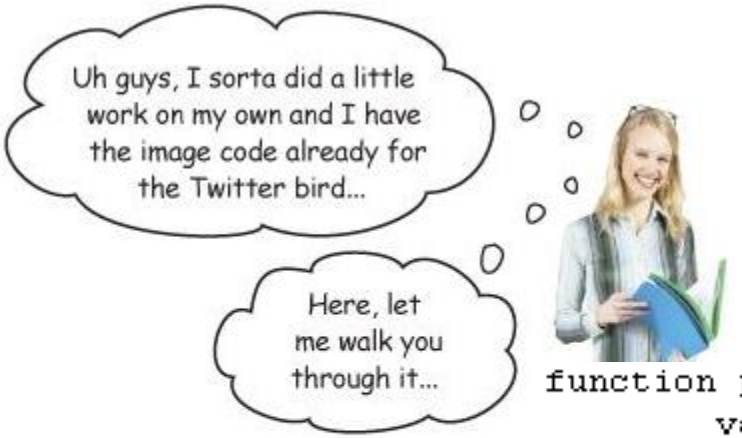
실습과제 11-6

We hope you're seeing what we're seeing! Nice huh? Give the interface a real bit of quality assurance testing: try all the combinations of colors and shapes, or swap out the username for another you like.



http://ksamkeun.dothome.co.kr/wp/hfhtml5/ch7/index_drawtext.html

Feel like you're ready to launch this for real? Let's do it!




```
function previewHandler() {
        var canvas = document.getElementById("tshirtCanvas");
        var context = canvas.getContext("2d");
        .
        .
        .

        drawText(canvas, context);
        drawBird(canvas, context);
}
```

1. **twitterBird.png** 를 추가하자.
   To get that into the canvas we first need a JavaScript image object.
   Here's how we get one:

```
var twitterBird = new Image();
twitterBird.src = "twitterBird.png";
```

← Create a new image object.

← And set its source to be the image of the Twitter bird.

2. Context 메소드를 사용하여 canvas에 이미지를 그려보자: drawImage

```
context.drawImage(twitterBird, 20, 120, 70, 70);
```

Using the drawImage method

Here's our image object.

And we specify the x, y location, width and height

3. Images don't always load immediately -> onload handler

```
twitterBird.onload = function() {
    context.drawImage(twitterBird, 20, 120, 70, 70);
};
```

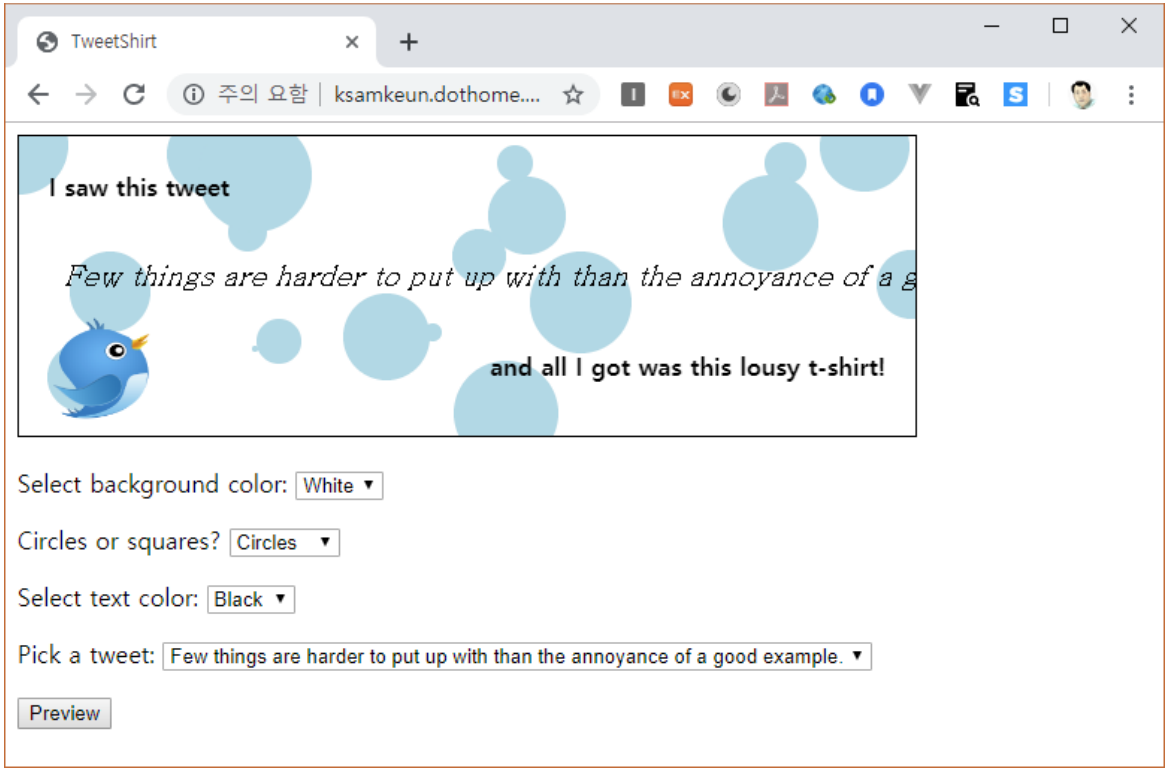
Here, we're saying: when the image has loaded, then execute this function.

We draw the image to the canvas using the context's drawImage method.

```
// draws the twitter bird image
function drawBird(canvas, context) {
        var twitterBird = new Image();
        twitterBird.src = "twitterBird.png";
        twitterBird.onload = function() {
                context.drawImage(twitterBird, 20, 120, 70, 70);
        };

}
```

실습과제 11-7

Give it a few tries; try it with circles or squares. You'll notice that we used a **png** with a transparent background so that the circles and squares work if they're behind the bird.



http://ksamkeun.dothome.co.kr/wp/hfhtml5/ch7/index_bird.html

http://ksamkeun.dothome.co.kr/wp/hfhtml5/ch7/twitterBird.png

Q & A