EJERCICIOS GSAP Y CANVAS

EJERCICIO 1: ANIMACIÓN BÁSICA

Objetivo: Animar un elemento para que se mueva horizontalmente.

Instrucciones:

1. Crea un archivo HTML con un elemento que tenga un id="box".
2. Usa GSAP para mover el 200 píxeles a la derecha en 2 segundos.

```
gsap.to("#box", { x: 200, duration: 2 });
```

EJERCICIO 2: ANIMACIÓN DE OPACIDAD

Objetivo: Animar un elemento para que cambie su opacidad.

Instrucciones:

1. Usa el mismo archivo HTML del ejercicio anterior.
2. Haz que el cambie su opacidad de 1 a 0 en 1 segundo.

```
gsap.to("#box", { opacity: 0, duration: 1 });
```

EJERCICIO 3: ANIMACIÓN DE ESCALA

Objetivo: Animar un elemento para que cambie su tamaño.

Instrucciones:

1. Usa el mismo archivo HTML del ejercicio anterior.
2. Haz que el aumente su tamaño al doble en 2 segundos.

```
gsap.to("#box", { scale: 2, duration: 2 });
```

EJERCICIO 4: ANIMACIÓN DE ROTACIÓN

Objetivo: Animar un elemento para que rote.

Instrucciones:

1. Usa el mismo archivo HTML del ejercicio anterior.
2. Haz que el rote 360 grados en 3 segundos.

```
gsap.to("#box", { rotation: 360, duration: 3 });
```

EJERCICIO 5: ANIMACIÓN COMBINADA

Objetivo: Combinar múltiples animaciones en una sola línea de tiempo.

Instrucciones:

1. Usa el mismo archivo HTML del ejercicio anterior.
2. Crea una línea de tiempo que mueva el 200 píxeles a la derecha, cambie su opacidad a 0.5, y lo rote 180 grados, todo en 4 segundos.

```
const tl = gsap.timeline();

tl.to("#box", { x: 200, duration: 2 })

  .to("#box", { opacity: 0.5, duration: 1 })

  .to("#box", { rotation: 180, duration: 1 });
```

EJERCICIO 6: ANIMACIÓN CON CONTROL DE USUARIO

Objetivo: Crear una animación que se active al hacer clic en un botón.

Instrucciones:

1. Añade un botón al archivo HTML con un id="animateButton".
2. Configura una animación para que el se mueva 100 píxeles hacia abajo cuando se haga clic en el botón.

```
document.getElementById("animateButton").addEventListener("click", () => {

  gsap.to("#box", { y: 100, duration: 1 });

});
```

EJERCICIO 7: ANIMACIÓN INFINITA

Objetivo: Crear una animación que se repita infinitamente.

Instrucciones:

1. Usa el mismo archivo HTML del ejercicio anterior.
2. Haz que el se mueva 50 píxeles a la derecha y luego vuelva a su posición original, repitiendo este ciclo infinitamente.

```
gsap.to("#box", { x: 50, duration: 1, repeat: -1, yoyo: true });
```

EJERCICIO 8: ANIMACIÓN DE SVG

Objetivo: Animar un elemento SVG.

Instrucciones:

1. Crea un archivo HTML con un elemento SVG, como un círculo.
2. Usa GSAP para animar el círculo, cambiando su radio o posición.

```
gsap.to("#circle", { attr: { r: 50 }, duration: 2 });
```

EJERCICIO 9: ANIMACIÓN DE SCROLL

Objetivo: Crear una animación que se active al hacer scroll.

Instrucciones:

1. Usa el plugin ScrollTrigger de GSAP.
2. Configura una animación para que un elemento se mueva al hacer scroll en la página.

```
gsap.registerPlugin(ScrollTrigger);
```

```
gsap.to("#box", {
  x: 300,
  duration: 2,
  scrollTrigger: {
    trigger: "#box",
    start: "top center",
    end: "bottom center",
    scrub: true,
  },
});
```

EJERCICIO 10: ANIMACIÓN COMPLEJA CON LÍNEAS DE TIEMPO ANIDADAS

Objetivo: Crear una animación compleja utilizando líneas de tiempo anidadas.

Instrucciones:

1. Usa el mismo archivo HTML del ejercicio anterior.
2. Crea una línea de tiempo principal que contenga varias líneas de tiempo anidadas, cada una con diferentes animaciones.

```
const masterTl = gsap.timeline();


const tl1 = gsap.timeline();
tl1.to("#box", { x: 100, duration: 1 });


const tl2 = gsap.timeline();
tl2.to("#box", { y: 100, duration: 1 });


masterTl.add(tl1).add(tl2);
```

EJERCICIO 1: DIBUJAR UN RECTÁNGULO

Objetivo: Dibujar un rectángulo en el canvas.

Instrucciones:

1. Crea un archivo HTML con un elemento <canvas> con un id="myCanvas".
2. Usa JavaScript para dibujar un rectángulo en el canvas.

```
const canvas = document.getElementById('myCanvas');
const ctx = canvas.getContext('2d');
ctx.fillStyle = 'blue';
ctx.fillRect(10, 10, 150, 100);
```

EJERCICIO 2: DIBUJAR UN CÍRCULO

Objetivo: Dibujar un círculo en el canvas.

Instrucciones:

1. Usa el mismo archivo HTML del ejercicio anterior.
2. Dibuja un círculo con un radio de 50 píxeles en el centro del canvas.

```
ctx.beginPath();
ctx.arc(canvas.width / 2, canvas.height / 2, 50, 0, Math.PI * 2);
ctx.fillStyle = 'red';
ctx.fill();
```

EJERCICIO 3: DIBUJAR UNA LÍNEA

Objetivo: Dibujar una línea en el canvas.

Instrucciones:

1. Usa el mismo archivo HTML del ejercicio anterior.
2. Dibuja una línea desde la esquina superior izquierda hasta la esquina inferior derecha del canvas.

```
ctx.beginPath();
ctx.moveTo(0, 0);
ctx.lineTo(canvas.width, canvas.height);
ctx.strokeStyle = 'green';
ctx.stroke();
```

EJERCICIO 4: DIBUJAR TEXTO

Objetivo: Dibujar texto en el canvas.

Instrucciones:

1. Usa el mismo archivo HTML del ejercicio anterior.
2. Escribe "Hola, Canvas!" en el centro del canvas.

```
ctx.font = '30px Arial';
ctx.fillStyle = 'black';
ctx.textAlign = 'center';
ctx.fillText('Hola, Canvas!', canvas.width / 2, canvas.height / 2);
```

EJERCICIO 5: DIBUJAR UNA IMAGEN

Objetivo: Dibujar una imagen en el canvas.

Instrucciones:

1. Usa el mismo archivo HTML del ejercicio anterior.
2. Carga una imagen y dibújala en el canvas.

```
const img = new Image();
img.src = 'ruta/a/tu/imagen.jpg';
img.onload = () => {
  ctx.drawImage(img, 0, 0, canvas.width, canvas.height);
};
```

EJERCICIO 6: ANIMACIÓN BÁSICA

Objetivo: Crear una animación simple en el canvas.

Instrucciones:

1. Usa el mismo archivo HTML del ejercicio anterior.
2. Anima un círculo para que se mueva horizontalmente a través del canvas.

```
let x = 0;
function animate() {
  ctx.clearRect(0, 0, canvas.width, canvas.height);
  ctx.beginPath();
  ctx.arc(x, canvas.height / 2, 20, 0, Math.PI * 2);
  ctx.fillStyle = 'purple';
  ctx.fill();
  x += 2;
  if (x > canvas.width) {
    x = 0;
  }
```

```
  requestAnimationFrame(animate);
}
animate();
```

EJERCICIO 7: INTERACCIÓN CON EL USUARIO

Objetivo: Permitir al usuario dibujar en el canvas con el ratón.

Instrucciones:

1. Usa el mismo archivo HTML del ejercicio anterior.
2. Configura el canvas para que el usuario pueda dibujar en él al hacer clic y arrastrar el ratón.

```
let isDrawing = false;

canvas.addEventListener('mousedown', (e) => {
  isDrawing = true;
  ctx.beginPath();
  ctx.moveTo(e.offsetX, e.offsetY);
});

canvas.addEventListener('mousemove', (e) => {
  if (isDrawing) {
    ctx.lineTo(e.offsetX, e.offsetY);
    ctx.stroke();
  }
});

canvas.addEventListener('mouseup', () => {
  isDrawing = false;
});
```

EJERCICIO 8: DIBUJAR UN PATRÓN

Objetivo: Dibujar un patrón repetitivo en el canvas.

Instrucciones:

1. Usa el mismo archivo HTML del ejercicio anterior.
2. Dibuja un patrón de líneas diagonales en todo el canvas.

```
for (let i = 0; i < canvas.width; i += 20) {
```

```
 ctx.beginPath();
 ctx.moveTo(i, 0);
 ctx.lineTo(0, i);
 ctx.stroke();
}
```

EJERCICIO 9: DIBUJAR UN GRÁFICO SIMPLE

Objetivo: Dibujar un gráfico de barras en el canvas.

Instrucciones:

1. Usa el mismo archivo HTML del ejercicio anterior.
2. Dibuja un gráfico de barras con datos ficticios.

```
const data = [120, 190, 150, 230, 180];
const barWidth = 50;

data.forEach((value, index) => {
 ctx.fillStyle = 'orange';
 ctx.fillRect(index * (barWidth + 10), canvas.height - value, barWidth, value);
});
```

EJERCICIO 10: ANIMACIÓN CON INTERACCIÓN

Objetivo: Crear una animación que responda a la interacción del usuario.

Instrucciones:

1. Usa el mismo archivo HTML del ejercicio anterior.
2. Crea un círculo que siga el cursor del ratón.

```
canvas.addEventListener('mousemove', (e) => {
 ctx.clearRect(0, 0, canvas.width, canvas.height);
 ctx.beginPath();
 ctx.arc(e.offsetX, e.offsetY, 30, 0, Math.PI * 2);
 ctx.fillStyle = 'blue';
 ctx.fill();
});
```

EJERCICIO 11: ANIMACIÓN DE PARTÍCULAS

Objetivo: Crear una animación de partículas que se muevan aleatoriamente por el canvas.

Instrucciones:

1. Usa el mismo archivo HTML del ejercicio anterior.
2. Crea múltiples partículas que se muevan en direcciones aleatorias y reboten en los bordes del canvas.

```
class Particle {
  constructor(x, y) {
    this.x = x;
    this.y = y;
    this.vx = Math.random() * 2 - 1;
    this.vy = Math.random() * 2 - 1;
  }

  update() {
    this.x += this.vx;
    this.y += this.vy;

    if (this.x <= 0 || this.x >= canvas.width) this.vx = -this.vx;
    if (this.y <= 0 || this.y >= canvas.height) this.vy = -this.vy;
  }

  draw(ctx) {
    ctx.beginPath();
    ctx.arc(this.x, this.y, 3, 0, Math.PI * 2);
    ctx.fillStyle = 'purple';
    ctx.fill();
  }
}

const particles = [];
for (let i = 0; i < 100; i++) {
  particles.push(new Particle(Math.random() * canvas.width, Math.random() *
canvas.height));
}

function animateParticles() {
```

```
  ctx.clearRect(0, 0, canvas.width, canvas.height);
  particles.forEach(particle => {
    particle.update();
    particle.draw(ctx);
  });
  requestAnimationFrame(animateParticles);
}

animateParticles();
```

EJERCICIO 12: GRÁFICO INTERACTIVO

Objetivo: Crear un gráfico interactivo que permita al usuario ver detalles al pasar el ratón sobre las barras.

Instrucciones:

1. Usa el mismo archivo HTML del ejercicio anterior.
2. Dibuja un gráfico de barras y muestra un tooltip con el valor de la barra cuando el usuario pasa el ratón sobre ella.

```
const data = [120, 190, 150, 230, 180];
const barWidth = 50;

data.forEach((value, index) => {
  ctx.fillStyle = 'orange';
  ctx.fillRect(index * (barWidth + 10), canvas.height - value, barWidth, value);
});

canvas.addEventListener('mousemove', (e) => {
  const barIndex = Math.floor(e.offsetX / (barWidth + 10));
  if (barIndex >= 0 && barIndex < data.length) {
    ctx.clearRect(0, 0, canvas.width, canvas.height);
    data.forEach((value, i) => {
      ctx.fillStyle = 'orange';
      ctx.fillRect(i * (barWidth + 10), canvas.height - value, barWidth, value);
    });
    ctx.fillStyle = 'rgba(0, 0, 0, 0.5)';
```

```
  ctx.fillText(data[barIndex], barIndex * (barWidth + 10), canvas.height - data[barIndex] -
10);
 }
});
```

EJERCICIO 13: JUEGO SIMPLE

Objetivo: Crear un juego simple donde el usuario controle un personaje para evitar obstáculos.

Instrucciones:

1. Usa el mismo archivo HTML del ejercicio anterior.
2. Crea un juego donde el usuario controle un cuadrado para evitar obstáculos que caen desde arriba.

```
let player = { x: canvas.width / 2, y: canvas.height - 30, width: 20, height: 20, color: 'blue' };
let obstacles = [];

function createObstacle() {
 const obstacle = { x: Math.random() * canvas.width, y: 0, width: 20, height: 20, color:
'red' };
 obstacles.push(obstacle);
}

function update() {
 ctx.clearRect(0, 0, canvas.width, canvas.height);

 ctx.fillStyle = player.color;
 ctx.fillRect(player.x, player.y, player.width, player.height);

 obstacles.forEach((obstacle, index) => {
  obstacle.y += 2;
  ctx.fillStyle = obstacle.color;
  ctx.fillRect(obstacle.x, obstacle.y, obstacle.width, obstacle.height);

  if (obstacle.y > canvas.height) {
   obstacles.splice(index, 1);
  }
```

```
    if (
      player.x < obstacle.x + obstacle.width &&
      player.x + player.width > obstacle.x &&
      player.y < obstacle.y + obstacle.height &&
      player.y + player.height > obstacle.y
    ) {
      alert('Game Over!');
      document.location.reload();
    }
  });

  requestAnimationFrame(update);
}

document.addEventListener('keydown', (e) => {
  if (e.code === 'ArrowLeft') player.x -= 20;
  if (e.code === 'ArrowRight') player.x += 20;
});

setInterval(createObstacle, 1000);
update();
```

EJERCICIO 14: FRACTALES

Objetivo: Dibujar un fractal en el canvas.

Instrucciones:

1. Usa el mismo archivo HTML del ejercicio anterior.
2. Dibuja un fractal, como el copo de nieve de Koch o el triángulo de Sierpinski.

```
function drawSierpinski(ctx, x, y, size) {
  if (size < 1) return;

  const h = size * Math.sqrt(3) / 2;
  ctx.beginPath();
  ctx.moveTo(x, y);
  ctx.lineTo(x + size / 2, y + h);
```

```
  ctx.lineTo(x - size / 2, y + h);
  ctx.closePath();
  ctx.stroke();

  drawSierpinski(ctx, x, y, size / 2);
  drawSierpinski(ctx, x - size / 4, y + h / 2, size / 2);
  drawSierpinski(ctx, x + size / 4, y + h / 2, size / 2);
}

drawSierpinski(ctx, canvas.width / 2, 20, 200);
```

EJERCICIO 15: ANIMACIÓN DE TRANSFORMACIONES

Objetivo: Crear una animación que utilice transformaciones (escalado, rotación) en el canvas.

Instrucciones:

1. Usa el mismo archivo HTML del ejercicio anterior.
2. Anima un objeto para que rote y cambie de tamaño simultáneamente.

```
let angle = 0;
let scale = 1;

function animateTransformations() {
  ctx.clearRect(0, 0, canvas.width, canvas.height);
  ctx.save();
  ctx.translate(canvas.width / 2, canvas.height / 2);
  ctx.rotate(angle);
  ctx.scale(scale, scale);
  ctx.fillStyle = 'green';
  ctx.fillRect(-50, -50, 100, 100);
  ctx.restore();

  angle += 0.02;
  scale = 1 + Math.sin(angle) * 0.5;

  requestAnimationFrame(animateTransformations);
}
```

```
animateTransformations();
```

EJERCICIO 16: SIMULACIÓN FÍSICA

Objetivo: Crear una simulación física simple, como una pelota que rebota con gravedad.

Instrucciones:

1. Usa el mismo archivo HTML del ejercicio anterior.
2. Simula una pelota que cae y rebota bajo la influencia de la gravedad.

```
let ball = { x: 50, y: 50, vx: 2, vy: 0, radius: 20, gravity: 0.5, bounce: 0.7 };

function updateBall() {
  ctx.clearRect(0, 0, canvas.width, canvas.height);

  ball.vy += ball.gravity;
  ball.x += ball.vx;
  ball.y += ball.vy;

  if (ball.y + ball.radius > canvas.height) {
    ball.y = canvas.height - ball.radius;
    ball.vy = -ball.vy * ball.bounce;
  }

  if (ball.x + ball.radius > canvas.width || ball.x - ball.radius < 0) {
    ball.vx = -ball.vx;
  }

  ctx.beginPath();
  ctx.arc(ball.x, ball.y, ball.radius, 0, Math.PI * 2);
  ctx.fillStyle = 'blue';
  ctx.fill();

  requestAnimationFrame(updateBall);
}
```

```
updateBall();
```

EJERCICIO 17: ANIMACIÓN DE MORPHING

Objetivo: Crear una animación que transforme una forma en otra.

Instrucciones:

1. Usa el mismo archivo HTML del ejercicio anterior.
2. Anima un círculo que se transforma en un cuadrado y viceversa.

```
let morph = 0;
let direction = 1;

function animateMorphing() {
 ctx.clearRect(0, 0, canvas.width, canvas.height);

 const size = 100;
 const x = canvas.width / 2;
 const y = canvas.height / 2;

 ctx.beginPath();
 ctx.moveTo(x, y - size / 2);

 for (let i = 0; i < 4; i++) {
  const angle = (i + morph) * Math.PI / 2;
  const dx = Math.cos(angle) * size / 2;
  const dy = Math.sin(angle) * size / 2;
  ctx.lineTo(x + dx, y + dy);
 }

 ctx.closePath();
 ctx.stroke();

 morph += 0.01 * direction;
 if (morph > 1 || morph < 0) direction = -direction;

 requestAnimationFrame(animateMorphing);
```

```
}


animateMorphing();
```

EJERCICIO 18: VISUALIZACIÓN DE DATOS EN TIEMPO REAL

Objetivo: Crear una visualización de datos en tiempo real, como un gráfico que se actualiza con nuevos datos.

Instrucciones:

1. Usa el mismo archivo HTML del ejercicio anterior.
2. Simula datos en tiempo real y actualiza un gráfico de líneas en el canvas.

```
const dataPoints = [];
for (let i = 0; i < 50; i++) {
 dataPoints.push(Math.random() * 100);
}


function drawChart() {
 ctx.clearRect(0, 0, canvas.width, canvas.height);


 ctx.beginPath();
 ctx.moveTo(0, canvas.height - dataPoints[0]);
 for (let i = 1; i < dataPoints.length; i++) {
  ctx.lineTo(i * (canvas.width / dataPoints.length), canvas.height - dataPoints[i]);
 }
 ctx.strokeStyle = 'blue';
 ctx.stroke();
}


function updateChart() {
 dataPoints.shift();
 dataPoints.push(Math.random() * 100);
 drawChart();
 requestAnimationFrame(updateChart);
}
```

```
updateChart();
```

EJERCICIO 19: ANIMACIÓN DE GRADIENTES

Objetivo: Crear una animación que utilice gradientes para llenar formas.

Instrucciones:

1. Usa el mismo archivo HTML del ejercicio anterior.
2. Anima un rectángulo que cambia de color usando gradientes.

```
let hue = 0;

function animateGradient() {
  ctx.clearRect(0, 0, canvas.width, canvas.height);

  const gradient = ctx.createLinearGradient(0, 0, canvas.width, canvas.height);
  gradient.addColorStop(0, `hsl(${hue}, 100%, 50%)`);
  gradient.addColorStop(1, `hsl(${hue + 180}, 100%, 50%)`);

  ctx.fillStyle = gradient;
  ctx.fillRect(0, 0, canvas.width, canvas.height);

  hue += 1;
  if (hue >= 360) hue = 0;

  requestAnimationFrame(animateGradient);
}

animateGradient();
```

EJERCICIO 20: ANIMACIÓN DE PATRONES COMPLEJOS

Objetivo: Crear una animación que genere patrones complejos, como un efecto de ondas.

Instrucciones:

1. Usa el mismo archivo HTML del ejercicio anterior.
2. Anima un patrón de ondas que se mueve a través del canvas.

```
let time = 0;
```

```
function animateWaves() {
  ctx.clearRect(0, 0, canvas.width, canvas.height);

  for (let i = 0; i < canvas.width; i++) {
    const y = Math.sin((i + time) / 20) * 20 + canvas.height / 2;
    ctx.fillRect(i, y, 1, 1);
  }

  time += 1;
  requestAnimationFrame(animateWaves);
}

animateWaves();
```

VISUALIZADOR INTERACTIVO DEL CONJUNTO DE MANDELBROT UTILIZANDO EL ELEMENTO <CANVAS> EN HTML5 Y JAVASCRIPT.

El conjunto de Mandelbrot es un fractal que se puede generar iterando una función compleja y determinando si los valores divergen o no.

Ejemplo de Código

HTML

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Conjunto de Mandelbrot</title>
  <style>
    canvas {
      border: 1px solid black;
    }
  </style>
</head>
<body>
  <canvas id="mandelbrotCanvas" width="800" height="800"></canvas>
  <script src="mandelbrot.js"></script>
```

```
</body>
</html>
```

JavaScript (mandelbrot.js)

```
const canvas = document.getElementById('mandelbrotCanvas');
const ctx = canvas.getContext('2d');
const width = canvas.width;
const height = canvas.height;

let xMin = -2.0;
let xMax = 1.0;
let yMin = -1.5;
let yMax = 1.5;

function drawMandelbrot() {
  const image = ctx.createImageData(width, height);
  const pixels = image.data;

  for (let x = 0; x < width; x++) {
    for (let y = 0; y < height; y++) {
      const zx = xMin + (x / width) * (xMax - xMin);
      const zy = yMin + (y / height) * (yMax - yMin);

      let i = maxIterations;
      let nx = 0;
      let ny = 0;

      while (nx * nx + ny * ny <= 4 && i > 0) {
        const temp = nx * nx - ny * ny + zx;
        ny = 2.0 * nx * ny + zy;
        nx = temp;
        i--;
      }
```

```
      const color = i === 0 ? 0 : 360 * (maxIterations - i) / maxIterations;
      const [r, g, b] = hslToRgb(color, 100, 50);

      const index = 4 * (y * width + x);
      pixels[index] = r;
      pixels[index + 1] = g;
      pixels[index + 2] = b;
      pixels[index + 3] = 255;
    }
  }

  ctx.putImageData(image, 0, 0);
}

const maxIterations = 1000;

function hslToRgb(h, s, l) {
  h /= 360;
  s /= 100;
  l /= 100;

  let r, g, b;

  if (s === 0) {
    r = g = b = l;
  } else {
    const hue2rgb = (p, q, t) => {
      if (t < 0) t += 1;
      if (t > 1) t -= 1;
      if (t < 1 / 6) return p + (q - p) * 6 * t;
      if (t < 1 / 2) return q;
      if (t < 2 / 3) return p + (q - p) * (2 / 3 - t) * 6;
      return p;
```

```
    };

    const q = l < 0.5 ? l * (1 + s) : l + s - l * s;
    const p = 2 * l - q;
    r = hue2rgb(p, q, h + 1 / 3);
    g = hue2rgb(p, q, h);
    b = hue2rgb(p, q, h - 1 / 3);
  }

  return [Math.round(r * 255), Math.round(g * 255), Math.round(b * 255)];
}

canvas.addEventListener('click', (event) => {
  const rect = canvas.getBoundingClientRect();
  const x = event.clientX - rect.left;
  const y = event.clientY - rect.top;

  const scale = 0.5;
  const xSpan = xMax - xMin;
  const ySpan = yMax - yMin;

  const newXMin = xMin + (x / width) * xSpan * (1 - scale);
  const newXMax = xMin + (x / width) * xSpan * (1 + scale);
  const newYMin = yMin + (y / height) * ySpan * (1 - scale);
  const newYMax = yMin + (y / height) * ySpan * (1 + scale);

  xMin = newXMin;
  xMax = newXMax;
  yMin = newYMin;
  yMax = newYMax;

  drawMandelbrot();
});
```

```
drawMandelbrot();
```

Explicación del Código

1. **Configuración del Canvas:** Se establece un canvas de 800x800 píxeles.
2. **Dibujo del Conjunto de Mandelbrot:** La función drawMandelbrot calcula si cada píxel pertenece al conjunto de Mandelbrot y lo colorea en consecuencia.
3. **Interacción del Ratón:** Al hacer clic en el canvas, se calcula una nueva región para hacer zoom, centrada en el punto donde se hizo clic.
4. **Coloreado:** Se utiliza una función hslToRgb para convertir colores HSL a RGB, lo que permite una mejor visualización del fractal.