Week 4: Colours and palettes

By now, I would assume that most of you are going to town with random colours. It’s fun, so if you haven’t, here’s one I prepared earlier **(!!epilepsy warning!!)**:

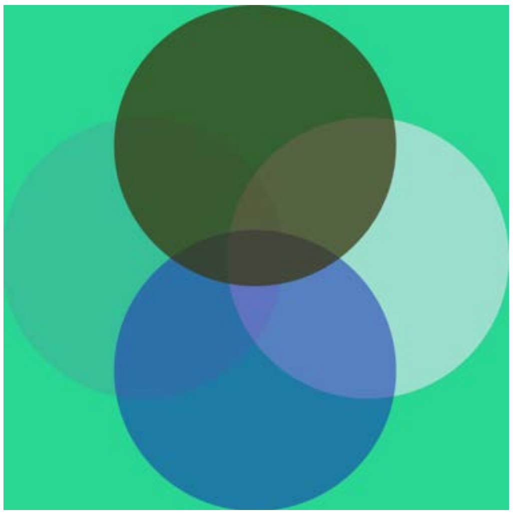
```
void setup(){
  size(900,900);
  background(0);
  noStroke();
}

void draw(){
  background(random(255), random(255), random(255));
  fill(random(255), random(255), random(255));
  rect(random(900), random(900),100,100);
  fill(random(255), random(255), random(255));
  ellipse(width/2, height/2,random(150,200),random(150,200));
}
```

Now, unless you’re intentionally after this eye-bleeding aesthetic, this can be a little overwhelming for most. Step two in our journey of creative coding is organising the chaos.

This week we’re going to focus on some colour techniques to better curate what the computer produces. Now, I’m not going to bore you with colour theory but rather, let’s talk about three ways to produce better colours.

Colour tip #1: Limit your values

By default, Processing accepts RGB values. These are represented by three numbers, each between 0-255 inclusive, telling the computer how much red, green and blue you want. These values are added together, to create an RGB colour. Therefore, if we're saying `fill(random(255), random(255), random(255), random(100));` we're pretty much telling the computer take any RGB value of any alpha value and print it to the screen. Let's generate a combo each time we press our mouse*.



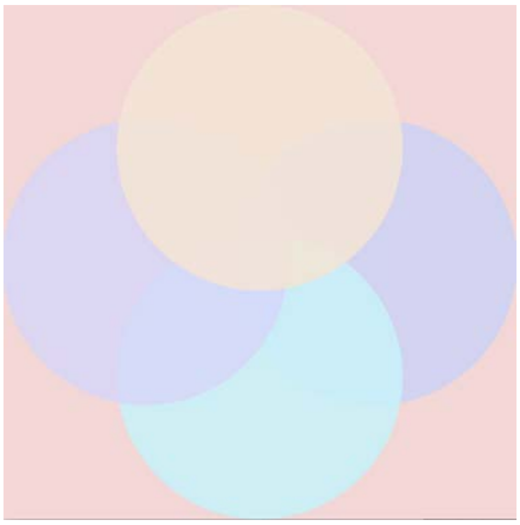
```
void setup(){
  size(900,900);
  background(255);
  noStroke();
}

void draw(){
}

void mousePressed(){
  background(random(255), random(255),
  random(255));
  fill(random(255), random(255),
  random(255),random(100));
  ellipse(width/2+200,height/2,500,500);
  fill(random(255), random(255),
  random(255),random(100));
  ellipse(width/2,height/2+200,500,500);
  fill(random(255), random(255),
  random(255),random(100));
  ellipse(width/2-200,height/2,500,500);
  fill(random(255), random(255),
  random(255),random(100));
  ellipse(width/2,height/2-200,500,500);
}
```

Now, this technique will eventually spit out something good - but it may take 1 mouse press or 100 mouse presses. We want to aim for something a little more reliable.

So by design, if we limit the pool of values that the computer can take from, we limit the amount of random freedom we're giving the computer to produce a colour. This is something human vision can do much easier than teaching a computer what is and isn't aesthetically pleasing.

*Note: **`mousePressed()`** still requires the **`draw()`** function to exist even if there's nothing there.

So let's try it again with limited values:



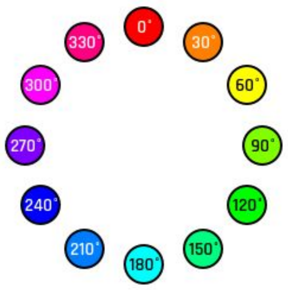
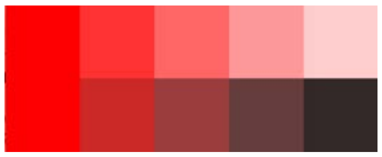
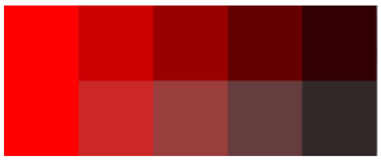
```
void setup(){
  size(900,900);
  background(255);
  noStroke();
}

void draw(){
}

void mousePressed(){
  background(random(220,250),
  random(210,250),random(200,250));

  fill(random(200,250), random(200,250),
  random(200,250), random(200,250));
  ellipse(width/2+200,height/2,500,500);

  fill(random(200,250), random(200,250),
  random(200,250), random(200,250));
  ellipse(width/2,height/2+200,500,500);

  fill(random(200,250), random(200,250),
  random(200,250), random(200,250));
  ellipse(width/2-200,height/2,500,500);

  fill(random(200,250), random(200,250),
  random(200,250), random(200,250));
  ellipse(width/2,height/2-200,500,500);
}
```

Now you will notice, that most of the generated colours aren’t so random, and therefore - feel a lot more human-selected than computer-selected. Not perfect, but better...

Colour tip #2: Change the colour mode

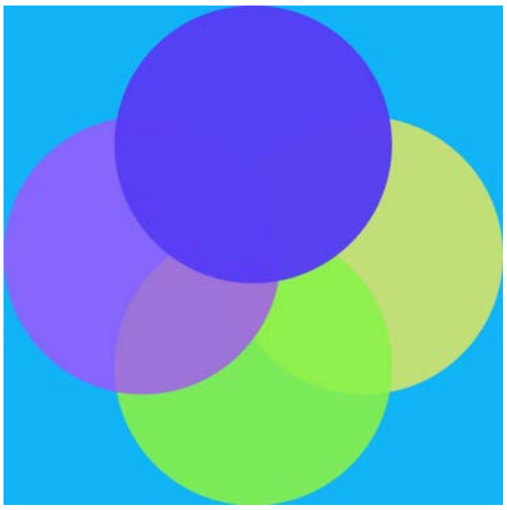
Even though we could technically look at every value within our limited RGB values to curate them - it would take a long time. Therefore, when it comes to manipulating colour, the HSB colour mode is more human friendly. HSB stands for Hue, Saturation and Brightness, which makes a lot more sense than trying to think about how much Red, Green and/or Blue is in a colour.

The standard HSB values are:

Hue: A value between 0-360° on the colour wheel.  0° 30° 60° 90° 120° 150° 180 210° 240° 270° 300° 330° Hint: hsb(0,0,100) = white and hsb(0,0,0) = black.	Saturation: A value between 0 and 100%. This represents the intensity of a colour from grayscale to full saturation.  Top row: 100% to 0% Saturation only Bottom row: 100% to 0% Saturation & 100% to 0% Brightness	Brightness: A value between 0 and 100%. This represents the amount of black and white we want in our colour.  Top row: 100% to 0% Brightness only Bottom row: 100% to 0% Saturation & 100% to 0% Brightness

It is worth noting here, that when you decrease both the saturation and the brightness of your colour, you tend to get a muddier colour. Sometimes that's a good thing but if you want richer colours, decrease the saturation levels to get a more muted colour and decrease the brightness to get a darker colour.

Let's just give it a go. Have a go playing around with the numbers.



```
void setup(){
  colorMode(HSB,360,100,100,100);
  size(900,900);
  background(random(180,220), random(80,100),
  random(95,100));
  noStroke();
}

void draw(){
}

void mousePressed(){
  background(random(180,220), random(80,100),
  random(95,100));
  fill(random(50,60), random(60,80),
  random(90,100), random(70,100));
  ellipse(width/2+200,height/2,500,500);
  fill(random(90,120), random(60,80),
  random(90,100), random(70,100));
  ellipse(width/2,height/2+200,500,500);
  fill(random(260,280), random(60,80),
  random(90,100), random(70,100));
  ellipse(width/2-200,height/2,500,500);
  fill(random(240,250), random(60,80),
  random(0,10), random(70,100));
  ellipse(width/2,height/2-200,500,500);
}
```

Now you should start catching yourself, changing the random values of a hue, saturation or brightness level. Do you feel more in control of your random colours now?

Let's try something. We want a colour palette of different shades and tints of pink and blue. How would you go about randomising these shades and tints?

Colour tip #3: Be intentional

These are oldies but goodies. Choose your colours beforehand - shock horror 😱 !
If you're like me and find colours terrifyingly hard, here are some tools that you may or may not have encountered to make our lives easier:

The super fast color schemes generator
https://coolors.co/

An AI colour generator
http://khroma.co/

Search by colour through other designers' work for inspiration
https://www.designspiration.net/
https://dribbble.com/colors/6c16c7

Select a colour. Now, make a palette of four other colours to complement your chosen colour. You'll use these colours in the next section.

Arrays and colour palettes

The next stage in our Creative Coding journey is being able to make a colour palette for our artwork that we can draw upon. The way that we are going to do this is to create an **`Array`** of colours. What is an **`Array`** you might ask? Well, I'm glad you asked...

An **`Array`** is yet another datatype which can store a list of "things". The "things" can be any datatype from an **`int`** to a **`float`** to a **`String`** and sometimes even MORE **`Arrays`**. In our case, we'll be using the datatype **`color`** to create a palette we can draw upon during our sketch.

To make an **`Array`** of colours, we need to define it at the top of our sketch including what datatype we will be using(**`color`**), a variable name(**`colorList`**) and a list of items.

```
color[] colorList = {item 0, item 1, item 2, item 3, item 4};
```

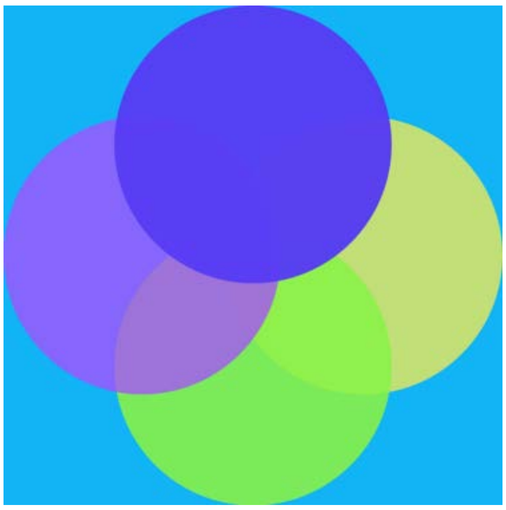
Now, we're going to replace each item with a different colour in the list. Use your colours from the above exercise and pay close attention to syntax here.

```
color[] colorList = {color(360,100,100), color(200,100,100)...};
```

Now we've made our colour palette, we want to grab the first "colour" in the array, also known as the 0th item. To do this, instead of explicitly stating a colour, we state:

```
colorList[0]
```

Let's see what this looks like.



```
color[] colorList = {color(320,100,100),
color(20,90,100), color(180,80,100),
color(220,80,100), color(280,80,100),
color(300,80,100)};

void setup(){
  colorMode(HSB,360,100,100);
  size(900,900);
  noStroke();
}

void draw(){
  background(colorList[0]);
  fill(colorList[1]);
  ellipse(width/2, height/2, 800,800);
}
```

How would you use the last `color` in the array as the `fill()` value?

technecolour

And so now, as a natural progression of course, we want to grab a random **color** in the palette every time we click our mouse. Try this baby:

```
color[] colorList = {color(320,100,100), color(20,90,100), color(180,80,100),
color(220,80,100), color(280,80,100)};

void setup(){
  colorMode(HSB,360,100,100);
  size(900,900);
  noStroke();
}

void draw(){
}

void mousePressed(){
  background(colorList[random(6)]);
  fill(colorList[random(6)]);
  ellipse(width/2, height/2, 800,800);
}
```

```
cannot convert from float to int
```

Oh no! Wondering what this is? This is because **random()** does not accept whole numbers(ie. **int**) - it only accepts **float** values. However, we have a problem - An **Array** only accept integers as it's positions. What to do?! Time for a little float conversion with the **int()** function.

```
color[] colorList = {color(320,100,100), color(20,90,100), color(180,80,100),
color(220,80,100), color(280,80,100)};

void setup(){
  colorMode(HSB,360,100,100);
  size(900,900);
  noStroke();
}

void draw(){
}

void mousePressed(){
  background(colorList[int(random(5))]);
  fill(colorList[int(random(5))]);
  ellipse(width/2, height/2, 800,800);
}
```

Remember how we spoke about the 0th, 1st, 2nd, 3rd, 4th index positions? You'll notice when we use **random()** above, we're saying **int(random(5))** but if you were to just call the last item it would be **colorList[4]**.

This is because random returns a value, in this case from 0 to 5, not including 5. Therefore, the possibilities are as follows: 0, 1, 2, 3 and 4. If we were to say **int(random(4))**, we would miss the possibility of the last colour option **color(280,80,100)**.

Save your images

The last step this week in curating our sketches, is to be able to save our canvas into a image when we're happy with it. You will use this to create your two static images for Task 1. Create a folder called "saved". To save an image to our folder only when we hit a key on our keyboard, let's try the following:

```
color[] colorList = {color(320,100,100), color(20,90,100), color(180,80,100),
color(220,80,100), color(280,80,100)};

void setup(){
  colorMode(HSB,360,100,100);
  size(900,900);
  noStroke();
}

void draw(){
}

void mousePressed(){
  background(colorList[int(random(5))]);
  fill(colorList[int(random(5))]);
  ellipse(width/2, height/2, 800,800);
}

void keyPressed(){
  save("saved/mySavedImage.png");
}
```

This will save a single image. If there's already an image called "mySavedImage.png" in the saved folder, it will replace the image. If we want to capture multiple frames at once, we can use the **`saveFrame()`** function instead. **Try saving frames instead.**

Week 4 Exercise

Use your sketch from week 2 to implement a colour palette. Code a sketch that produces a different set of colours for your images each time you press your mouse.

Keep your colour tips in mind if you want to add some random colours to your sketch.

Save your image when you're happy with it and upload it into your sketch. Comment the URL to the image at the top of your sketch.