Aalto University
School of Arts, Design
and Architecture

Creative Coding with Processing
Department of Art and Media
2025/2026

Programming for Visual Artists

Welcome

Arrays

Object-Oriented Programming (OOP)

BREAK

Coding tasks

Q&A

For today...

Useful Resource

Processing Reference
https://processing.org/reference

Use it to:

- check how a function works
- see examples
- find parameters and syntax

Programmers don’t memorize everything.

We constantly check documentation.

Recap

Functions!!
That was a lot already! :)

Yesterday we learned how to build reusable tools using functions.

One drawing is interesting.
Many elements create systems.

Examples:

- many circles moving
- rain
- stars
- particles

What if we want 100 circles moving?
Do we write 100 variables?

This creates the need for arrays.

Arrays

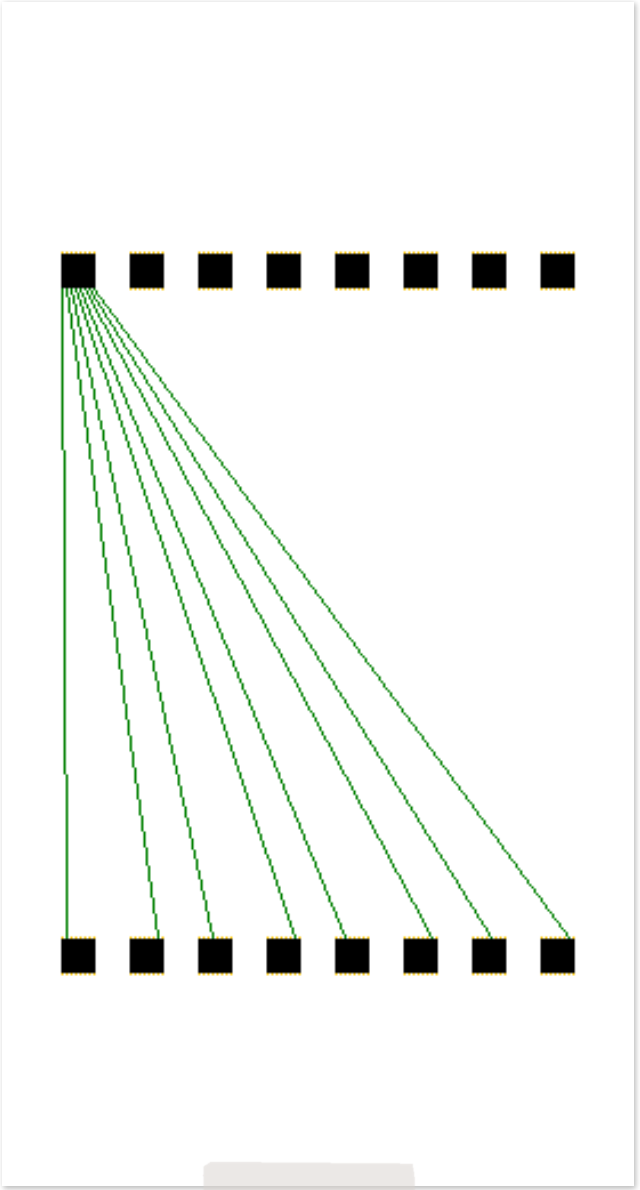
Arrays are like lists that store many values under one name.

Example:
int[] numbers = {10, 20, 30, 40, 50};

Important ideas:

- Single Type
  - An array stores items of the same type.
- Index
  - Each value has a position starting at 0.
- Fixed Size
  - The number of elements is defined when the array is created.

Index:	0	1	2	3	4
Value:	10	20	30	40	50



Arrays - static

Example: an array with fixed values

int[] numbers = {10, 20, 30, 40, 50};

Key ideas:

- numbers.length → how many elements the array has
- numbers[i] → access each value
- a for loop lets us go through the whole array

```
int[] numbers = {10, 20, 30, 40, 50}; // Array of integers

void setup() {
  size(400, 200);
  textSize(20);   // Set a larger text size
  fill(0);        // Set text color to black
}

void draw() {
  background(255); // Clear the screen with a white background

  // Loop through the array and display each number
  for (int i = 0; i < numbers.length; i++) {
    text("Number: " + numbers[i], 20, 30 * (i + 1));
  }
}
```

```
int[] sizes = {10, 20, 30, 40, 50};

void setup() {
  size(400, 200);
  noStroke();
}

void draw() {
  background(255);

  for (int i = 0; i < sizes.length; i++) {
    ellipse(60 * (i + 1), height/2, sizes[i], sizes[i]);
  }
}
```

Arrays - dynamic

Normal arrays

- fixed size

ArrayLists

- can grow
- can shrink

Example:

ArrayList<Integer> numbers;

numbers.add(10);

numbers.add(20);

numbers.add(30);

Interaction:

- Click the mouse → add a new random number

Analogy:

Arrays = fixed palette

ArrayLists = palette you can keep adding colors to

add() → adds a new element to the list

get() → retrieves an element from the list

```
ayList<Float> sizes = new ArrayList<Float>()

d setup() {
ize(400, 400);

izes.add(20);
izes.add(40);
izes.add(60);


d draw() {
ackground(255);

or (int i = 0; i < sizes.size(); i++) {
 ellipse(50 + i * 70, height/2, sizes.get(i)



d mousePressed() {
izes.add(random(10, 80));
```

Object-Oriented Programming (OOP)

OOP organizes code using objects.

Think of objects like real-life things that have:
- properties (data)
- behaviors (functions)

Class
A class is a blueprint that defines what an object is and what it can do.

Object
An object is an instance of a class.
Example: if we have a Ball class, each ball we create is an object.

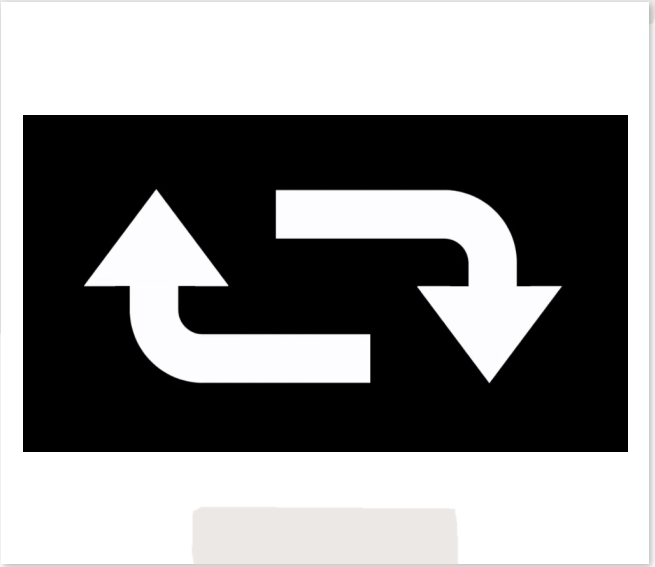
Encapsulation
Encapsulation means keeping related data and functions together inside a class.
Example:
A Ball class contains:
- its position
- how it moves
- how it is drawn

Arrays help us manage many elements.
Objects help us define what each element is.

Ball object
---
position -> properties (data)
move() -> behavior
display() ->behavior

```
class Ball {
  float x;
  float y;

  Ball(float startX, float startY) {
    x = startX;
    y = startY;
  }

  void move() {
    x += 2;
  }

  void display() {
    ellipse(x, y, 20, 20);
  }
}

Ball myBall;

void setup() {
  size(400, 400);
  myBall = new Ball(50, 200);
}

void draw() {
  background(255);
  myBall.move();
  myBall.display();
}
```

Examples of OOP for Art



Shapes as objects

- A circle, square, or line can be its own object
- Each object can have properties:
  - position, size, color
- Methods define how it moves or changes

Particles

- A Particle class can represent a small moving dot
- Many particles together create dynamic patterns

Interactivity

- Objects can respond to:
  - mouse clicks
  - Mouse hover
  - interaction with other objects

Modularity

- Complex artworks can be split into smaller parts
- Each class manages one element of the scene

Reusability

- A class can be reused in different sketches

OOP – creating objects

Creating an object in Processing is done using the new keyword and a class constructor.

Example:

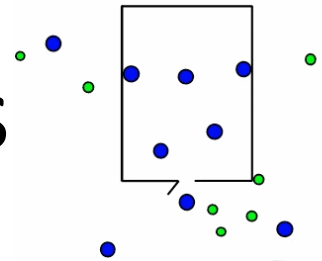
Shape myShape = new Shape(200, 200, 50);

- The **class** defines what a Shape is.
- The **new** keyword creates one Shape in the program.

```
class Shape {
  float x;
  float y;
  float size;

  // Constructor
  Shape(float startX, float startY, float startSize) {
    x = startX;
    y = startY;
    size = startSize;
  }

  // Method to draw the shape
  void display() {
    ellipse(x, y, size, size);
  }
}

Shape myShape;

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

  // Create an object from the Shape class
  myShape = new Shape(200, 200, 50);
}

void draw() {
  background(255);

  // Ask the object to draw itself
  myShape.display();
}
```

```
class Particle {
  float x;
  float y;
  float vx;
  float vy;
  float life;

  Particle(float startX, float startY) {
    x = startX;
    y = startY;
    vx = random(-1, 1);
    vy = random(-2, -0.5);
    life = 255;
  }

  void update() {
    x += vx;
    y += vy;
    life -= 2;
  }

  void display() {
    noStroke();
    fill(0, life);
    ellipse(x, y, 8, 8);
  }

  boolean isDead() {
    return life <= 0;
  }
}

ArrayList<Particle> particles = new ArrayList<Particle>();

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

void draw() {
  background(255);

  particles.add(new Particle(mouseX, mouseY));

  for (int i = particles.size()-1; i >= 0; i--) {
    Particle p = particles.get(i);
    p.update();
    p.display();

    if (p.isDead()) {
      particles.remove(i);
    }
  }
}
```

Particle systems



A particle system simulates natural phenomena like smoke, fire, rain, or sparks.

It works by managing many small objects called particles.

Concepts:

Particles
- individual elements in the system
- properties: position, velocity, size, lifespan

Emitter
- the source that creates particles
- can be a point, line, or area

Behavior
- particles can move, fade, grow, or shrink
- forces like gravity or wind can affect them

Lifespan
- particles exist for a limited time
- when they die, they are removed from the system

A particle is just an object, like the Ball class we just saw. Together, they form dynamic and organic visual patterns.

ArrayList<Particle>
many objects = particle system

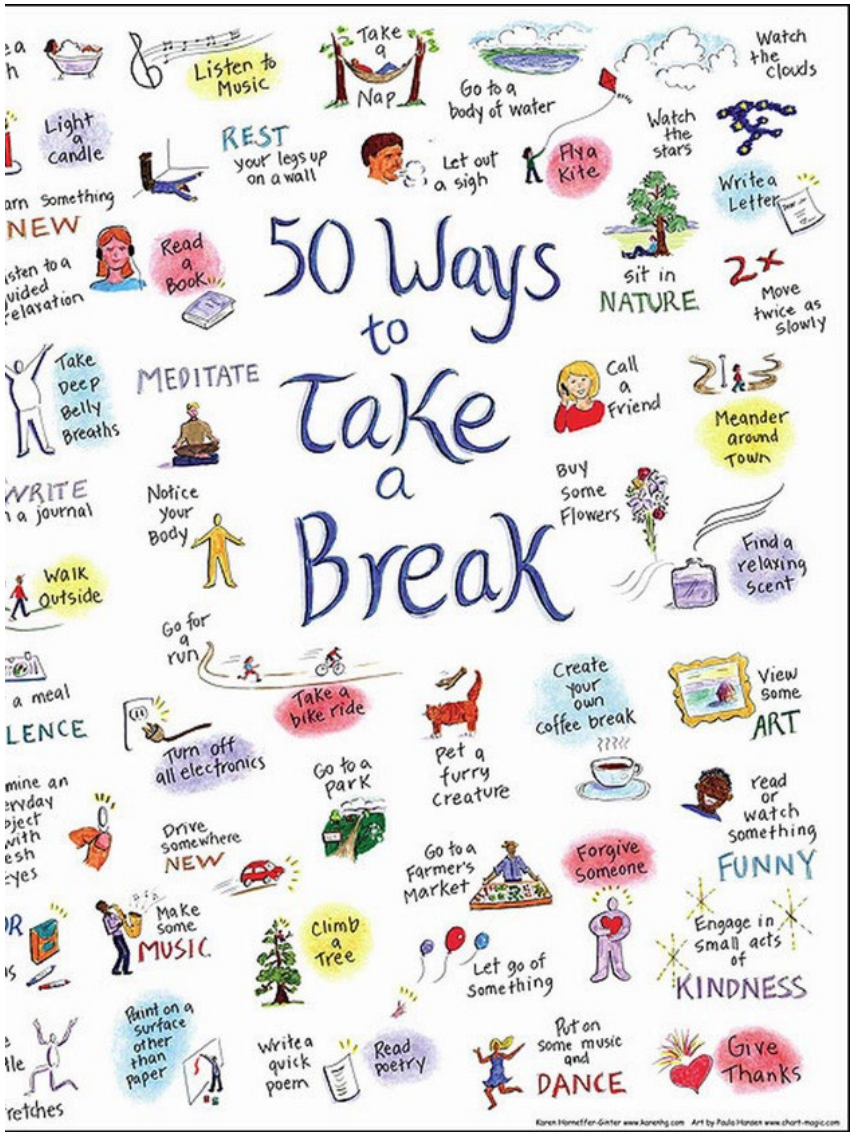
arrays → many elements
objects → structure of each element
particle systems → many objects working together

Break

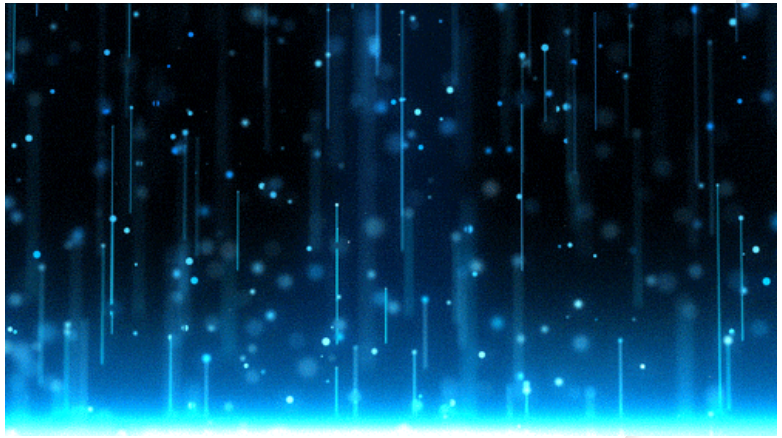
15 minutes

Stretch. Coffee. Reset.

Back at 10:45



Hands-On Exercise!



Your task

Work on the particle system exercise.

Explore how many small objects can create complex visual behaviour.

Think about:

How particles are created

How each particle moves

Why particles disappear over time

How the ArrayList manages many objects at once

Try modifying the system to create different visual effects.

Template:

Download from:
https://codeberg.org/ptiagomp/aalto-programming-visual-artists-25-26/src/branch/main/Session-06_10032026

Optional Challenge

If you finish early, try the extra exercise.

Experiment with a particle system driven by **noise()** to create flowing motion.

Observe how smooth randomness can produce organic and evolving patterns.

Discussion and Questions

What happens when one element becomes many?
How do simple rules create complex behaviour?
When does motion become a system?

Today you didn't just draw shapes.
You built systems made of many independent elements.

One particle is simple.
Many particles create dynamic visuals.

Next Week

We begin **final project pitch conversations**.
Start thinking about:

- What kind of system or behavior interests you?
- What visual ideas would you like to explore?
- What kind of interaction or motion could drive your project?

Bring references, sketches, or experiments.

Don't forget the assignments!