Access Workshop

1. Go to: **https://www.artecs.org/**

2. Click on: **Projects**

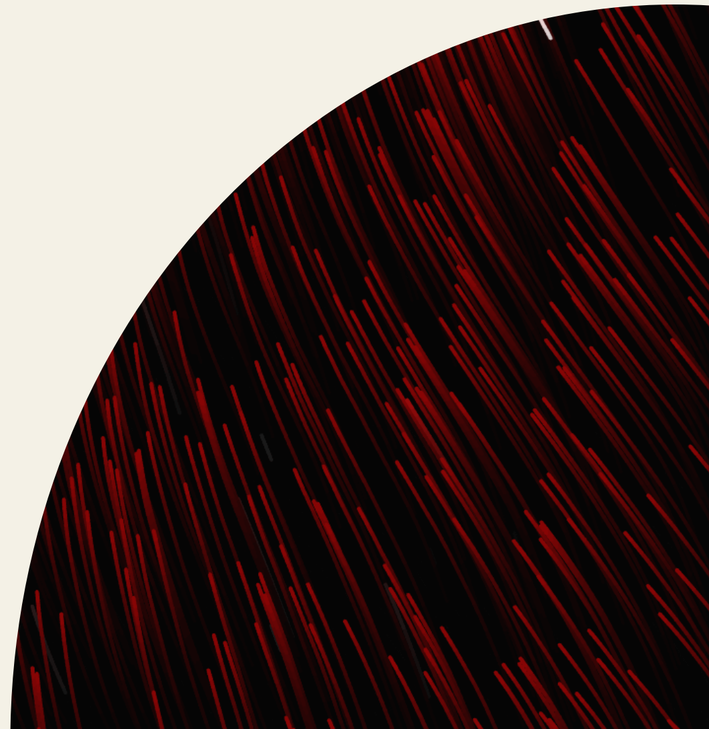
3. Select: **Workshop 2**

p5.js

ARTECS

What is P5.JS?

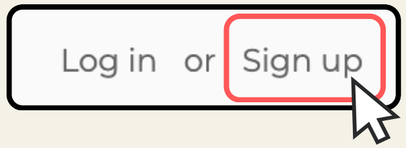
It's a **JavaScript library** that lets you create interactive animations, drawings, and even games.



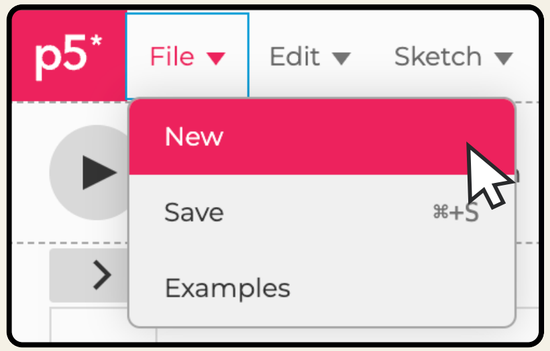
Set Up

1. Go to: **https://editor.p5js.org/**

2. Click on: **Sign up**



3. Select: **File -> New**



Step 0:

Understanding setup() and draw() Functions.

The `setup()` Function:

- Runs once at the beginning of the program.
- Used to set up initial properties, like canvas size and variables.

The `draw()` Function:

- Runs continuously after setup(), creating a loop.
- Used for animations and updating visuals on the canvas.

For example:

```
function setup() {
  createCanvas(400, 400);
}
```

Creates a 400x400 pixel canvas.

```
function draw() {
  background(220);
}
```

Sets the background color to light gray.

Step 1:

Setting Up the Canvas

To begin, let's set up a basic canvas using

the `setup()` and `draw()` functions in **P5.JS**

```
function setup() {
  createCanvas(400, 400); // Creates a 400x400 pixel canvas
}

function draw() {
  background(220); // Sets the background color to light gray
}
```

Step 2:

Adding Particles - Setup function

A **particle** is simply a point that moves around the canvas.

We need an **array** to store multiple particles.

```
// Create an empty array to store particle positions
let particles = [];

function setup() {
  createCanvas(windowWidth, windowHeight); // Create a canvas that fits the window size

  // Initialize particles with random positions
  for (let i = 0; i < 1; i++) { // Start with one particle
    particles.push(createVector(random(width), random(height))); // Store random x and y coordinates
  }
}

```

Step 2:

Adding Particles - Draw function

A **particle** is simply a point that moves around the canvas.

We need an **array** to store multiple particles.

```
12
13 function draw() {
14   background(220); // Set the background color to light gray
15
16   stroke(0); // Set the stroke color to black
17   strokeWeight(10); // Set the stroke thickness
18
19   // Loop through all particles and draw them as points
20   for (let p of particles) {
21     point(p.x, p.y); // Draw each particle at its respective position
22   }
23 }
24
```

Checkpoint: Complete code so far!

```
// Create an empty array to store particle positions
let particles = [];

function setup() {
  createCanvas(windowWidth, windowHeight); // Create a canvas that fits the window size

  // Initialize particles with random positions
  for (let i = 0; i < 1; i++) { // Start with one particle
    particles.push(createVector(random(width), random(height))); // Store random x and y coordinates
  }
}

function draw() {
  background(220); // Set the background color to light gray

  stroke(0); // Set the stroke color to black
  strokeWeight(10); // Set the stroke thickness

  // Loop through all particles and draw them as points
  for (let p of particles) {
    point(p.x, p.y); // Draw each particle at its respective position
  }
}

```

Step 3:

Moving the Particles

We use **Perlin noise** to control particle movement, giving it a fluid look.

```
12
13  const noiseScale = 0.01; // Scale factor for Perlin noise to influence movement
14  let pointSize = 10; // Size of each particle point
15
16  function draw() {
17    background(220); // Set the background color to light gray
18
19    stroke(0); // Set the stroke color to black
20    strokeWeight(pointSize); // Set the stroke thickness
21
22    // Loop through all particles and update their movement
23    for (let p of particles) {
24      let n = noise(p.x * noiseScale, p.y * noiseScale); // Get noise value based on particle position
25      let angle = TAU * n; // Convert noise value to an angle (TAU = 2 * PI)
26
27      // Update particle position using trigonometric functions
28      p.x += cos(angle);
29      p.y += sin(angle);
30
31      wrapAround(p); // Ensure particles wrap around screen edges
32      point(p.x, p.y); // Draw each particle at its respective position
33    }
34  }
```

Step 3:

Moving the Particles

We use **Perlin noise** to control particle movement, giving it a fluid look.

```
12
13  const noiseScale = 0.01; // Scale factor for Perlin noise to influence movement
14  let pointSize = 10; // Size of each particle point
15
16  function draw() {
17    background(220); // Set the background color to light gray
18
19    stroke(0); // Set the stroke color to black
20    strokeWeight(pointSize); // Set the stroke thickness
21
22    // Loop through all particles and update their movement
23    for (let p of particles) {
24      let n = noise(p.x * noiseScale, p.y * noiseScale); // Get noise value based on particle position
25      let angle = TAU * n; // Convert noise value to an angle (TAU = 2 * PI)
26
27      // Update particle position using trigonometric functions
28      p.x += cos(angle);
29      p.y += sin(angle);
30
31      wrapAround(p); // Ensure particles wrap around screen edges
32      point(p.x, p.y); // Draw each particle at its respective position
33    }
34  }
```

Checkpoint: Complete code so far!

```
// Create an empty array to store particle positions
let particles = [];

function setup() {
  createCanvas(windowWidth, windowHeight); // Create a canvas that fits the window size

  // Initialize particles with random positions
  for (let i = 0; i < 1; i++) { // Start with one particle
    particles.push(createVector(random(width), random(height))); // Store random x and y coordinates
  }
}

const noiseScale = 0.01; // Scale factor for Perlin noise to influence movement
let pointSize = 10; // Size of each particle point

function draw() {
  background(220); // Set the background color to light gray

  stroke(0); // Set the stroke color to black
  strokeWeight(pointSize); // Set the stroke thickness

  // Loop through all particles and update their movement
  for (let p of particles) {
    let n = noise(p.x * noiseScale, p.y * noiseScale); // Get noise value based on particle position
    let angle = TAU * n; // Convert noise value to an angle (TAU = 2 * PI)

    // Update particle position using trigonometric functions
    p.x += cos(angle);
    p.y += sin(angle);

    wrapAround(p); // Ensure particles wrap around screen edges
    point(p.x, p.y); // Draw each particle at its respective position
  }
}

```

Step 4:

Keeping Particles on the Screen

Particles should **not permanently disappear** when they leave the screen.

We **reset them** to a random position instead.

```
35
36 function wrapAround(p) {
37   if (p.x < 0 || p.x > width || p.y < 0 || p.y > height) {
38     p.x = random(width);
39     p.y = random(height);
40   }
41 }
```

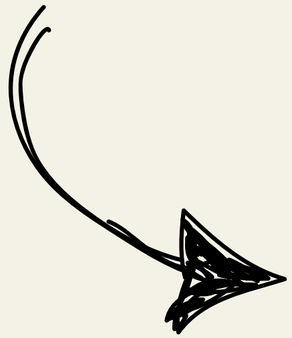
Checkpoint: Complete code so far!

```
// Create an empty array to store particle positions
let particles = [];

function setup() {
  createCanvas(windowWidth, windowHeight); // Create a canvas that fits the window size

  // Initialize particles with random positions
  for (let i = 0; i < 1; i++) { // Start with one particle
    particles.push(createVector(random(width), random(height))); // Store random x and y coordinates
  }
}

const noiseScale = 0.01; // Scale factor for Perlin noise to influence movement
let pointSize = 10; // Size of each particle point

function draw() {
  background(220); // Set the background color to light gray

  stroke(0); // Set the stroke color to black
  strokeWeight(pointSize); // Set the stroke thickness

  // Loop through all particles and update their movement
  for (let p of particles) {
    let n = noise(p.x * noiseScale, p.y * noiseScale); // Get noise value based on particle position
    let angle = TAU * n; // Convert noise value to an angle (TAU = 2 * PI)

    // Update particle position using trigonometric functions
    p.x += cos(angle);
    p.y += sin(angle);

    wrapAround(p); // Ensure particles wrap around screen edges
    point(p.x, p.y); // Draw each particle at its respective position
  }
}

function wrapAround(p) {
  if (p.x < 0 || p.x > width || p.y < 0 || p.y > height) {
    p.x = random(width);
    p.y = random(height);
  }
}
```

Step 5:
Adding Particles Trail

We use the a small **alpha value** (transparency) makes the previous frames fade out slowly.

```
background(220); // Set the background color to light gray
```



```
background(220, 10); // Set the background color to light gray
```

Thank You!