Aalto University
School of Arts, Design
and Architecture

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

Programming for Visual Artists

IN CASE YOU
MISSED IT

Recap - what we covered last session

- What is Creative Coding?
- Tools & Environment (Processing)
- The Boilerplate Structure (setup() / draw())
- Basic Drawing Functions
- Color, Background, and Shapes

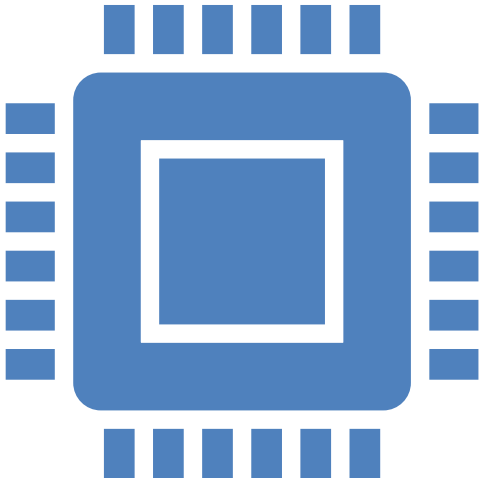
Useful Resource

Publishing

- GitHub Pages
https://pages.github.com/
- Hugo (Static Site Generator)
https://gohugo.io/

Documentation

- Processing Reference
https://processing.org/reference



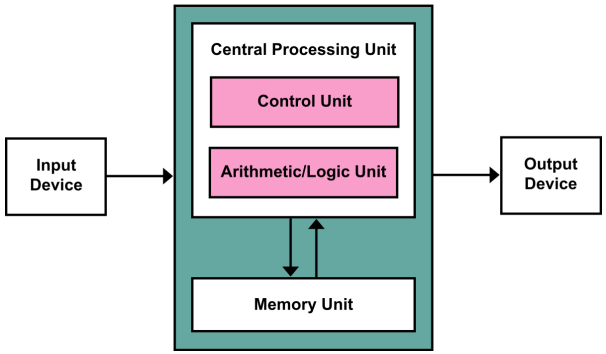
The Machine Behind the Canvas

When you write a sketch:

- The CPU interprets instructions
- The GPU translates math into pixels
- RAM holds the current state
- Storage preserves your work

Digital art is a dialogue between code and hardware.

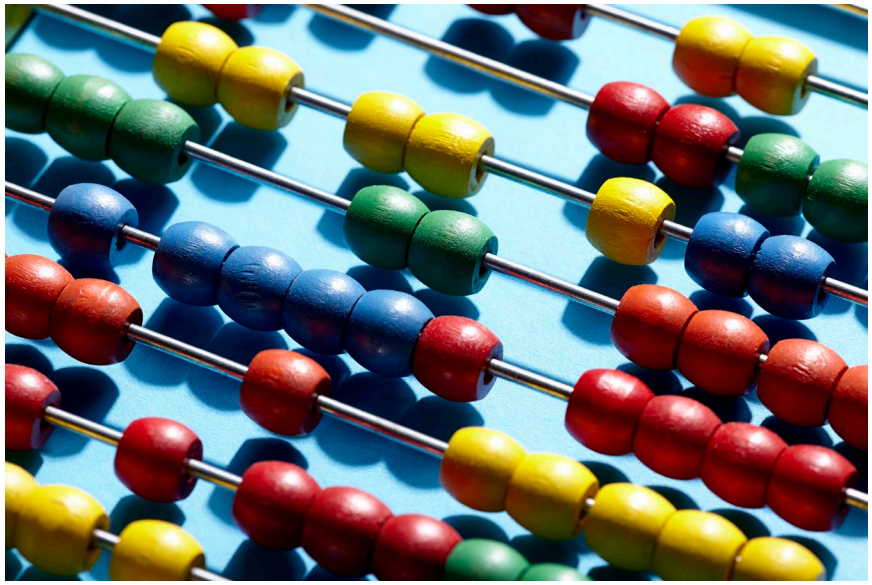
Some Basics - The Von Neumann Model



Most modern computers follow a simple principle:

- One memory stores both instructions and data.
- The CPU reads instructions from memory.
- It processes them.
- It writes results back to memory.

This cycle happens millions of times per second.



Operator	Meaning	Example
+	Addition	`x + y`
–	Subtraction	`x - y`
*	Multiplication	`x * y`
/	Division	`x / y`
%	Modulus	`x % y`

Some Basics - Arithmetic Operators (Processing)

We use these operators to:

- Move objects → x = x + 2
- Scale shapes → size * 1.2
- Create patterns → i % 2
- Animate → angle += 0.05
- That connects math to motion.

Welcome & Recap

Spatial Thinking (Coordinates)

Colour in Digital Systems

Interaction & Animation

Break

Hands-on Coding

Reflection & Discussion

Preview of Tomorrow

For today...

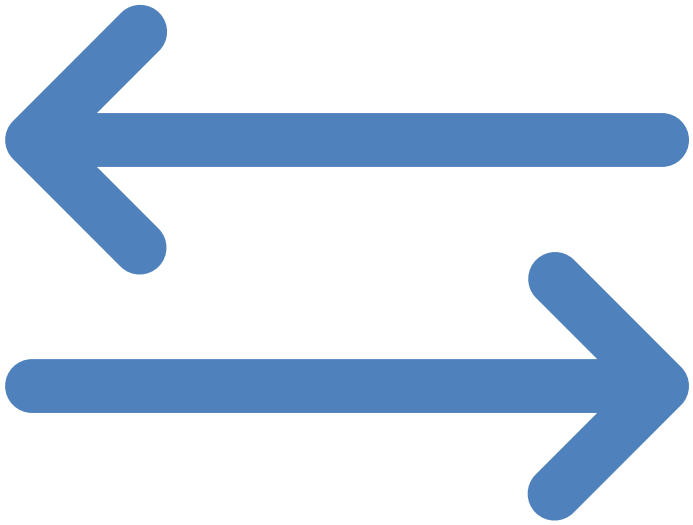
Drawing & Interaction

mouseX & mouseY

- mouseX → horizontal mouse position
- mouseY → vertical mouse position

These values update automatically while the sketch runs.

They allow your program to react to movement in real time.



Drawing & Interaction

When you move the mouse:

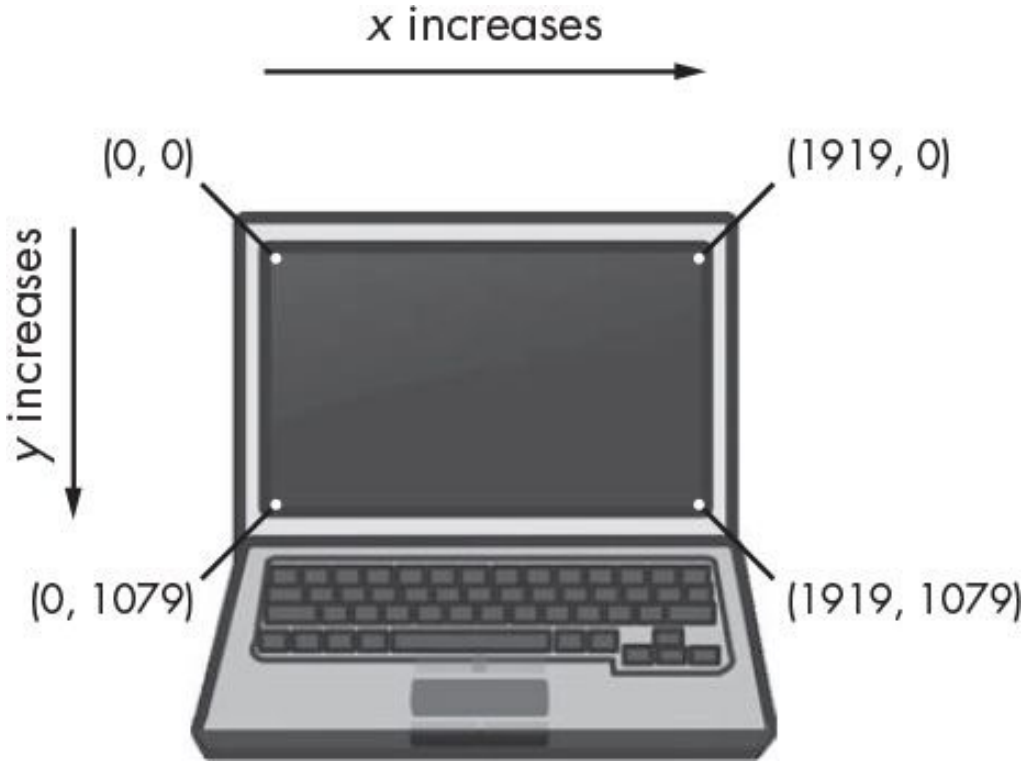
- mouseX updates
- mouseY updates

The circle is drawn at those coordinates.

Because draw() runs continuously,
the circle follows the cursor in real time.

```
void setup() {
  // Initialize the canvas with a width of 800 pixels and a height of 600 pixels
  size(800, 600);
}

void draw() {
  // Clear the screen by setting the background color to white
  background(255);

  // Set the fill color to black for the shapes
  fill(0);

  // Draw a circle at the current mouse position
  // The circle has a diameter of 50 pixels
  ellipse(mouseX, mouseY, 50, 50);
}
```

Interaction - coordinates



The Processing Coordinate System

- Origin (0, 0) → top-left corner
- x increases → to the right
- y increases → downward

Example (800 × 600 canvas)

- (800, 0) → top-right
- (0, 600) → bottom-left
- (800, 600) → bottom-right
- Negative values → off-screen.

The screen is a measurable space.
Every pixel has an address.

If I write:

- ellipse(0, 0, 50, 50); Where will it appear?
- What about ellipse(width/2, height/2, 50, 50);?

Drawing & Interaction - Coordinates in Practice

- This example uses rect() to draw a square and ellipse() to draw a circle.
- The red square is positioned at (0,0), while the blue circle is placed at (width, height) to stay at the bottom right.
- Using width and height ensures correct positioning even if the canvas size changes.
- These principles apply to more complex designs and animations.
- To dynamically adapt the sketch to the full screen, use displayWidth and displayHeight for the screen resolution.
- In newer versions of Processing this was substituted by fullScreen()!

```
void setup() {
  // Set up the canvas with a width of 800 pixels and a height of 600 pixels
  size(800, 600);
}

void draw() {
  // Clear the screen by setting the background color to white
  background(255);

  // Set the fill color to red
  fill(255, 0, 0);

  // Draw a square at the top-left corner (0,0) with a side length of 100 pixels
  rect(0, 0, 100, 100);

  // Set the fill color to blue
  fill(0, 0, 255);

  // Draw a circle at the bottom-right corner
  // The center of the circle is placed at (width, height)
  // However, this causes part of the circle to go off-screen.
  ellipse(width, height, 100, 100);
}
```

```
void setup() {
  // Set up the canvas to match the maximum screen width and height
  size(displayWidth, displayHeight);
}
```

Color in Digital Space

```
void setup() {
  // Initialize the canvas with a width of 800 pixels and a heig
  size(800, 600);
}

void draw() {
  // Clear the screen and set the background color to white
  background(255);

  // Set the fill color to a semi-transparent blue
  // RGB values: (0, 0, 255) → Blue
  // Alpha value: 100 (semi-transparent)
  fill(0, 0, 255, 100);

  // Draw a circle at the center of the canvas
  // Center position: (width/2, height/2)
  // Diameter: 200 pixels
  ellipse(width / 2, height / 2, 200, 200);
}
```

In digital systems:

- Hue becomes numeric values.
- Light is additive (RGB).
  - Red (0–255)
  - Green (0–255)
  - Blue (0–255)
- Transparency allows visual layering (Alpha Channel).
  - fill(0, 0, 255, 100);
  - The last number controls transparency.
  - 0 → fully transparent
  - 255 → fully opaque
  - This allows layering and blending effects.

Processing uses additive color mixing:

Red + Green + Blue = White.

Drawing & Interaction - Event Functions

- Processing allows you to respond to specific events.
- In this example,
  - mouseMoved() runs only when the mouse moves.
  - It updates the circle's position.
  - draw() continuously renders the circle.

Interaction = event → update → redraw.

- You can respond to:
  - Mouse movement
  - Mouse clicks
  - Keyboard input
- Event functions modify variables.
  - draw() displays the result.

```
// Variables to store the circle's position
float circleX = 0;
float circleY = 0;

// Circle size (diameter)
int circleSize = 50;

void setup() {
  // Initialize the canvas with a width of 800 pixels and a height of 600 pixels
  size(800, 600);
}

void draw() {
  // Clear the screen by setting the background color to white
  background(255);

  // Set the fill color to black for the circle
  fill(0);

  // Draw the circle at its current position
  ellipse(circleX, circleY, circleSize, circleSize);
}

void mouseMoved() {
  // Update the circle's position to match the mouse cursor
  circleX = mouseX;
  circleY = mouseY;
}
```

Drawing & Interaction - Keyboard Input

keyPressed() runs when a key is pressed.

In this example:

- Arrow keys update the square's position.
- The position variables change.
- draw() renders the updated square.

Movement happens because variables change.

Letters use key, arrow keys use keyCode.

```
// Variables to store the square's position
float squareX = 0;
float squareY = 0;

// Square size (width and height)
int squareSize = 50;

void setup() {
  // Initialize the canvas with a width of 800 pixels and a height of 600 pixels
  size(800, 600);
}

void draw() {
  // Clear the screen by setting the background color to white
  background(255);

  // Set the fill color to red for the square
  fill(255, 0, 0);

  // Draw the square at its current position
  rect(squareX, squareY, squareSize, squareSize);
}

void keyPressed() {
  // Check if the user pressed an arrow key and move the square accordingly
  if (key == CODED) { // CODED is required for special keys like arrow keys
    if (keyCode == UP) {
      squareY -= 10; // Move up
    } else if (keyCode == DOWN) {
      squareY += 10; // Move down
    } else if (keyCode == LEFT) {
      squareX -= 10; // Move left
    } else if (keyCode == RIGHT) {
      squareX += 10; // Move right
    }
  }
}
```

Drawing & Interaction - Event Functions

Mouse Events

- mousePressed()
- mouseReleased()
- mouseClicked()
- mouseMoved()
- mouseDragged()
- mouseWheel()

Keyboard Events

- keyPressed()
- keyReleased()
- keyTyped()

You don't need to memorize these.

Just remember: If something happens, there's probably an event function for it.

Drawing & Interaction - Animation Concepts

Animation = change over time.

Some common techniques:

- Easing → smooth motion
- Keyframes → defined states over time
- Looping & bouncing → repeating motion
- Particle systems → many small moving elements
- Timeline-based animation → coordinated changes

We will explore some of these later.

Drawing & Interaction – some examples

Easing Functions (Smooth Movement)

```
float x, targetX;
float easing = 0.05; // Controls how fast it eases

void setup() {
  size(800, 600);
  x = width / 2;
}

void draw() {
  background(255);

  // Move towards the target position with easing
  x += (targetX - x) * easing;

  fill(0);
  ellipse(x, height / 2, 50, 50);
}

void mousePressed() {
  // Set target position to mouse click
  targetX = mouseX;
}
```

Drawing & Interaction – some examples

Looping & Bouncing (Back-and-Forth Motion)

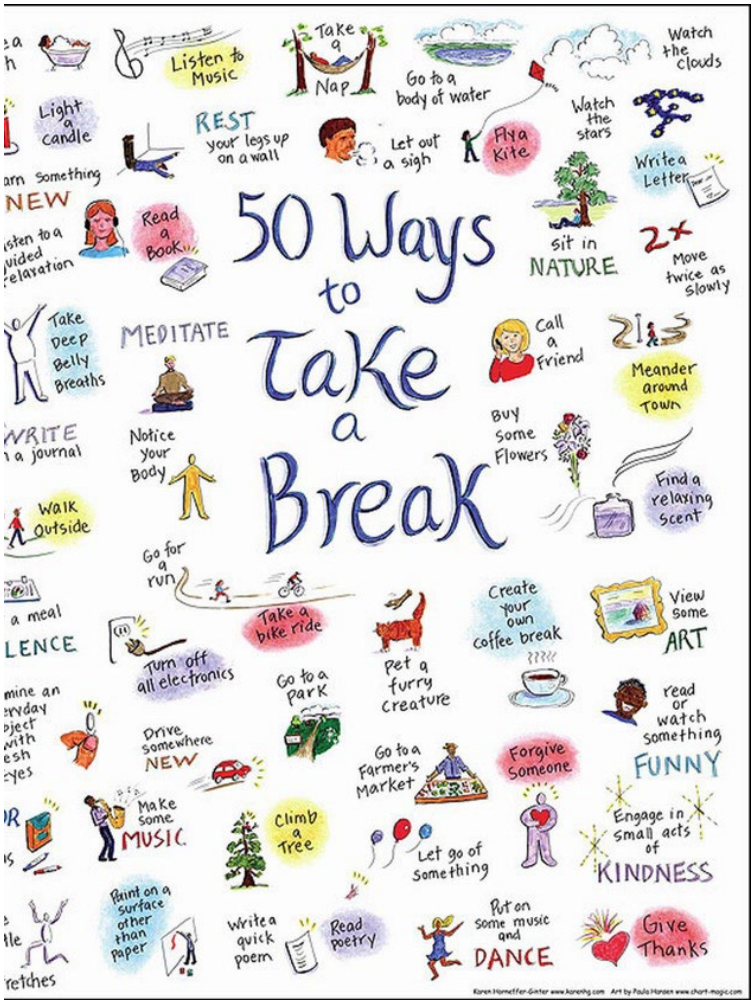
```
float x = 100;
float speed = 5;

void setup() {
  size(800, 600);
}

void draw() {
  background(255);

  // Move and bounce when hitting screen edges
  x += speed;
  if (x > width - 50 || x < 0) {
    speed *= -1; // Reverse direction
  }

  fill(0);
  ellipse(x, height / 2, 50, 50);
}
```

Break

15 minutes

Stretch. Coffee. Reset.

Back at 10:45



Hands-On Exercise!

Your task

- Make a shape follow the mouse.
- Change its color based on mouse position.
- Add a second shape with inverted movement.
- Create a dynamic background.

Experiment. Break things. Explore.

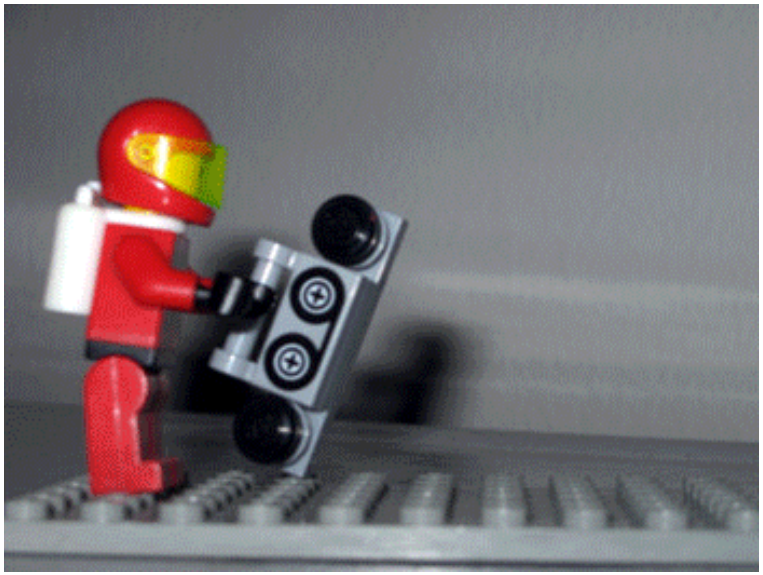
Template

- Download from: https://codeberg.org/ptiagomp/aalto-programming-visual-artists-25-26/src/branch/main/Session-02_24022026

Optional Challenge

If you finish early:

- Add transparency
- Add keyboard interaction
- Add bouncing
- Create a small composition



Discussion and Questions

What changed your perception of code today?
Where did control become expression?
Where did logic become visual?

Next Week

- Control Structures (loops, conditionals)
- Transformations.
- Functions
- Modular code.

Don’t forget the assignments!