```
const CANVAS_WIDTH = 600;
const CANVAS_HEIGHT = 400;

// Current tool settings
let p; // Processing object, accessible from anywhere
let color0 = [160, 100, 50];
let color1 = [320, 100, 50];
let brushSize = 1;

function startDrawing(p) {
  // Change if you want to start with a different background,
  // or even *no background!*
  p.background(0, 0, 50);
}

let brushes = [
  // Your brushes here!
  //=========================================================
  {
    label: "🧽",
    isActive: true,
    description:
      "Eraser which erases things as you drag your mouse across them.",

    mouseDragged() {
      console.log("Erasing");
      let x = p.mouseX;
      let y = p.mouseY;
      let x1 = p.pmouseX;
      let y1 = p.pmouseY;

      if (p.mouseIsPressed) {
        p.stroke(0, 0, 50);

        p.strokeWeight(brushSize * 40 + 2);
        p.line(x, y, x1, y1);
      }
    },
  },

  {
    label: "▫",
    isActive: true,
    description:
      "A brush using lines of squares.  It uses the color on the left of the two
options and the size properties set by the sliders.  It is a 'discrete' brush due
to the fact that each individual 'frame' of the brush does not connect to the
next.",

    mouseDragged() {
      console.log("Dragged");
      let x = p.mouseX;
      let y = p.mouseY;
      let r = brushSize * 30 + 1;

      p.noStroke();
      p.fill(...color0);

      p.square(x, y, r);
```

```
    },
  },

  //=====================================================
  {
    label: "~~",
    isActive: true,
    description:
      "A basic line brush.  It uses pmouseX,pmouseY to draw to where the last mouse
position was. It uses the color on the right side so you can switch between brushes
without needing to change colour constantly. It is a continuous brush",

    draw() {
      console.log("draw");
      let x = p.mouseX;
      let y = p.mouseY;
      let x1 = p.pmouseX;
      let y1 = p.pmouseY;

      if (p.mouseIsPressed) {
        p.stroke(...color1);

        p.strokeWeight(brushSize * 30 + 2);
        p.line(x, y, x1, y1);
      }
    },
  },
  {
    label: "▫",
    isActive: true,
    description:
      "A brush that creates an image not unlike yarn twining to form a filled shape
from scratch. Uses color on the left.",

    mousePressed() {
      this.points = [];
    },

    mouseDragged() {
      console.log("drawing");

      let x = p.mouseX;
      let y = p.mouseY;
      this.points.unshift([x, y]);

      p.noFill();
      p.stroke(...color0);
        p.strokeWeight(brushSize * 3 + 2);
      p.beginShape();

      this.points
        .filter((pt, index) => index % 10 == 0)
        .forEach(([x, y]) => {
          let dx = Math.random() * 100;
          let dy = Math.random() * 10;

          p.curveVertex(x + dx, y + dy);
        });
```

```
      p.endShape();
    },
  },

  {
    label: "↱↴",
    isActive: true,
    description:
      "A brush that creates lines with slight curves. Uses the color on the
right.",

    mousePressed() {
      this.points = [];
    },

    mouseDragged() {
      let x = p.mouseX;
      let y = p.mouseY;
      this.points.unshift([x, y]);

      p.noFill();
      p.stroke(...color1);
        p.strokeWeight(brushSize * 3 + 2);
      p.beginShape();

      this.points
        .filter((pt, index) => index % 10 == 0)
        .forEach(([x, y]) => {
          let dx = Math.random();
          let dy = Math.random();
          p.curveVertex(x + dx, y + dy);
        });

      p.endShape();
    },
  },

  {
    label: "!",
    isActive: true,
    description:
      "Rotating Sticks. Uses color on the left.",
    draw() {
      console.log("draw");
      let x = p.mouseX;
      let y = p.mouseY;
      let x1 = p.pmouseX;
      let y1 = p.pmouseY;

      if (p.mouseIsPressed) {
        let t = p.millis() * 0.001;

        let size = 200;
        let count = 6;

        let r = size * Math.random();
        let theta = Math.random() * Math.PI * 2;

        let lineSize = (Math.random() + 1) * size * 0.2;
```

```
      p.noFill();
      p.stroke(...color0);
      p.strokeWeight(brushSize * 3 + 2);
      let x2 = x + r * Math.cos(theta);
      let y2 = y + r * Math.sin(theta);
      p.line(x2, y2, x1, y1);
    }
  },
},
{
  label: "🍥",
  isActive: true,
  description:
    "Rotating Curves. Uses color on the right.",
 mousePressed() {
    this.points = [];
  },

  mouseDragged() {
    console.log("draw");
    let x = p.mouseX;
    let y = p.mouseY;
    let x1 = p.pmouseX;
    let y1 = p.pmouseY;

    if (p.mouseIsPressed) {
      let t = p.millis() * 0.001;

      let size = 200;
      let count = 6;

      let r = size * Math.random();
      let theta = Math.random() * Math.PI * 2;

      let lineSize = (Math.random() + 1) * size * 0.2;
      p.noFill();
      p.stroke(...color1);
      p.strokeWeight(brushSize * 3 + 2);
      let x2 = x + r * Math.cos(theta);
      let y2 = y + r * Math.sin(theta);
      this.points.unshift([x2, y2]);
      p.beginShape();

    this.points
      .filter((pt, index) => index % 10 == 0)
      .forEach(([x2, y2]) => {
        let dx = Math.random();
        let dy = Math.random();
        p.curveVertex(x2 + dx, y2 + dy);
      });

    p.endShape();
    }
  },
},

//=====================================================

{
```

```
    label: "🖉",
    isActive: false,
    description:
      "Complicated discrete brush. It uses the color0, color1, and size properties
set by the sliders",

    setup() {
      //       Count how many times we've drawn
      this.drawCount = 0;
    },

    // Options: setup (when tool is selected), draw (every frame),
    mouseDragged() {
      //       Here I am keeping track of both the current time, and how many times
this brush has drawn

      let t = p.millis() * 0.001; // Get the number of seconds
      this.drawCount += 1;
      let x = p.mouseX;
      let y = p.mouseY;

      //       Controllable brush size
      let r = brushSize * 10 + 10;

      //       Change the brush by how many we have drawn
      // r *= 0.5 + p.noise(this.drawCount * 0.1);
      //       Change the brush by the current time
      // r *= 0.5 + p.noise(t * 10);

      //       Shadow
      p.noStroke();
      p.fill(color0[0], color0[1], color0[2] * 0.2, 0.1);
      p.circle(x, y + r * 0.15, r * 1.1);

      // Big circle
      p.noStroke();
      p.fill(color0[0], color0[1], color0[2]);
      p.circle(x, y, r);

      // Small contrast circle
      p.noStroke();
      p.fill(color1[0], color1[1], color1[2]);
      p.circle(x - r * 0.1, y - r * 0.1, r * 0.7);

      //       Highlight
      p.noStroke();
      p.fill(color1[0], color1[1], color1[2] * 1.4);
      p.circle(x - r * 0.15, y - r * 0.15, r * 0.5);
    },
  },

  //=========================================================

  {
    label: "🩹",
    description:
      "Scatter brush, places lots of dots in both colors (discrete!)",
    isActive: false,
```

```
  mouseDragged() {
    let t = p.millis() * 0.001;
    let x = p.mouseX;
    let y = p.mouseY;

    let size = 20;
    let count = 6;

    // Scale the cluster by how far we have moved since last frame
    // the "magnitude" of the (movedX, movedY) vector
    let distanceTravelled = p.mag(p.movedX, p.movedY);
    size = distanceTravelled * 2 + 10;

    // I often draw a shadow behind my brush,
    // it helps it stand out from the background
    p.noStroke();
    p.fill(0, 0, 0, 0.01);
    p.circle(x, y, size * 2);

    // Draw some dots

    for (var i = 0; i < count; i++) {
      // Offset a polar
      let r = size * Math.random();
      let theta = Math.random() * Math.PI * 2;

      let brightnessBump = Math.random() * 50 - 20;
      brightnessBump = 20 * Math.sin(t * 7);

      let opacity = Math.random() * 0.5 + 0.2;
      if (Math.random() > 0.5)
        p.fill(color0[0], color0[1], color0[2] + brightnessBump, opacity);
      else p.fill(color1[0], color1[1], color1[2] + brightnessBump, opacity);

      let circleSize = (Math.random() + 1) * size * 0.2;

      let x2 = x + r * Math.cos(theta);
      let y2 = y + r * Math.sin(theta);
      p.circle(x2, y2, circleSize);
    }
  },
},

//=========================================================

{
  label: "💕",
  description: "Emoji scatter brush",
  isActive: false,

  mouseDragged() {
    let hearts = ["♡", "▫", "♡", "💖", "♡", "♡"];
    console.log("Drag...");
    let x = p.mouseX;
    let y = p.mouseY;

    let size = 20;
    let count = 2;
```

```
    // Scale the cluster by how far we have moved since last frame
    // the "magnitude" of the (movedX, movedY) vector
    let distanceTravelled = p.mag(p.movedX, p.movedY);
    size = distanceTravelled * 2 + 10;

    // I often draw a shadow behind my brush,
    // it helps it stand out from the background
    p.noStroke();
    p.fill(0, 0, 0, 0.01);
    p.circle(x, y, size * 2);
    p.circle(x, y, size * 1);

    // Draw some emoji
    p.fill(1);

    for (var i = 0; i < count; i++) {
      // Offset a polar
      let r = size * Math.random();
      let theta = Math.random() * Math.PI * 2;
      p.textSize(size);
      let emoji = p.random(hearts);

      let x2 = x + r * Math.cos(theta);
      let y2 = y + r * Math.sin(theta);
      p.text(emoji, x2, y2);
    }
  },
},

//=========================================================
//=========================================================
{
  label: "🌱",
  isActive: false,
  description:
    "Growing brush, leaves behind a trail that .... moves each frame!",

  setup() {
    // Store all the poitns this brush has made
    this.points = [];
  },

  mouseDragged() {
    // Every time we move
    // Add a new point to the beginning of this list
    let x = p.mouseX;
    let y = p.mouseY;
    let pt = [x, y];

    // How long does this dot live?
    pt.totalLifespan = 10 + Math.random() * 10;

    // Try a longer lifespan ☺
    // pt.totalLifespan = 10 + Math.random()*100;

    pt.lifespan = pt.totalLifespan;
    this.points.push(pt);

    p.circle(x, y, 4);
```

```
    },

    draw() {
      let radius = 5;
      let t = p.millis() * 0.001;

      // Each point keeps drawing itself, as long as it has a lifespan
      this.points.forEach((pt, index) => {
        //
        pt.lifespan--;

        if (pt.lifespan > 0) {
          let pctLife = pt.lifespan / pt.totalLifespan;
          let r = radius * 0.5;
          let theta = p.noise(index, t * 0.1) * 100;

          // Grow in some direction
          pt[0] += r * Math.cos(theta);
          pt[1] += r * Math.sin(theta);

          p.noStroke();
          p.fill(color0[0], color0[1], color0[2] * 0.1, 0.1);
          p.circle(...pt, pctLife * radius * 2);

          p.fill(
            color0[0] + p.noise(index) * 40,
            color0[1],
            color0[2] * (1 - pctLife)
          );

          //           Get smaller at the end of your life
          p.circle(...pt, pctLife ** 0.2 * radius);
        }
      });
    },
  },
];
```