



**Aalto University
School of Arts, Design
and Architecture**

Programming for Visual Artists

2024/2025
Department of Art
and Media



Recap



**IN CASE YOU
MISSED IT**



- Some Creative Coding background
- Tools
- Boilerplate
- Processing 101
 - Colors, background, shapes

Some things to check

GitHub Pages: <https://pages.github.com/>

Hugo: <https://gohugo.io/>

Processing reference:
<https://processing.org/reference>



Welcome

Recap

Today's Goals

Drawing and Interaction

Cartesian Coordinate System

Colour Theory basics

Interactivity

BREAK (10:30–10:45)

Coding tasks

Shapes follow the mouse and change colour

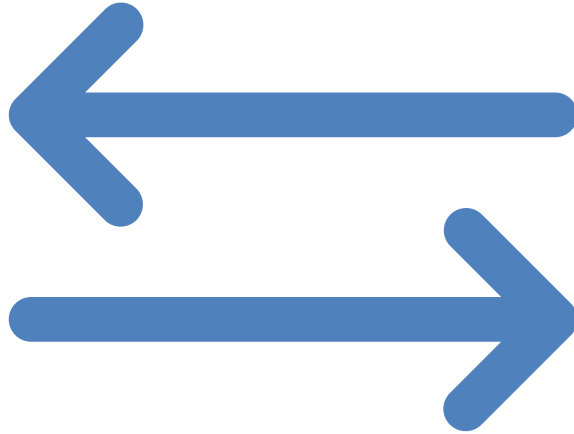
Invert movement and dynamic background

Q&A

Brief display of sketches

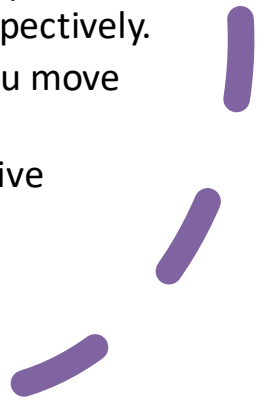
For tomorrow...

For tomorrow...



Drawing & Interaction

- `mouseX` and `mouseY` are built-in variables in Processing that store the current horizontal (`x`) and vertical (`y`) position of your mouse cursor, respectively.
- These values are updated automatically as you move your mouse around within the window.
- You can use these variables to create interactive experiences in your Processing sketches.



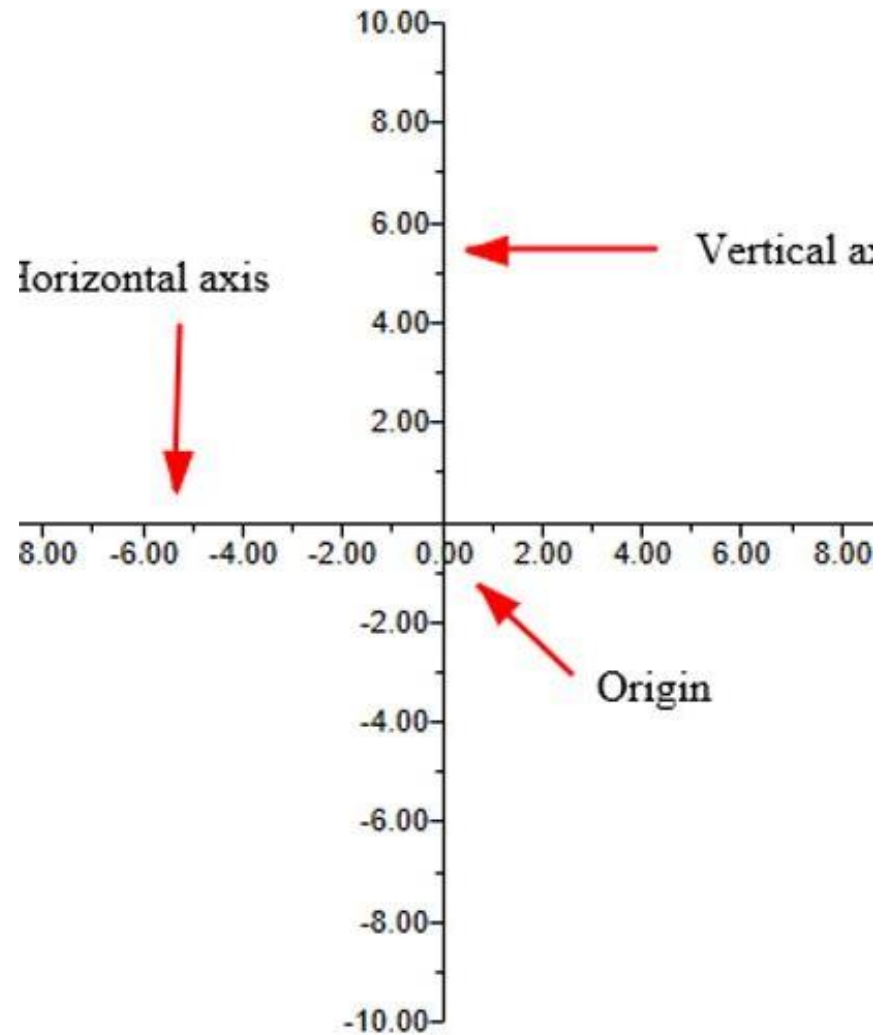
Drawing & Interaction

- In this example, whenever you move your mouse around within the window, a black circle will follow its cursor.
- The `ellipse()` function is called with `mouseX` and `mouseY` as arguments, which means that it's drawing the circle at the current position of the mouse.
- Since these positions are updated continuously in real-time, this creates an interactive experience where moving your mouse results in a visual response on the screen.

```
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

- In Processing (and others too), the coordinate system is Cartesian, with the **x-axis** representing **width** and the **y-axis** representing **height**.
- In Processing, the origin **(0,0)** is at the top-left corner. The **x-axis** increases to the right, and the **y-axis** increases **downward**.
- For an 800×600 canvas, **(800,0)** is the top-right corner, and **(0,600)** is the bottom-left. Negative coordinates position elements off-screen.
- Understanding the coordinate system helps accurately position objects in Processing. By specifying **(x, y)** coordinates, objects maintain their placement, enabling dynamic and responsive designs that adapt to different screen sizes.



Drawing & Interaction - coordinates

```
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);  
}
```

- 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.

Drawing & Interaction - color theory basics

- Colour theory studies properties like hue, saturation, brightness, and contrast.
- Colours enhance visuals and convey meaning.
- The RGB model (Red-Green-Blue) represents colours using values from 0 to 255, allowing over 16 million combinations.
- An optional alpha channel (0–255) controls transparency for blending effects.

```
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 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);  
}
```

Drawing & Interaction - interactivity

```
// 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;
}
```

- In Processing, you can handle mouse and keyboard events to create interactive animations that respond to user input in real-time.
- In this example, the **mouseMoved()** function updates a black circle's position to match the cursor, creating an animation where the circle follows the mouse.

Drawing & Interaction - interactivity

- The `keyPressed()` function detects when a user presses an arrow key, updating a red square's position to create movement.
- Handling mouse and keyboard events allows for dynamic, interactive animations in Processing.
- Experimenting with event handlers enhances interactivity.

```
// 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 - interactivity



Some Mouse Event Functions in Processing:

`mousePressed()`
`mouseReleased()`
`mouseClicked()`
`mouseMoved()`
`mouseDragged()`
`mouseWheel(MouseEvent event)`

Some Keyboard Event Functions in Processing:

`keyPressed()`
`keyReleased()`
`keyTyped()`

Drawing & Interaction – about animation

Animation in Processing enables dynamic visuals.

Key techniques include:

- **Easing functions:** Smooth transitions by controlling animation speed.
(Smooth Movement)
- **Keyframes:** Define specific points in time for smooth interpolation.
(Smooth Animation between Points)
- **Looping & bouncing:** Repeat sequences or add elasticity for lively effects.
(Back-and-Forth Motion)
- **Particle systems:** Simulate multiple particles for effects like smoke or fire.
(Simulating Multiple Particles)
- **Timeline-based animation:** Define property changes over time for complex transitions.
(Controlling Multiple Properties Over Time)

These techniques enhance animation complexity and visual appeal.



Easing Functions (Smooth Movement)

Drawing & Interaction – some examples

```
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;
}
```

Keyframes (Smooth Animation between Points)

Drawing & Interaction – some examples

```
float[] keyframes = {100, 300, 500, 700}; // Keyframe positions
int index = 0;
float x;

void setup() {
  size(800, 600);
  x = keyframes[0];
}

void draw() {
  background(255);

  // Interpolate towards the next keyframe
  x = lerp(x, keyframes[index], 0.05);

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

void mousePressed() {
  // Move to the next keyframe
  index = (index + 1) % keyframes.length;
}
```


Looping & Bouncing
(Back-and-Forth Motion)

Drawing & Interaction – some examples

```
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);
}
```

Particle System (Simulating Multiple Particles)

Drawing & Interaction – some examples

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

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

void draw() {
  background(255);

  // Add a new particle at mouse position
  particles.add(new Particle(mouseX, mouseY));

  // Update and display particles
  for (int i = particles.size() - 1; i >= 0; i--) {
    Particle p = particles.get(i);
    p.update();
    p.display();
    if (p.lifespan <= 0) {
      particles.remove(i); // Remove faded particles
    }
  }
}

class Particle {
  float x, y, speedX, speedY, lifespan;

  Particle(float x, float y) {
    this.x = x;
    this.y = y;
    this.speedX = random(-2, 2);
    this.speedY = random(-3, -1);
    this.lifespan = 255;
  }

  void update() {
    x += speedX;
    y += speedY;
    lifespan -= 5;
  }

  void display() {
    fill(0, lifespan);
    ellipse(x, y, 10, 10);
  }
}
```

Timeline-Based Animation

(Controlling Multiple Properties Over Time)

Drawing & Interaction – some examples

```
float startX = 100, endX = 700;
float startY = 300, endY = 100;
int duration = 120; // Frames to complete the animation

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

void draw() {
  background(255);

  // Calculate progress based on frame count
  float t = (frameCount % duration) / float(duration);

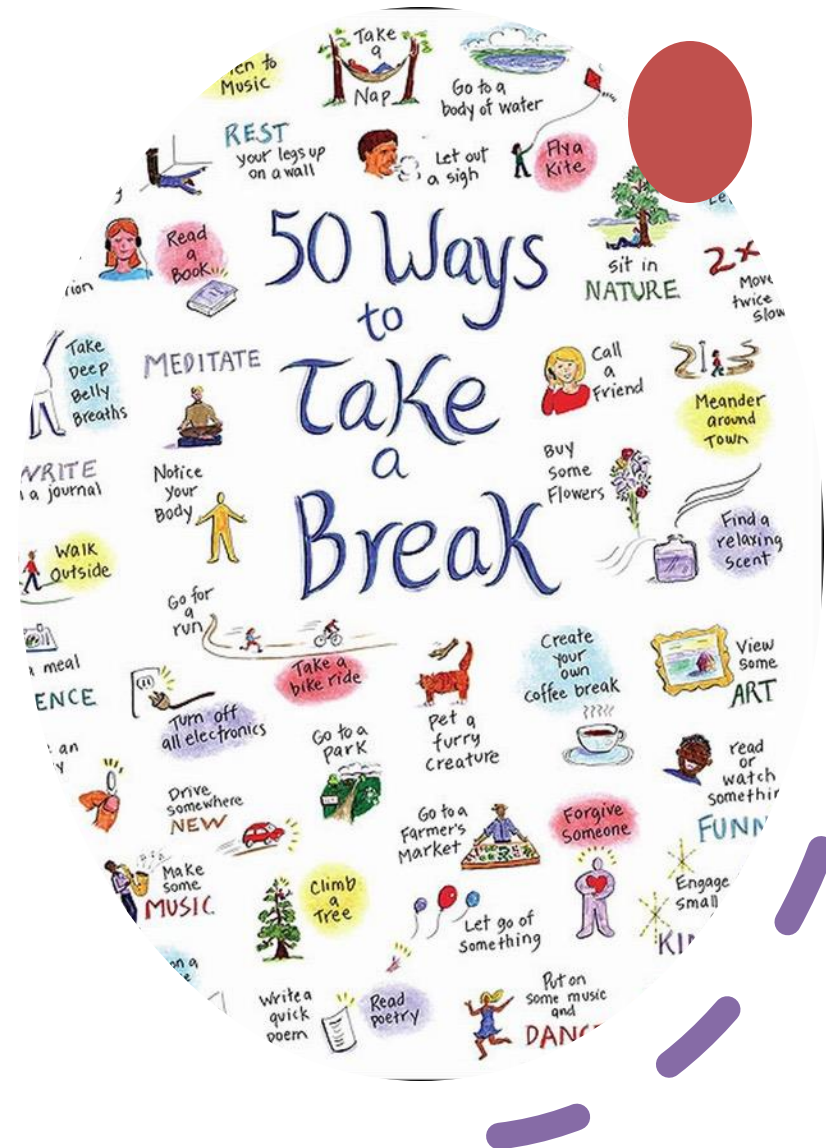
  // Interpolate position over time
  float x = lerp(startX, endX, t);
  float y = lerp(startY, endY, t);

  fill(0);
  ellipse(x, y, 50, 50);
}
```

Break

15 min.!

Please don't be late!



Hands-On Exercise!

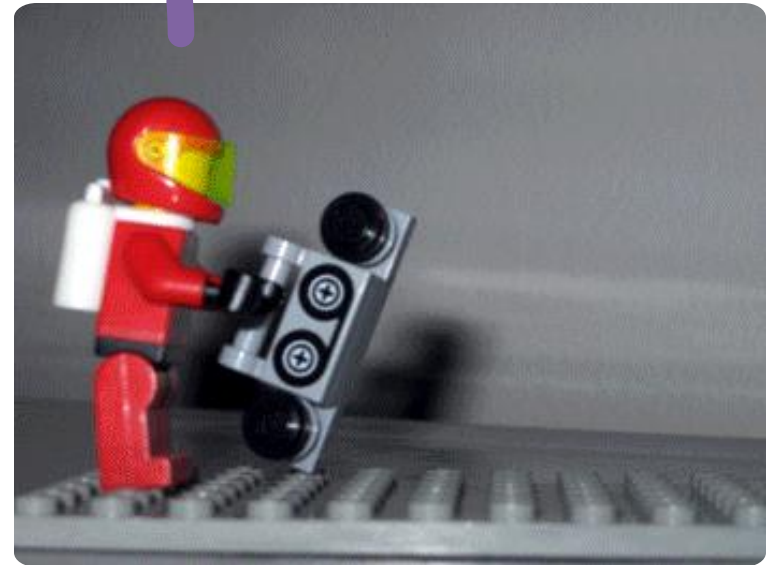
Objectives:

- Shapes follow the mouse or change colour with mouse position.
- Invert movement for second shape, and dynamic background.
- Experiment! Try out things!

Get an example from here:

https://github.com/ptiagomp/aalto-programming-visual-artists-24-25/tree/main/Session-02_25022025

(if bored, check for the “extra” files!)





FEEDBACK

Discussion & Q&A

Share your feedback!

Am I going too fast or too slow? Is this too easy or too hard?

Next week's topics:

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

Don't forget the assignments!

