https://docs.google.com/presentation/d/1yqMKVF2FICF13TFxLWNIhh579gE8U0FsF3xM_sJJU_A/edit?usp=sharing

## 1: Introducción y Formas Básicas

### 1.1 Dibujando Formas Básicas (45 minutos)

- **Contenido:**
  - Funciones básicas: `size()`, `background()`, `ellipse()`, `rect()`, `line()`.
  - Sistema de coordenadas.
- **Actividad:**
  - Crear un lienzo de 800x600 píxeles.
  - Dibujar un círculo, un rectángulo y una línea en diferentes posiciones.

**Código de ejemplo:**

```
void setup() {
  size(800, 600);
  background(255);
}

void draw() {
  ellipse(400, 300, 150, 150);  // Círculo
  rect(200, 150, 100, 200);  // Rectángulo
  line(0, 0, 800, 600);          // Línea diagonal
}
```

---

## 2: Uso del Color y Formas Avanzadas

### 2.1 Uso del Color (30 minutos)

- **Contenido:**
  - Funciones `fill()`, `stroke()`, `noFill()`, `noStroke()`.
  - Valores RGB y transparencia.
- **Actividad:**

- Dibujar formas usando diferentes colores y niveles de transparencia.

**Código de ejemplo:**

```
void setup() {
  size(800, 600);
  background(255);
}

void draw() {
  fill(255, 0, 0);         // Rojo
  ellipse(200, 200, 150, 150);

  fill(0, 255, 0, 150);  // Verde con transparencia
  rect(300, 300, 150, 150);

  noFill();
  stroke(0, 0, 255);              // Solo contorno azul
  line(0, 0, 800, 600);
}
```

```
void setup() {
  size(800, 600);
  background(255);
}

void draw() {
  fill(255, 0, 0);         // Rojo
  ellipse(267, 200, 118, 150);

  fill(0, 255, 0, 150);   // Verde con transparencia
  rect(300, 300, 150, 150);

  noFill();
  stroke(0, 0, 255);   // Solo contorno azul
  line(0, 0, 800, 600);
}
```

```
void keyPressed() {
  if (key == 's' || key == 'S') {
        saveFrame("screenshot-####.jpg");
  }
}
```

---

**2.2 Formas Avanzadas (30 minutos)**

- **Contenido:**
  - Funciones `beginShape()`, `vertex()`, `endShape()`.
  - Creación de polígonos y formas complejas.
- **Actividad:**
  - Dibujar un triángulo y un polígono complejo.

**Código de ejemplo:**

```
void setup() {
  size(800, 600);
  background(255);
}

void draw() {
  beginShape();
  vertex(100, 100);
  vertex(150, 200);
  vertex(50, 200);
  endShape(CLOSE);

  beginShape();
  vertex(300, 300);
  vertex(350, 350);
  vertex(300, 400);
  vertex(250, 350);
  endShape(CLOSE);
}
```

**ROSTRO**

```
void setup() {
  size(400, 400);
  background(255);

  // Cabeza
  fill(255, 220, 185);
  ellipse(width/2, height/2, 200, 250);

  // Ojos
  fill(255);
  ellipse(width/2 - 50, height/2 - 50, 50, 30);
  ellipse(width/2 + 50, height/2 - 50, 50, 30);

  // Pupilas
  fill(0);
  ellipse(width/2 - 50, height/2 - 50, 20, 20);
  ellipse(width/2 + 50, height/2 - 50, 20, 20);

  // Nariz
  fill(255, 200, 170);
  triangle(width/2, height/2 - 20, width/2 - 20, height/2 + 20, width/2 + 20, height/2 + 20);

  // Boca
  fill(255, 0, 0);
  arc(width/2, height/2 + 60, 80, 40, 0, PI);
}
```

---

### 3: Interacción y Animación Básica

#### 3.1 Interacción con el Ratón y Teclado (30 minutos)

- **Contenido:**
  - Funciones `mousePressed()`, `keyPressed()`, `mouseX`, `mouseY`.

    - Uso de variables para rastrear el estado.
- **Actividad:**
    - Dibujar una forma que siga el ratón.
    - Cambiar el color de fondo al presionar una tecla.

**Código de ejemplo:**

```
void setup() {
  size(800, 600);
}

void draw() {
  background(255);
  ellipse(mouseX, mouseY, 50, 50);  // Círculo sigue al ratón
}

void keyPressed() {
  if (key == 'r') {
        background(255, 0, 0);  // Fondo rojo
  } else if (key == 'g') {
        background(0, 255, 0);  // Fondo verde
  } else if (key == 'b') {
        background(0, 0, 255);  // Fondo azul
  }
}
```

### 3.2 Animación Básica (30 minutos)

- **Contenido:**
  - Uso de `frameRate()`.
  - Movimiento de objetos.
- **Actividad:**
  - Crear una animación simple donde una forma se mueve automáticamente por la pantalla.

**Código de ejemplo:**

```
float x = 0;
float y = 300;
float speed = 2;

void setup() {
  size(800, 600);
  frameRate(60);
}

void draw() {
  background(255);
  ellipse(x, y, 50, 50);  // Círculo que se mueve
  x += speed;

  if (x > width) {
        x = 0;  // Reiniciar posición
  }
}
```

**semáforo**

```
int estado = 0; // 0 = rojo, 1 = amarillo, 2 = verde

void setup() {
  size(200, 600);
}

void draw() {
  background(255);
  dibujarSemaforo();
}

void mousePressed() {
  estado = (estado + 1) % 3; // Cambiar al siguiente estado
}

void dibujarSemaforo() {
  // Dibujar contorno del semáforo
  fill(50);
  rect(50, 50, 100, 300);

  // Dibujar luces
  if (estado == 0) {
        fill(255, 0, 0); // Rojo
  } else {
        fill(150, 0, 0);
  }
  ellipse(100, 100, 80, 80);

  if (estado == 1) {
        fill(255, 255, 0); // Amarillo
  } else {
        fill(150, 150, 0);
  }
  ellipse(100, 200, 80, 80);
```

```
  if (estado == 2) {
        fill(0, 255, 0); // Verde
  } else {
        fill(0, 150, 0);
  }
  ellipse(100, 300, 80, 80);
}
```

---

**ojos que guiñan**

```
int eyeState = 0; // 0 = normal, 1 = wink right, 2 = wink left
boolean mouthOpen = false;

void setup() {
  size(400, 400);
}

void draw() {
  background(255);
  drawFace();
  if (eyeState == 1) {
        winkRightEye();
  } else if (eyeState == 2) {
        winkLeftEye();
  }
  if (mouthOpen) {
        openMouth();
  }
}

void drawFace() {
  // Head
  fill(255, 224, 189);
  ellipse(width/2, height/2, 200, 250);

  // Eyes
```

```
  fill(255);
  ellipse(width/2 - 50, height/2 - 50, 40, 20); // Left eye
  ellipse(width/2 + 50, height/2 - 50, 40, 20); // Right eye

  fill(0);
  ellipse(width/2 - 50, height/2 - 50, 10, 10); // Left pupil
  ellipse(width/2 + 50, height/2 - 50, 10, 10); // Right pupil

  // Nose
  fill(255, 224, 189);
  triangle(width/2, height/2 - 30, width/2 - 10, height/2 + 20, width/2 + 10, height/2 + 20);

  // Mouth
  fill(255, 0, 0);
  rect(width/2 - 30, height/2 + 60, 60, 20);
}

void winkRightEye() {
  fill(255, 224, 189);
  rect(width/2 + 30, height/2 - 60, 40, 20); // Cover right eye

  fill(0);
  line(width/2 + 30, height/2 - 50, width/2 + 70, height/2 - 50); // Right eye wink line
}

void winkLeftEye() {
  fill(255, 224, 189);
  rect(width/2 - 70, height/2 - 60, 40, 20); // Cover left eye

  fill(0);
  line(width/2 - 70, height/2 - 50, width/2 - 30, height/2 - 50); // Left eye wink line
}

void openMouth() {
  fill(255, 0, 0);
  rect(width/2 - 30, height/2 + 60, 60, 40); // Open mouth
}
```

```
void keyPressed() {
  if (key == 'D' || key == 'd') {
        eyeState = 1; // Wink right eye
  } else if (key == 'I' || key == 'i') {
        eyeState = 2; // Wink left eye
  } else if (key == 'B' || key == 'b') {
        mouthOpen = true; // Open mouth
  }
}

void keyReleased() {
  if (key == 'D' || key == 'd' || key == 'I' || key == 'i') {
        eyeState = 0; // Reset eyes
  } else if (key == 'B' || key == 'b') {
        mouthOpen = false; // Close mouth
  }
}
```

---

**gato (sin orejas)**

```
int eyeStateRight = 0;  // 0 = open, 1 = closed
int eyeStateLeft = 0;   // 0 = open, 1 = closed
boolean tongueOut = false;

void setup() {
  size(800, 600);
}

void draw() {
  background(255);
  drawCat();
}

void drawCat() {
  // Draw face
```

```
  fill(200, 150, 100);
  ellipse(width / 2, height / 2, 300, 300);

  // Draw eyes
  drawEye(width / 2 - 70, height / 2 - 40, eyeStateLeft);
  drawEye(width / 2 + 70, height / 2 - 40, eyeStateRight);

  // Draw nose
  fill(150, 75, 0);
  triangle(width / 2 - 20, height / 2, width / 2 + 20, height / 2, width / 2, height / 2 + 30);

  // Draw mouth
  noFill();
  stroke(150, 75, 0);
  strokeWeight(4);
  arc(width / 2, height / 2 + 50, 80, 40, 0, PI);

  // Draw tongue if out
  if (tongueOut) {
        fill(255, 0, 0);
        ellipse(width / 2, height / 2 + 89, 21, 39);
  }
}

void drawEye(float x, float y, int state) {
  if (state == 0) {
        fill(255);
        ellipse(x, y, 50, 30);
        fill(0);
        ellipse(x, y, 20, 20);
  } else {
        fill(255);
        arc(x, y, 50, 30, 0, PI);
  }
}

void keyPressed() {
```

```
  if (key == 'D' || key == 'd') {
        eyeStateRight = 1;
  } else if (key == 'I' || key == 'i') {
        eyeStateLeft = 1;
  } else if (key == 'L' || key == 'l') {
        tongueOut = true;
  }
}

void keyReleased() {
  if (key == 'D' || key == 'd') {
        eyeStateRight = 0;
  } else if (key == 'I' || key == 'i') {
        eyeStateLeft = 0;
  } else if (key == 'L' || key == 'l') {
        tongueOut = false;
  }
}
```

---

**gato con orejas**

```
int eyeStateRight = 0;  // 0 = open, 1 = closed
int eyeStateLeft = 0;   // 0 = open, 1 = closed
boolean tongueOut = false;

void setup() {
  size(800, 600);
}

void draw() {
  background(255);
  drawCat();
}

void drawCat() {
  // Draw ears
```

```
  fill(200, 150, 100);
   triangle(width / 2 - 150, height / 2 - 100, width / 2 - 70, height / 2 - 200, width / 2 - 50, height / 2 - 100);
   triangle(width / 2 + 150, height / 2 - 100, width / 2 + 70, height / 2 - 200, width / 2 + 50, height / 2 - 100);

  // Draw face
  fill(200, 150, 100);
  ellipse(width / 2, height / 2, 300, 300);

  // Draw eyes
  drawEye(width / 2 - 70, height / 2 - 40, eyeStateLeft);
  drawEye(width / 2 + 70, height / 2 - 40, eyeStateRight);

  // Draw nose
  fill(150, 75, 0);
  triangle(width / 2 - 20, height / 2, width / 2 + 20, height / 2, width / 2, height / 2 + 30);

  // Draw mouth
  noFill();
  stroke(150, 75, 0);
  strokeWeight(4);
  arc(width / 2, height / 2 + 50, 80, 40, 0, PI);

  // Draw tongue if out
  if (tongueOut) {
        fill(255, 0, 0);
        ellipse(width / 2, height / 2 + 89, 21, 39);
  }
}

void drawEye(float x, float y, int state) {
  if (state == 0) {
        fill(255);
        ellipse(x, y, 50, 30);
        fill(0);
        ellipse(x, y, 20, 20);
```

```
  } else {
        fill(255);
        arc(x, y, 50, 30, 0, PI);
  }
}

void keyPressed() {
  if (key == 'D' || key == 'd') {
        eyeStateRight = 1;
  } else if (key == 'I' || key == 'i') {
        eyeStateLeft = 1;
  } else if (key == 'L' || key == 'l') {
        tongueOut = true;
  }
}

void keyReleased() {
  if (key == 'D' || key == 'd') {
        eyeStateRight = 0;
  } else if (key == 'I' || key == 'i') {
        eyeStateLeft = 0;
  } else if (key == 'L' || key == 'l') {
        tongueOut = false;
  }
}
```

---

**tictactoe**

```
int[][] board = new int[3][3]; // 0 = empty, 1 = X, 2 = O
int currentPlayer = 1; // 1 = X, 2 = O
boolean gameEnded = false;

void setup() {
  size(300, 300);
  for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
```

```
        board[i][j] = 0;
        }
  }
}

void draw() {
  background(255);
  drawBoard();
  checkWin();
}

void drawBoard() {
  strokeWeight(4);
  line(100, 0, 100, 300);
  line(200, 0, 200, 300);
  line(0, 100, 300, 100);
  line(0, 200, 300, 200);

  for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
        if (board[i][j] == 1) {
        drawX(i, j);
        } else if (board[i][j] == 2) {
        drawO(i, j);
        }
        }
  }
}

void drawX(int i, int j) {
  line(100 * i + 20, 100 * j + 20, 100 * i + 80, 100 * j + 80);
  line(100 * i + 80, 100 * j + 20, 100 * i + 20, 100 * j + 80);
}

void drawO(int i, int j) {
  ellipse(100 * i + 50, 100 * j + 50, 60, 60);
}
```

```
void mousePressed() {
  if (!gameEnded) {
        int i = mouseX / 100;
        int j = mouseY / 100;
        if (board[i][j] == 0) {
        board[i][j] = currentPlayer;
        currentPlayer = 3 - currentPlayer; // switch player
        }
  }
}

void keyPressed() {
  if (key == 'X' || key == 'x') {
        currentPlayer = 1;
  } else if (key == 'O' || key == 'o') {
        currentPlayer = 2;
  }
}

void checkWin() {
  for (int i = 0; i < 3; i++) {
        if (board[i][0] != 0 && board[i][0] == board[i][1] && board[i][1] == board[i][2]) {
        gameEnded = true;
        displayWinner(board[i][0]);
        }
        if (board[0][i] != 0 && board[0][i] == board[1][i] && board[1][i] == board[2][i]) {
        gameEnded = true;
        displayWinner(board[0][i]);
        }
  }
  if (board[0][0] != 0 && board[0][0] == board[1][1] && board[1][1] == board[2][2]) {
        gameEnded = true;
        displayWinner(board[0][0]);
  }
  if (board[2][0] != 0 && board[2][0] == board[1][1] && board[1][1] == board[0][2]) {
        gameEnded = true;
```

```
        displayWinner(board[2][0]);
  }
}

void displayWinner(int winner) {
  fill(243,252,7);
  textSize(32);
  if (winner == 1) {
        text("X Wins!", 90, 150);
  } else if (winner == 2) {
        text("O Wins!", 90, 150);
  }
}
```

---

***dibujar_con_mouse***

```
void setup() {
  size(800, 600); // Tamaño de la ventana
  background(255); // Fondo blanco
}

void draw() {
  // No hacemos nada en draw porque el dibujo lo manejamos en mouseDragged
}

void mouseDragged() {
  stroke(0); // Color del trazo (negro)
  strokeWeight(2); // Grosor del trazo
   line(pmouseX, pmouseY, mouseX, mouseY); // Dibuja una línea desde la posición anterior del mouse a la posición actual
}
```

---

***letras_con_teclado***

```
char currentKey = ' '; // Variable para almacenar la tecla presionada
```

```
void setup() {
  size(500, 200); // Tamaño de la ventana
  background(0); // Fondo negro
  fill(255); // Color del texto (blanco)
  textSize(48); // Tamaño del texto
  textAlign(CENTER, CENTER); // Alineación del texto
}

void draw() {
  background(0); // Fondo negro (se refresca cada frame)
  text(currentKey, width / 2, height / 2); // Mostrar la tecla presionada en el centro
}

void keyPressed() {
  currentKey = key; // Actualizar la tecla presionada
}
```

---

**esfera que cambia de color al chocar con las paredes**

```
float x, y; // Posición de la esfera
float xSpeed, ySpeed; // Velocidad de la esfera
color ballColor; // Color de la esfera

void setup() {
  size(800, 600); // Tamaño de la ventana
  x = width / 2; // Posición inicial en el centro
  y = height / 2;
  xSpeed = 10; // Velocidad inicial
  ySpeed = 8;
  ballColor = color(255); // Color inicial (blanco)
}

void draw() {
  background(0); // Fondo negro

  fill(ballColor); // Relleno de la esfera con el color actual
```

```
  noStroke();
  ellipse(x, y, 50, 50); // Dibujar la esfera

  x += xSpeed; // Actualizar posición en x
  y += ySpeed; // Actualizar posición en y

  // Verificar colisiones con los bordes
  if (x <= 25 || x >= width - 25) {
        xSpeed *= -1; // Invertir la velocidad en x
        changeColor(); // Cambiar color
  }
  if (y <= 25 || y >= height - 25) {
        ySpeed *= -1; // Invertir la velocidad en y
        changeColor(); // Cambiar color
  }
}

// Función para cambiar el color a un nuevo color RGB aleatorio
void changeColor() {
  ballColor = color(random(255), random(255), random(255));
}
```

---

```
int x, y;
int diameter = 50;
float speedY = 0;
float gravity = 0.5;
boolean isFalling = true;

void setup() {
  size(800, 600);
  x = width / 2;
  y = diameter / 2;
  noStroke();
}

void draw() {
```

```
  background(255);
  fill(255, 255, 0);
  ellipse(x, y, diameter, diameter);

  if (isFalling) {
        speedY += gravity;
        y += speedY;
        if (y > height - diameter / 2) {
        y = height - diameter / 2;
        speedY *= -0.8;
        if (abs(speedY) < 1) {
        speedY = 0;
        isFalling = false;
        }
        }
  }
}

void mousePressed() {
  y = diameter / 2;
  speedY = 0;
  isFalling = true;
}
```

---

**esferas de color sobre fondo negro**

```
void setup() {
  size(800, 800);
  noStroke();
  background(0);
}

void draw() {
  fill(0, 12);
  rect(0, 0, width, height);
```

```
  float r = random(180, 255);
  float g = random(100, 125);
  float b = random(100, 155);

  fill(r, g, b, 150);
  float d = random(200, 350);
  ellipse(random(width), random(height), d, d);
}
```

---

**elipse dibuja aleatoriamente en pantalla**

```
float speed = 30.0;
int diameter = 80;
float x;
float y;

void setup() {
  size(800, 800);
  background(60);
  x = width / 2;
  y = height / 2;
  randomSeed(30);
}

void draw() {
  x += random(-speed, speed);
  y += random(-speed, speed);
  x = constrain(x, diameter, width - diameter);
  y = constrain(y, diameter, height - diameter);
  ellipse(x, y, diameter, diameter);
}
```

---

**mandala**

```
void setup(){
```

```
 size(1000, 1000);
 background(0);
 noStroke();
 translate(width/2, height/2);

 int pointCount = 6;
 int steps = 60;
 float outerRadius = width * 0.5;
 float innerRadiusFactor = .5;
 float innerRadius = outerRadius * innerRadiusFactor;
 float outerRadiusRatio = outerRadius / steps;
 float innerRadiusRatio = innerRadius / steps;
 float shadeRatio = 255 / steps;
 float rotationRatio = 45.0 / steps;

 for(int i = 0; i < steps; i++){
        stroke(255 - shadeRatio * i, 100);
        colorMode(HSB);
        fill(shadeRatio * i, 100, 100);
        pushMatrix();
        rotate(rotationRatio * i + PI / 90);
        star(pointCount, outerRadius - outerRadiusRatio * i, innerRadius -
innerRadiusRatio * i);
        popMatrix();
 }
}

void star(int numPoints, float outerRadius, float innerRadius) {
 float angle = TWO_PI / numPoints;
 float halfAngle = angle / 2.0;
 beginShape();
 for (float a = 0; a < TWO_PI; a += angle) {
        float sx = cos(a) * outerRadius;
        float sy = sin(a) * outerRadius;
        vertex(sx, sy);
        sx = cos(a + halfAngle) * innerRadius;
        sy = sin(a + halfAngle) * innerRadius;
```

```
        vertex(sx, sy);
  }
  endShape(CLOSE);
}
```

---

**mandala interactivo**

```
void setup(){
  size(1000, 1000);
  background(0);
  noStroke();
}

void draw(){
  background(0);
  translate(width/2, height/2);

  int pointCount = 6;
   int steps = int(map(mouseX, 0, width, 10, 100)); // Número de pasos depende de la posición del ratón
  float outerRadius = width * 0.5;
  float innerRadiusFactor = .5;
  float innerRadius = outerRadius * innerRadiusFactor;
  float outerRadiusRatio = outerRadius / steps;
  float innerRadiusRatio = innerRadius / steps;
  float shadeRatio = 255 / steps;
  float rotationRatio = 45.0 / steps;

  for(int i = 0; i < steps; i++){
        stroke(255 - shadeRatio * i, 100);
        colorMode(HSB);
        fill(shadeRatio * i, 100, 100);
        pushMatrix();
        rotate(rotationRatio * i + PI / 90);
        star(pointCount, outerRadius - outerRadiusRatio * i, innerRadius - innerRadiusRatio * i);
```

```
        popMatrix();
  }
}

void star(int numPoints, float outerRadius, float innerRadius) {
  float angle = TWO_PI / numPoints;
  float halfAngle = angle / 2.0;
  beginShape();
  for (float a = 0; a < TWO_PI; a += angle) {
        float sx = cos(a) * outerRadius;
        float sy = sin(a) * outerRadius;
        vertex(sx, sy);
        sx = cos(a + halfAngle) * innerRadius;
        sy = sin(a + halfAngle) * innerRadius;
        vertex(sx, sy);
  }
  endShape(CLOSE);
}
```

---

**manala piscodélico**

```
void setup(){
  size(1000, 1000);
  background(0);

  int rows = 8;
  int cols = 8;
  float outerRadius = width / cols;

  int pointCount;
  int steps;
  float innerRadius;
  float outerRadiusRatio;
  float innerRadiusRatio;
  float shadeRatio;
  float rotationRatio;
```

```
  translate(outerRadius / 2, outerRadius / 2);
  for(int i = 0; i < rows; ++i){
        for(int j = 0; j < cols; ++j){
        pointCount = int(random(5, 15));
        steps = int(random(3, 20));
        innerRadius = outerRadius * random(.3, .9);
        outerRadiusRatio = outerRadius / steps;
        innerRadiusRatio = innerRadius / steps;
        float randCol = random(225, 255);
        shadeRatio = randCol / steps;
        rotationRatio = random(90, 200) / steps;

        pushMatrix();
        translate(outerRadius * j, outerRadius * i);
        for(int k = 0; k < steps; k++){
        fill(shadeRatio * k);
        stroke(randCol - shadeRatio * k, 100);
        pushMatrix();
        rotate(rotationRatio * k * PI / 180);
        star(pointCount, outerRadius - outerRadiusRatio * k, innerRadius -
innerRadiusRatio * k);
        popMatrix();
        }
        popMatrix();
        }
  }
}

void star(int numPoints, float outerRadius, float innerRadius) {
  float angle = TWO_PI / numPoints;
  float halfAngle = angle / 2.0;
  beginShape();
  for (float a = 0; a < TWO_PI; a += angle) {
        float sx = cos(a) * outerRadius;
        float sy = sin(a) * outerRadius;
        vertex(sx, sy);
```

```
        sx = cos(a + halfAngle) * innerRadius;
        sy = sin(a + halfAngle) * innerRadius;
        vertex(sx, sy);
  }
  endShape(CLOSE);
}

void draw(){
  setup();
}
```

---

**mandala con mouse**

```
void setup(){
  size(1000, 1000);
  background(0);

  int rows = 5;
  int cols = 5;
  float outerRadius = width / cols;

  int pointCount;
  int steps;
  float innerRadius;
  float outerRadiusRatio;
  float innerRadiusRatio;
  float shadeRatio;
  float rotationRatio;

  translate(outerRadius / 2, outerRadius / 2);
  for(int i = 0; i < rows; ++i){
        for(int j = 0; j < cols; ++j){
        pointCount = int(random(5, 15));
        steps = int(random(3, 20));
        innerRadius = outerRadius * random(.3, .9);
        outerRadiusRatio = outerRadius / steps;
```

```
        innerRadiusRatio = innerRadius / steps;
        float randCol = random(225, 255);
        shadeRatio = randCol / steps;
        rotationRatio = random(90, 200) / steps;

        pushMatrix();
        translate(outerRadius * j, outerRadius * i);
        for(int k = 0; k < steps; k++){
        fill(shadeRatio * k);
        stroke(randCol - shadeRatio * k, 100);
        pushMatrix();
        scale(0.4);
        rotate(rotationRatio * k * PI / 180);
        star(pointCount, outerRadius - outerRadiusRatio * k, innerRadius -
innerRadiusRatio * k);
        popMatrix();
        }
        popMatrix();
        }
  }
}

void star(int numPoints, float outerRadius, float innerRadius) {
  float angle = TWO_PI / numPoints;
  float halfAngle = angle / 2.0;
  beginShape();
  for (float a = 0; a < TWO_PI; a += angle) {
        float sx = cos(a) * outerRadius;
        float sy = sin(a) * outerRadius;
        vertex(sx, sy);
        sx = cos(a + halfAngle) * innerRadius;
        sy = sin(a + halfAngle) * innerRadius;
        vertex(sx, sy);
  }
  endShape(CLOSE);
}
```

```
void draw(){
  background(0);

   int rows = int(map(mouseY, 0, height, 2, 10)); // Cambia el número de filas según la posición vertical del ratón
   int cols = int(map(mouseX, 0, width, 2, 10)); // Cambia el número de columnas según la posición horizontal del ratón
  float outerRadius = width / cols;

  int pointCount;
  int steps;
  float innerRadius;
  float outerRadiusRatio;
  float innerRadiusRatio;
  float shadeRatio;
  float rotationRatio;

  translate(outerRadius / 2, outerRadius / 2);
  for(int i = 0; i < rows; ++i){
        for(int j = 0; j < cols; ++j){
        pointCount = int(random(5, 15));
        steps = int(random(3, 20));
        innerRadius = outerRadius * random(.3, .9);
        outerRadiusRatio = outerRadius / steps;
        innerRadiusRatio = innerRadius / steps;
        float randCol = random(225, 255);
        shadeRatio = randCol / steps;
        rotationRatio = random(90, 200) / steps;

        pushMatrix();
        translate(outerRadius * j, outerRadius * i);
        for(int k = 0; k < steps; k++){
        fill(shadeRatio * k);
        stroke(randCol - shadeRatio * k, 100);
        pushMatrix();
        scale(0.4);
        rotate(rotationRatio * k * PI / 180);
```

```
        star(pointCount, outerRadius - outerRadiusRatio * k, innerRadius -
innerRadiusRatio * k);
        popMatrix();
        }
        popMatrix();
        }
  }
}
```

---

```
PImage img;
PImage tintedImg;
color[] rainbow = {
  color(255, 0, 0),   // Red
  color(255, 127, 0), // Orange
  color(255, 255, 0), // Yellow
  color(0, 255, 0),   // Green
  color(0, 0, 255),   // Blue
  color(75, 0, 130),  // Indigo
  color(148, 0, 211)  // Violet
};

void setup() {
  size(800, 800); // Ajusta el tamaño según tus necesidades
  img = loadImage("usa.svg");
  img.resize(width, height);
  tintedImg = createImage(width, height, RGB);
  tintRainbow();
  tintedImg.save("prueba_tintada.png");
  noLoop(); // Para que draw() no se ejecute en bucle
}

void draw() {
  image(tintedImg, 0, 0);
}

void tintRainbow() {
```

```
  img.loadPixels();
  tintedImg.loadPixels();
  for (int y = 0; y < height; y++) {
        color c = rainbow[int(map(y, 0, height, 0, rainbow.length))];
        for (int x = 0; x < width; x++) {
        int loc = x + y * width;
        color original = img.pixels[loc];
        float r = red(original) * red(c) / 255.0;
        float g = green(original) * green(c) / 255.0;
        float b = blue(original) * blue(c) / 255.0;
        tintedImg.pixels[loc] = color(r, g, b);
        }
  }
  tintedImg.updatePixels();
}
```