Homework 3 - Colors
Solution

1. **Fill in the blanks below to convert colors between the RGB, HSV, and CMY models:**

	Red	Green	Blue	Hue°	Saturation%	Value%	Cyan	Magenta	Yellow
(a)	0	0	0	__	0	__	__	__	__
(b)	255	255	0	__	100	100	__	__	__
(c)	0	0	127.5	__	__	__	1	1	__
(d)	255	255	255	__	__	__	__	__	__
(e)	0	255	255	__	100	100	1	0	0
(f)	__	__	__	0	100	100	__	__	__
(g)	__	__	__	180	0	100	__	__	0
(h)	__	__	__	300	50	0	__	__	1

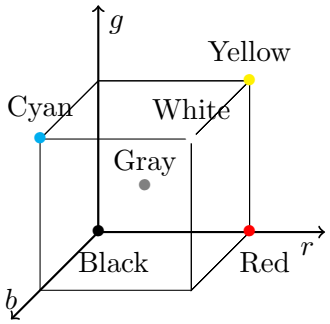
#	Red	Green	Blue	Hue°	Saturation%	Value%	Cyan	Magenta	Yellow
(a)	0	0	0	Any	0	0	1	1	1
(b)	255	255	0	60	100	100	0	0	1
(c)	0	0	127.5	240	100	50	1	1	0.5
(d)	255	255	255	Any	0	100	0	0	0
(e)	0	255	255	180	100	100	1	0	0
(f)	255	0	0	0	100	100	0	1	1
(g)	255	255	255	180	0	100	0	0	0
(h)	0	0	0	300	50	0	1	1	1

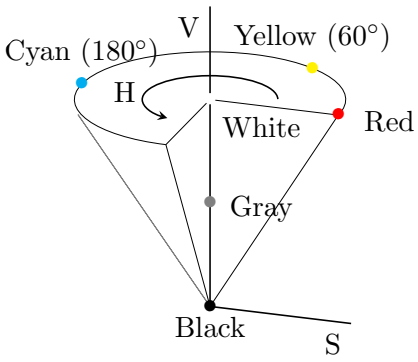
2. **Given a color c_1 = RGB(25,127,76), which color c_2 can you add to get a color pink? Define (using the RGB model) the pink you are aiming and then the color you added.**

   If we define pink as RGB(255,127,127).

$$
\begin{aligned}
RGB(25,127,76) + RGB(r,g,b) &= RGB(255,127,127) \\
RGB(r,g,b) &= RGB(255,127,127) - RGB(25,127,76) \\
RGB(r,g,b) &= RGB(255-25,127-127,127-76) \\
RGB(r,g,b) &= RGB(230,0,51)
\end{aligned}
$$

3. **Sketch the color models for RGB (cube) and HSV (cone). In both model sketches, mark (with color name) the position of each of the following colors: black, white, gray, red, cyan, and yellow.**





4. **Given the color c_1 = RGB(255,255,0), if you decrease the "saturation" as defined in HSV until fully desaturated, describe the range of RGB colors you will pass through, as well as the final result.**

   The color starts at pure yellow, and to understand saturation's impact better, we convert the color to HSV. RGB(255,255,0) = HSV(60°,100%,100%). As we decrease saturation (the second value), the color gradually becomes lighter resulting in white in the end HSV(60°,0%,100%) = RGB(255,255,255). We can also see, in RGB that the third value (Blue) results in 255 due to saturation decrease. Hence, saturation decrease results in purely an increase in blue.

5. **Linearly interpolate the color halfway between red and blue in each of the following color models. (Requires a separate calculation for each subproblem, not just converting the resulting color.)**

   (a) RGB

   A — P — B

   (0,0) (0.5,0) (1,0)

   RGB(255, 0, 0) RGB(0, 0, 255)

$$\frac{(0.5-0)}{1-0} * [255,0,0] + \frac{(1-0.5)}{1-0} * [0,0,255]$$
$$= 0.5 * [255,0,0] + 0.5 * [0,0,255]$$
$$= [127.5,0,0] + [0,0,127.5]$$
$$= [127.5,0,127.5]$$
$$= RGB(128,0,128) = DarkMagenta$$

   (b) HSV

   A — P — B

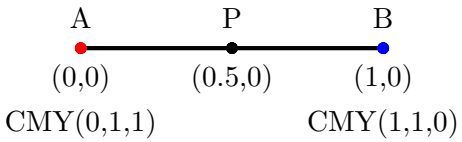
   (0,0) (0.5,0) (1,0)

   HSV(0°,100%,100%) HSV(240°,100%,100%)

$$\frac{(1-0.5)}{1-0} * [0,100,100] + \frac{(0.5-0)}{1-0} * [240,100,100]$$

$$= 0.5 * [0, 100, 100] + 0.5 * [240, 100, 100]$$
$$= [0, 50, 50] + [120, 50, 50]$$
$$= [120, 100, 100]$$
$$= HSV(120°, 100\%, 100\%) = Green$$

(c) CMY



$$\frac{(1-0.5)}{1-0} * [0, 1, 1] + \frac{(0.5-0)}{1-0} * [1, 1, 0]$$
$$= 0.5 * [0, 1, 1] + 0.5 * [1, 1, 0]$$
$$= [0, 0.5, 0.5] + [0.5, 0.5, 0]$$
$$= CMY(0.5, 1, 0.5) = DarkMagenta$$

6. **Considering the results of Question 5. What can be concluded about linear interpolation in different color spaces?**

   From Question 5, the following colors were produced:

   (a) RGB(128,0,128), (b) HSV(120°,100%,100%), (c) CMY(0.5,1,0.5)

   However, when we convert these three colors to the same color space (e.g. RGB):

   (a) RGB(128,0,128), (b) RGB(0,255,0), (c) RGB(128,0,128)

   We see they are all not necessarily the same color. Hence, interpolation is dependent on the color space it is conducted in.

7. **Consider the following javascript snippet to setup a vertex buffer and the the pair of vertex/fragment shaders defined in GLSL:**

```
//  === VERTEX SHADER
attribute vec2 a_Position;
attribute vec3 a_Color;

varying vec3 v_Color;
main() {
    v_Color = a_Color;
    gl_Position = vec4(a_Position, 0.0, 1.0);
}

//  === FRAGMENT SHADER
varying vec3 v_Color;
main() {
    gl_FragColor = vec4(v_Color, 1.0);
}

// === VERTEX BUFFER SETUP IN JAVASCRIPT
gl.clearColor(0.0, 0.0, 0.0, 1.0);
gl.clear(gl.COLOR_BUFFER_BIT);
```

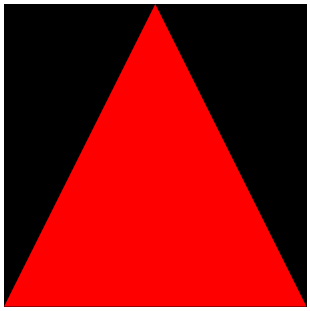
```
let vertices = new Float32Array([ ___ ]);

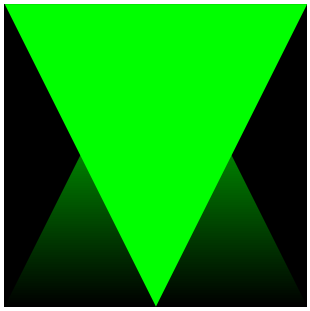
let a_Position = gl.getAttribLocation(gl.program, "a_Position");
gl.vertexAttribPointer(a_Position, ___, gl.FLOAT, false, ___, ___ );
gl.enableVertexAttribArray(a_Position);

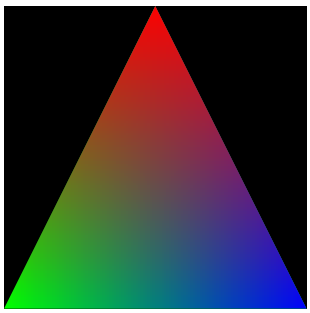
let a_Color = gl.getAttribLocation(gl.program, "a_Color");
gl.vertexAttribPointer(a_Color, ___, gl.FLOAT, false, ___, ___ );
gl.enableVertexAttribArray(a_Color);

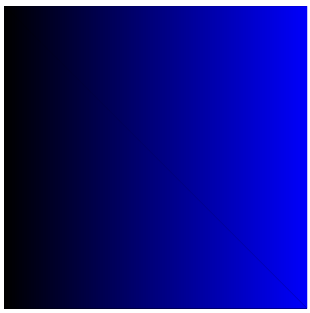
gl.bufferData(gl.ARRAY_BUFFER, vertices, gl.STATIC_DRAW);
gl.drawArrays(gl.TRIANGLES, 0, ___ );
```

Note there are blanks in this code highlighted in red. For each of the images below, fill the blanks of the code that would generate the output images:

(a) 

(c) 

(b) 

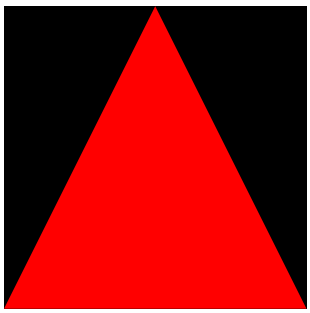
(d) 

We will define the Float32Array to have interleaved values for vertices with first position and then color for each vertex. That is it will contain [v1.X,v1.Y, v1.R,v1.G,v1.B, v2.X,v2.Y, v2.R,v2.G,v2.B,...].

vertexAttribPointer(attribName, *numberOfValues*, type, FALSE, *stride*, *offset*) sets up how to unpack this array into an attribute variable.

- *NumberOfValues* - We have 2 values for position and 3 values for color.
- *Stride* - is the distance from one vertex to the next. How far from v1.x to v2.x. FSIZE is the amount of memory a Float takes, and we have 5 elements, so we need to step FSIZE*5 between vertices.
- *Offset* - is the starting position for the first element in the array. Since position is first, it starts at 0, however color starts after we already had 2 position values, so color offset is 2*FSIZE. If Floats are 4 bytes, this says that color starts on byte 8.

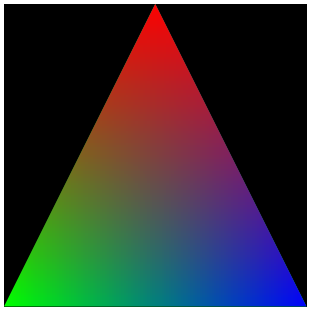
The last parameter of the drawArrays() function says how many vertices are in the buffer to draw.

1. 

```
let vertices = new Float32Array([
    0.0, 1.0, 1.0, 0.0, 0.0,
    -1.0, -1.0, 1.0, 0.0, 0.0,
    1.0, -1.0, 1.0, 0.0, 0.0]);

gl.vertexAttribPointer(a_Position, 2, gl.FLOAT, false, FSIZE*5, 0 );
gl.vertexAttribPointer(a_Color, 3, gl.FLOAT, false, FSIZE*5, FSIZE*2 );

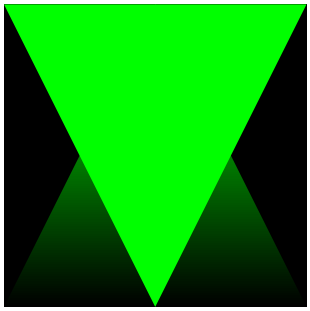
gl.drawArrays(gl.TRIANGLES, 0, 3 );
```

2. 

```
let vertices = new Float32Array([
    0.0, 1.0, 1.0, 0.0, 0.0,
    -1.0, -1.0, 0.0, 1.0, 0.0,
    1.0, -1.0, 0.0, 0.0, 1.0]);

gl.vertexAttribPointer(a_Position, 2, gl.FLOAT, false, FSIZE*5, 0 );
gl.vertexAttribPointer(a_Color, 3, gl.FLOAT, false, FSIZE*5, FSIZE*2 );

gl.drawArrays(gl.TRIANGLES, 0, 3 );
```

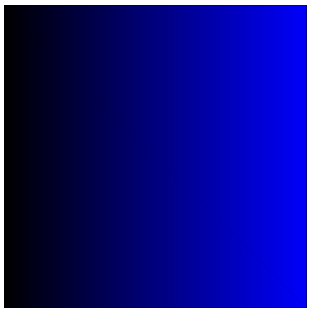


3.
```
let vertices = new Float32Array([
    0.0, 1.0, 0.0, 1.0, 0.0,
    -1.0, -1.0, 0.0, 0.0, 0.0,
    1.0, -1.0, 0.0, 0.0, 0.0,
    0.0, -1.0, 0.0, 1.0, 0.0,
```

```
    -1.0, 1.0, 0.0, 1.0, 0.0,
    1.0, 1.0, 0.0, 1.0, 0.0]);

gl.vertexAttribPointer(a_Position, 2, gl.FLOAT, false, FSIZE*5, 0 );
gl.vertexAttribPointer(a_Color, 3, gl.FLOAT, false, FSIZE*5, FSIZE*2 );

gl.drawArrays(gl.TRIANGLES, 0, 6 );
```



4. ```
   let vertices = new Float32Array([
       1.0, 1.0, 0.0, 0.0, 1.0,
       -1.0, -1.0, 0.0, 0.0, 0.0,
       1.0, -1.0, 0.0, 0.0, 1.0,
       -1.0, -1.0, 0.0, 0.0, 0.0,
       -1.0, 1.0, 0.0, 0.0, 0.0,
       1.0, 1.0, 0.0, 0.0, 1.0]);

   gl.vertexAttribPointer(a_Position, 2, gl.FLOAT, false, FSIZE*5, 0 );
   gl.vertexAttribPointer(a_Color, 3, gl.FLOAT, false, FSIZE*5, FSIZE*2 );

   gl.drawArrays(gl.TRIANGLES, 0, 6 );
   ```
   ```