**Project Overview**

This project is a **dynamic flower animation** where multiple flowers **spawn at the bottom of the canvas**, rise upwards, and **gradually grow in size**. Each flower has:

* A **center circle** and multiple **petals** arranged in a circular pattern.
* **Randomized horizontal position**, **upward speed**, and **colors** for petals and center.
* **Fixed number of petals** (15 in this implementation).

The animation continuously spawns flowers, and old flowers are removed once they leave the top of the canvas, keeping the scene clean and smooth.

**Class Design**

**Flower Class:**

The Flower class encapsulates all behavior and properties of a flower:

**Properties:**

* x, y → Position of the flower.
* numPetals → Number of petals.
* currentScale → Current scale factor to allow growth.
* maxScale → Maximum size of the flower.
* startY, endY → Starting and ending positions to control growth.
* speed → Vertical upward speed.
* centerColor, petalColor → Randomized colors for visual variation.

**Methods:**

1. show() → Draws the flower at its current position and scale.
2. move() → Updates the vertical position and growth scale of the flower.
3. offscreen() → Checks if the flower has left the canvas.

**Challenges Faced**

While developing this project, several challenges were encountered:

1. **Maintaining Organic Growth**
   * Initially, flowers **reached their maximum size too quickly**, making the growth appear unnatural.
   * Solution: We used the **map() function** to scale the flower proportionally based on its vertical position (y). This ensured that the flower grows gradually as it rises.
2. **Preventing Oversized or Undersized Flowers**
   * Without limits, slight calculation errors in growth could make flowers **too small or too large**, distorting the animation.
   * Solution: We used **constrain()** to ensure the growth scale remains within a fixed range (0.2 to maxScale). This kept flowers visually consistent throughout their rise.
3. **Efficiently Managing Offscreen Flowers**
   * Flowers that left the canvas continued to be updated in memory, which could eventually slow down the animation.
   * Solution: The **offscreen() function** checks whether a flower has exited the top of the canvas. Once true, the flower is removed from the array, improving performance.
4. **Achieving Randomness and Variation**
   * We wanted flowers to **spawn at different positions**, **rise at different speeds**, and **have varied colors**.
   * Solution: Random values were assigned for horizontal position, speed, and colors while keeping the number of petals fixed for consistency.

**Use of Key Functions**

**1. map()**

* Used to **link the flower's vertical position (y) to its scale**.
* Ensures **gradual and proportional growth** as the flower rises.
* Example:

let progress = map(this.y, this.startY, this.endY, 0.2, this.maxScale);

* When y = startY → progress = 0.2 (small).
* When y = endY → progress = maxScale (full size).

**2. constrain()**

* Used to **limit the flower’s growth** to avoid oversized or undersized flowers.
* Example:

this.currentScale = constrain(progress, 0.2, this.maxScale);

* Guarantees that the scale **never goes below 0.2 or above maxScale**.

**3. offscreen()**

* Checks if a flower has left the top of the canvas.
* Example:

offscreen() {

return this.y < -50;

}

* Ensures **flowers are removed from the array** once they are no longer visible, keeping animation performance smooth.

**Outcome**

* Successfully implemented a **class-based system** to create multiple independently behaving flowers.
* Learned how to **encapsulate properties and methods** inside a class.
* Effectively used map(), constrain(), and offscreen() to **control growth, maintain visual consistency, and manage memory**.