Processing Cheat Sheet – Loops, Modulus

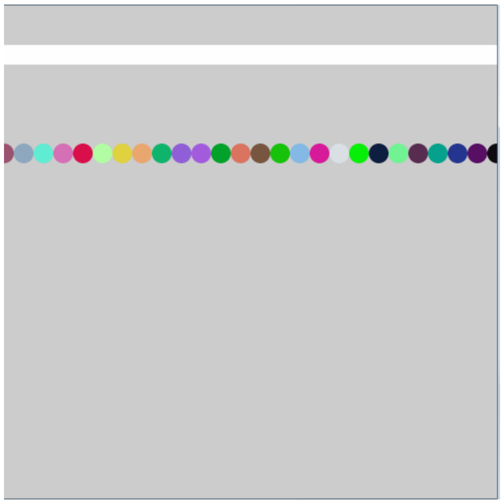
1 – for Loops and Modulus

Loops let you repeat a certain piece of code, to save you time and energy, and accomplish complex things that would otherwise be impossible.

The modulus (%) operator gives the remainder when dividing two numbers. This means we can look for a modulus (%) of zero to check if a number is divisible by another.

The loop below repeats ellipses across the screen, drawing one row without any spacing, and another row with an ellipse every 20 pixels:

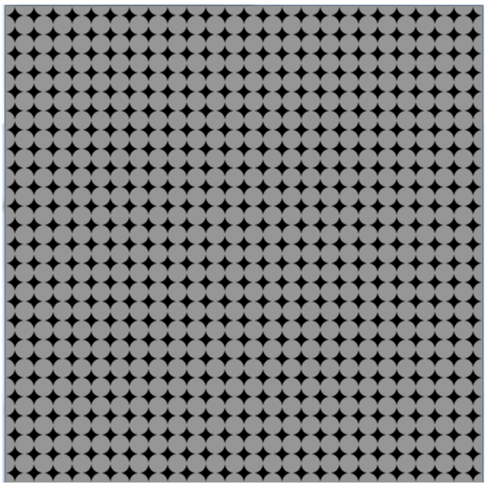
```
size (500, 500);
int counter;
//draws ellipses across screen
for (counter=0; counter <=width; counter++){
   fill(255, 100);
   noStroke();
    ellipse(counter, 50, 20, 20);
}
//spaced out ellipses
for (counter=0; counter <=width; counter++){
   fill(random(255), random(255), random(255));
   noStroke();
    if(counter%20 == 0) ellipse(counter, 150, 20, 20);
}
```



2 – Nested for Loops

By putting a loop inside a loop, we can traverse the screen both horizontally and vertically.

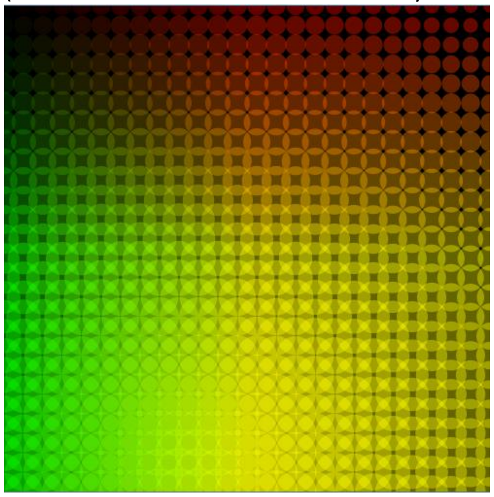
```
//draws ellipses across AND DOWN screen
size (500, 500);
background(0);
int counterx;
int countery;
for (countery=0; countery<=height; countery++){
      for (counterx=0; counterx<=width; counterx++){
             fill(255, 150);
             noStroke();
             if (counterx%20 == 0 && countery%20==0){
                  ellipse(counterx, countery, 20, 20);
              }
    }
}
```



3 – Nested for Loops + Mouse Interaction

We can use mouseX and mouseY here, like we would any other code, to affect things like shape size, position, and color. Here, we'll use a function called **dist()** to measure the distance between the mouse and each ellipse, and change the ellipse size based on mouse closeness.

(We also added counterx and countery to the color values to make things more visually interesting!)



```
void draw(){
      background(0);
    int counterx;
    int countery;
    for (countery=0; countery<=height; countery++){
        for (counterx=0; counterx<=width; counterx++){
          fill(counterx, countery, 0, 100);
           noStroke();
           if (counterx%20 == 0 && countery%20==0){
                 float distance = dist (counterx, countery, mouseX, mouseY);
                 float size = map(distance, 0, 700, 50, 5);
              ellipse(counterx, countery, size, size);
               }
        }
    }
}
```

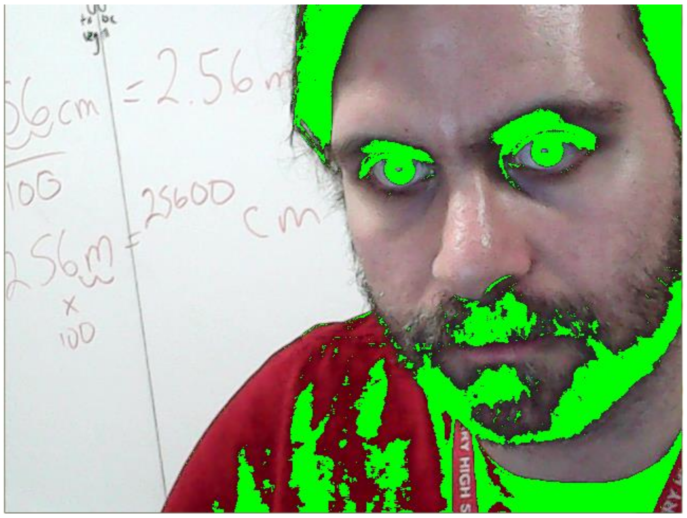
Processing Cheat Sheet –Video Input

1 – Basic Video Input

This code just create a capture feed from the camera, then captures frames from the camera, and draws them to the screen. Then the **filter()** function is used to add a posterize effect.

```
import processing.video.*;
Capture video;
void setup(){
     size(640, 480);
     video = new Capture(this, width, height);
      video.start(); // start the camera
      video.read();
}
void draw(){
    if (video.available()){
        video.read(); // read a new frame from
camera
        image(video, 0, 0); //draw frame to screen
        filter(POSTERIZE,3); // add this effect later for fun
    }
}
```

2 – Extracting and Altering Pixel Colors

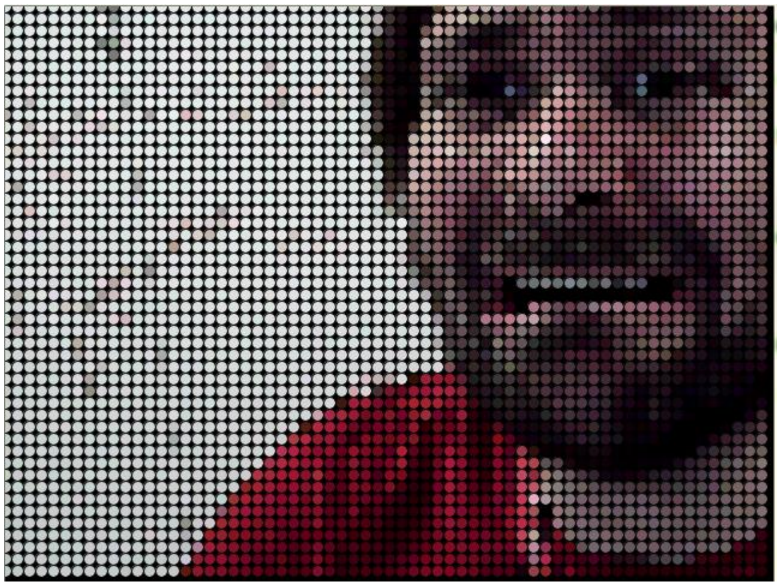


```
import processing.video.*;
Capture video;

void setup(){
     size(640, 480);
     video = new Capture(this, width, height);
      video.start();
      video.read();
}

void draw(){
    if (video.available()){
          video.read(); // read a new frame from camera
          video.loadPixels(); //extracts the pixel colors into an array called pixels[]
          int counter;
          //this for loop traverses all the pixels in the image
          for (counter = 0; counter < video.width*video.height; counter++){
          float r = red(video.pixels[counter]); // variables to store this image's color
          float g = green(video.pixels[counter]);
          float b = blue(video.pixels[counter]);
          if (r+g+b < 100){ // if pixel color is very dark...
             video.pixels[counter] = color(0,255,0); //replace pixel color with green
          }
          }
          image(video, 0, 0); //draw altered frame to screen
     }
}
```

3 – Extracting Pixel Colors and Using them in Shapes
This example extracts the color values of the pixels, like above, but uses them to color shapes, which are drawn using a loop and modulus, like our earlier loop projects.



```
import processing.video.*;
Capture video;

void setup(){
    size(640, 480);
    video = new Capture(this, width,
height);
    video.start();
    video.read();
}

void draw(){
    if (video.available()){
      background(0);
        video.read(); // read a new frame from camera
        video.loadPixels();//extracts the pixel colors into an array called pixels[]
        int counter;
        for (counter = 0; counter <width*height; counter +=10){
            color pixelcolor = video.pixels[counter]; // store the color from the pixel
            fill(pixelcolor); // use that color to draw next shape
            int column = int(counter % width); // calculate what ‘column’ we’re on
            int row = int(counter / width); // calculate what ‘row’ we’re on
            if (row%10==0) ellipse(column, row, 10, 10); // draw the ellipse
        }
    }
}
```