P5 Cheatsheet - Week 01

Using Variables

To declare a variable in Javascript use either the let or const keyword. If the variable is required in both the setup() and draw() functions of P5 then declare outside as a global variable.

```
// Variables
let posX = 50;

// P5 Setup Function
function setup() {
    createCanvas(500, 500);
    background(255, 0, 0);
}

// P5 Draw Functions (Loops Continuously)
function draw() {
    background(255, 0, 0);
    rect(posX, 50, 200, 200);
    posX = posX + 1
}
```

Using Random() in Javascript

Javascript has it own random function [1] which can be used without P5. It will return a number (float) between 0 and 1. If you want a random number between 0 and 20 then multiply the random value by 20

```
// Will return a number between 0 and 20
let randomNum =  Math.random()*20;
```

if you require a number between 40 and 100 then do the following

```
// This will give you a random number between
// 0 and 60, we then add 40 to that range
// which will result in a number between
// 40 and 100

let randomNum= Math.random()*60 + 40;
```

The other option when using P5 is to use the P5 built in random function [2] which takes two parameters. It must be used either inside the setup() or draw() functions

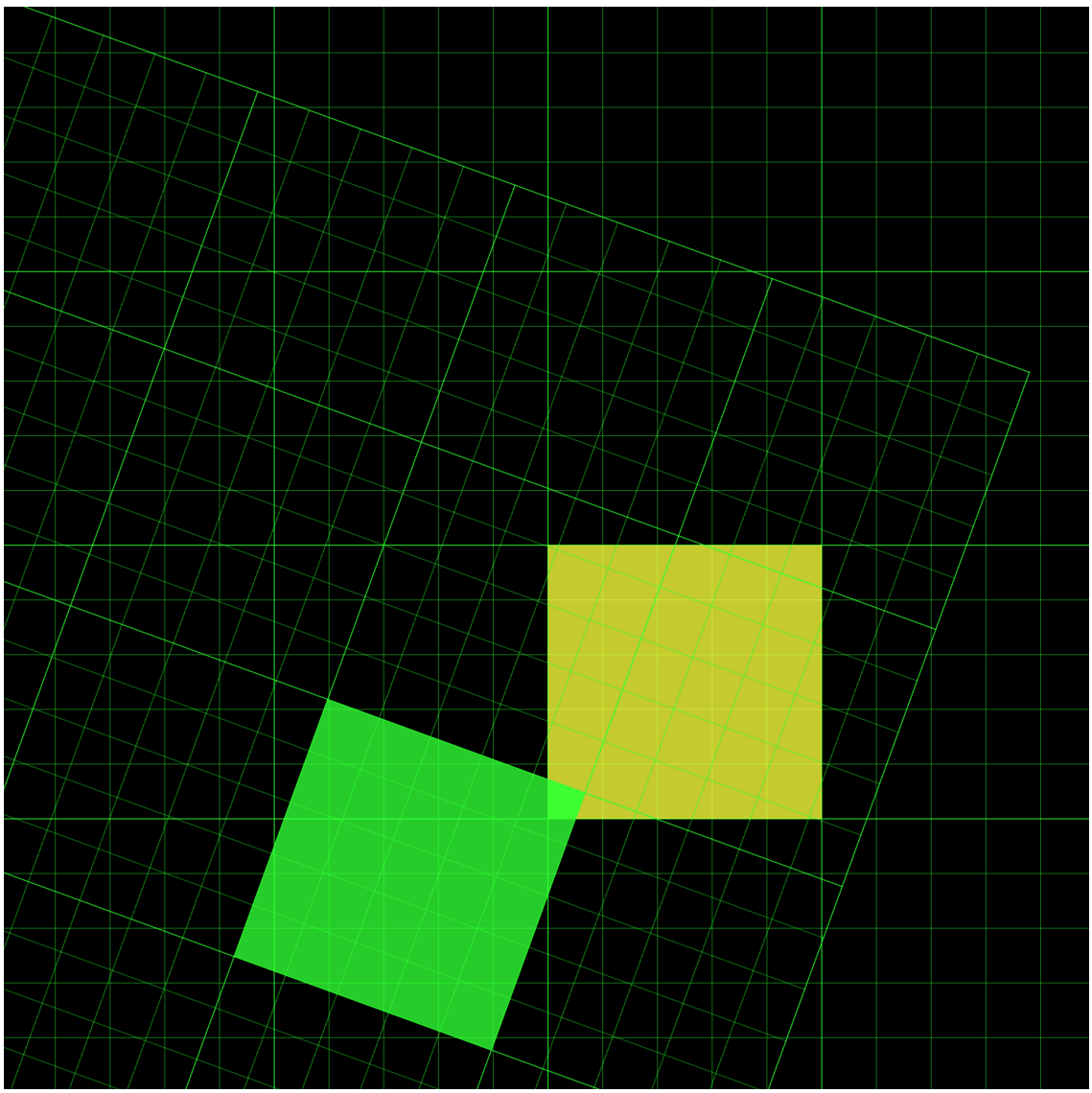
```
let randomNumber;

function setup(){
    randomNumber = random(40,100);
}
```

Using Translations in P5

By default the origin (0,0 position) is in the top left hand corner of the canvas. If you draw an object on the Canvas at location 100,100 the objects origin is still at the top left hand corner.

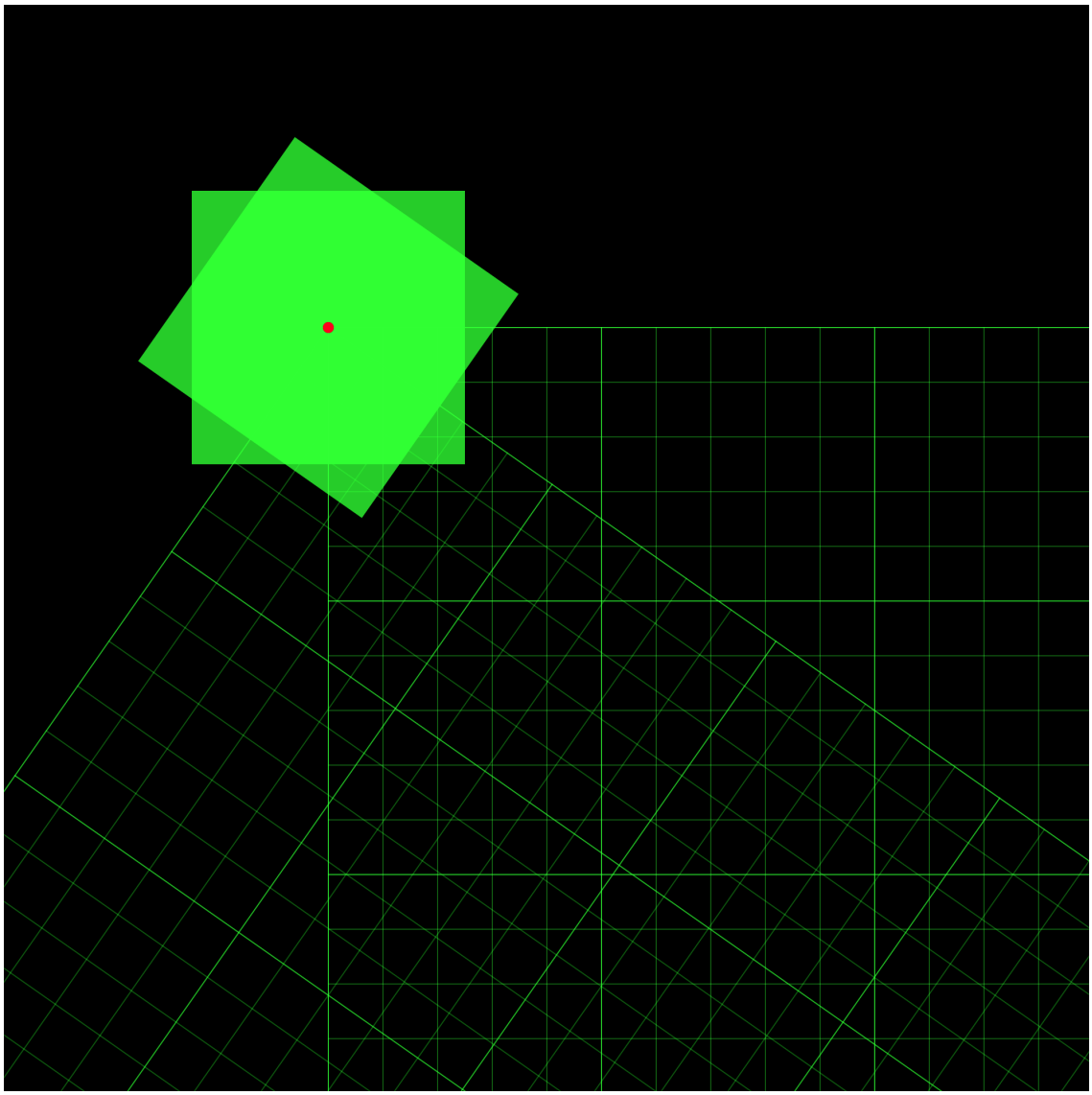
```
function draw(){
    rotate(15);
    rect(300,300,100,100);
}
```

This means that when you scale or rotate the object it will be around this point. This can be seen in the image below - the yellow square (if rotated) would rotate towards the Green square. THis is because its origin is in the top lef corner.



A better way of controlling where objects are to be rotated or scaled around is to draw the object correctly in relation to the origin point and then move the object to the desired location. In the example below we want the rectangle to rotate around its own center.

```
function draw(){
    rectMode(CENTER);
    translate(300,300);
    rotate(35);
    rect(0,0,100,100);
}
```



> Remember that when you are using translations it is important to wrap them in a push() [3] and pop() functions. This allows you to reset back to the original origin after each translation. The example below shows how this can be used.

```
function draw(){
    for(let i=0; i<10; i++){
        push()
        translate (i*50,i*50);
        rotate(30);
        rect(0,0,30,30);
        pop()
    }
}
```

AngleMode & RectMode

When you are using angles and creating shapes you may need to tell P5 how you want to do it. Angles by default are in radians and rectangles are drawn from the left corner with width and height. Ellipses are drawn by default from the center. AngleMode [4] controls the measurment used (degrees or Radians). RectMode [5] controls how rectangles are drawn. There is also ellipse mode [6]

```
function setup(){
    angleMode(DEGREES);
    rectMode(CENTER);
    rotate(45)
    rect(200,200,100,100)
}
```

Conditions

Conditions are used for checking a specific condition and executing code based on that condition being met. Think about as "if this is true or false then do something".After the IF statement we have soft brackets (parenthesis). Inside these we write the condition or number of conditions to be met. The && symbol means "and" and the || symbol means "or".

```
function draw(){
    let counter=0;
    // if counter greater than 10 and less than 20
    if (counter > 10 && counter < 20){
        console.log("Counter is between 10 and 20");
    }
    counter++;
}
```

> You have a choice to use if(condition){ doSomething } or if(condition){ doSomething } else { doSomethingElse }

```
function draw(){
    let counter=0;
    // if counter greater than 10 and less than 20
    if (counter > 10 && counter < 20){
        console.log("Counter is between 10 and 20");
    } else if (counter <10) {
        console.log("Counter is is less than 10");
    }
    counter++;
}
```

Conditions can also be written in ES6 using a shorthand method

```
function checkCounter(_counter){
    // if counter greater than 10 return true
    _counter > 10 ? true : false;
}
```

Modulus Operator

This is a really useful Mathematical operator. Operators like multiply, divide, addition and subtraction are really common. Modulus [7] is basically the same as remainder. In primary school we would say the follow:

```
12 / (divided) by 5 is 2 and remainder 2
15 / 4 is 3 remainder 3

16 % (modulus) 5 has a remainder of 1
17 % 5 returns 2
```

Modulus is really useful when you are counting groups or trying to work out if a number is odd or even. Even numbers always have a remainder of 0 when a modulus of 2 is used.

```
function checkEven(_num){
    if(_num % 2 == 0){
        return true
    } else {
        return false
    }
}

checkEven(2)  //should return true
checkEven(3)  //should return false
```

Drawing using loops

Sometimes in programming you will want to draw objects in a repeatable pattern - like in a grid, or in a circle, or in a diagonal. Loops, and understanding how to use them, can be really handy. A simple loop to draw a number of circles of width 20px in a line across the canvas would look like the following

```
function draw(){
    let canvasWidth =500
    let numOfBoxes=10;
    let boxSpacing = canvasWidth/numOfBoxes

    for (let x=0; x<numOfBoxes; x++){
        ellipse(x * boxSpacing, 100, 20,20)
    }
}
```

Now to develop that out and draw a grid of circles we would use a nested for loop, which is a loop inside a loop. The first loop is looping in the Y direction and the inner loop draws in the X direction. So that is how we draw a grid, line by line, left to right and down the page

```
function draw(){
    let canvasWidth =500
    let numOfBoxes=10;
    let boxSpacing = canvasWidth/numOfBoxes

    for (let y=0; y<numOfBoxes; y++){
        for (let x=0; x<numOfBoxes; x++){
        ellipse(x * boxSpacing, y * boxSpacing, 20,20)
        }
    }
}
```

If we go back to something we mentioned earlier, it is better to translate the objects so that we can draw it at 0,0. THis is useful if we need to rotate them later. Also, note that we can modulus to draw every second object.

```
function draw(){
    let canvasWidth =500
    let numOfBoxes=11;
    let boxSpacing = canvasWidth/numOfBoxes

    for (let y=0; y<numOfBoxes; y++){
        for (let x=0; x<numOfBoxes; x++){
            let gridPos = (y*numOfBoxes + x);
            if(gridPos % 2 == 0){
                translate(x * boxSpacing, y * boxSpacing)
                ellipse(0,0, 20,20)
            }
        }
    }
}
```

Activities

1. Write some code that will increment a value after each draw() cycle. Perhaps consider using a rectangle that has its x position changing. Give the rectangle a stroke and a different fill colour
2. Experiment with both types of random functions, the built in P5 random function and the Javascript Math.Random. See wheter you can console.log a number within a particul range like between 5 and 12. Can you convert this number to an integer or generate a random number of rectangles on the screen.
3. Try and draw a rectanlge and rotate it around its centre. You may need to use translate and rectMode
4. Using a for loop try and draw a grid with a number of horiszontal and vertical lines. Make every 5 lines a different colour of heavier stroke
5. Now combining all of your knowledge draw a grid of shapes that are rotated or rotating. Consider using modulus to create or generate patterns

---

1. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math/random ↩
2. https://p5js.org/reference/#/p5/random ↩
3. https://p5js.org/reference/#/p5/push ↩
4. https://p5js.org/reference/#/p5/angleMode ↩
5. https://p5js.org/reference/#/p5/rectMode ↩
6. https://p5js.org/reference/#/p5/ellipseMode ↩
7. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Remainder ↩