Aalto University
School of Arts, Design
and Architecture

Programming for Visual Artists

2024/2025
Department of Art and Media

Some things to check

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

Recap

IN CASE YOU
MISSED IT

- Functions
- OOP intro
- Classes
- Arrays
- Particle systems

Today…

Welcome

- Recap
- Today's Goals

P5.js vs Processing

- Differences
- Translating code
- p5.js Integration with my webpage

Images

Video

Examples

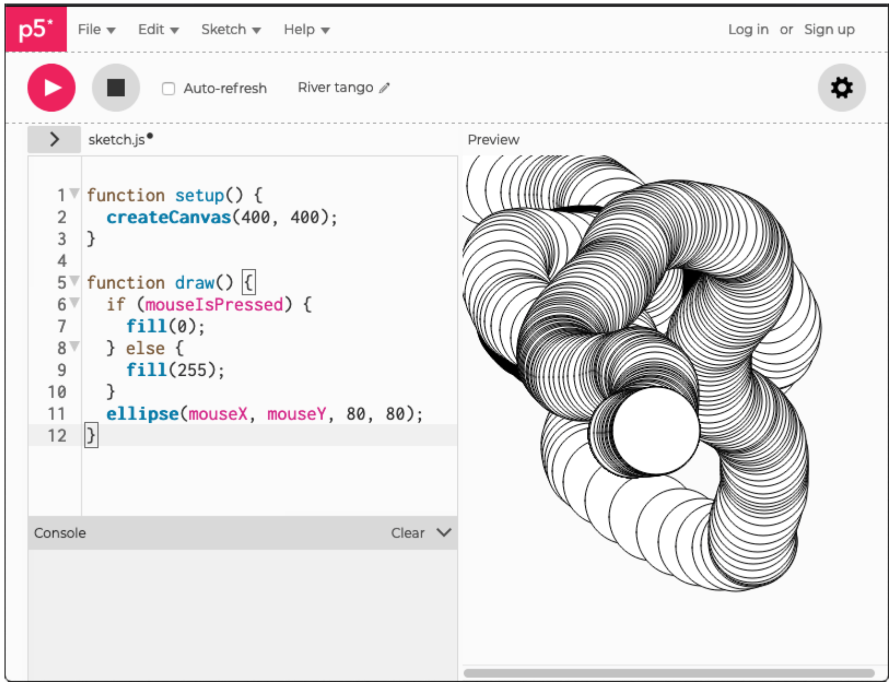
BREAK (10:30–10:45)

Coding tasks

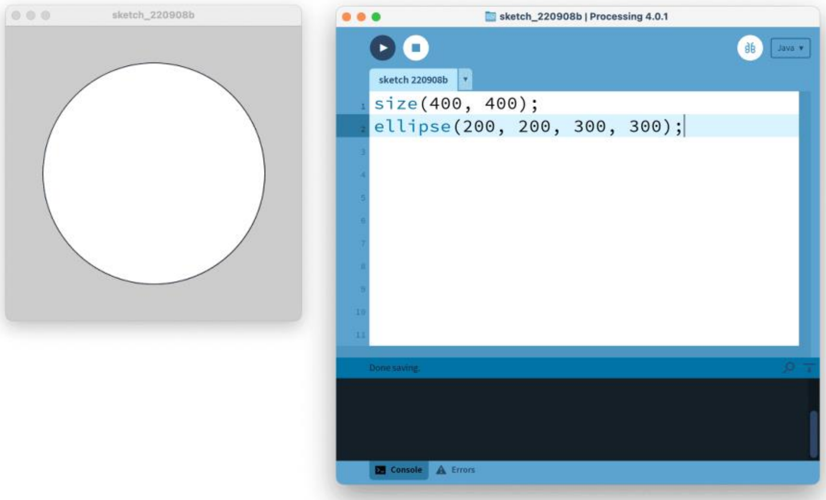
- 3 bursts of particles
- Particles & Perlin Noise

Q&A

- For tomorrow…



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

function draw() {
  if (mouseIsPressed) {
    fill(0);
  } else {
    fill(255);
  }
  ellipse(mouseX, mouseY, 80, 80);
}
```



```
size(400, 400);
ellipse(200, 200, 300, 300);
```

p5.js vs Processing

p5.js vs Processing - differences

p5.js: Uses **JavaScript**, making it suitable for web-based applications.
Processing: Uses **Java-based syntax** but also supports Python and other modes through the Processing Development Environment (PDE).

p5.js: Runs in a **web browser**, using HTML5 Canvas. This makes it accessible across devices without additional software installation.
Processing: Runs as a **standalone desktop application** and is compiled into Java bytecode, requiring Java to execute.

p5.js: Slightly slower for complex computations because it runs in a web environment and is interpreted by the browser.
Processing: Generally faster, as Java is a compiled language with better memory management and execution speed.

p5.js: Uses the **HTML5 Canvas API** and WebGL for rendering.
Processing: Uses **Java's Graphics2D and OpenGL** for rendering.

p5.js: Easily integrates with HTML, CSS, and JavaScript frameworks, making it ideal for interactive web projects.
Processing: Can integrate with Java libraries but is less suited for direct web-based interactions.

p5.js vs Processing – differences (2)

p5.js: Has a growing ecosystem of libraries, mainly for web-based interactions.
Processing: Has a more mature set of libraries for computer vision, physics, and 3D rendering.

p5.js: Limited support for hardware interactions (can use Web Serial API).
Processing: Can interface with **Arduino, Kinect, Raspberry Pi**, and other hardware through libraries.

p5.js: Best for **web-based visualizations, interactive art, and online education**.
Processing: Best for **installation art, computational design, and projects needing hardware interaction**.

p5.js: Easier for web developers familiar with JavaScript. **Processing**: More accessible for those new to coding, as Java is more structured.

Both have strong communities, but Processing **has been around longer** (since 2001), so it has more legacy resources. p5.js has a very active web-based community with modern documentation.

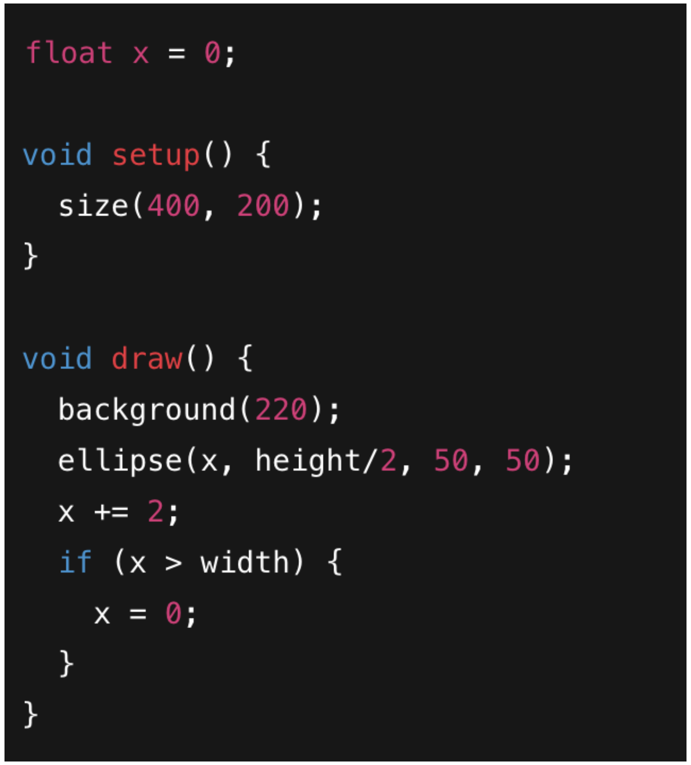
Which One to Use?

Use p5.js if you want to create **interactive web-based projects**.

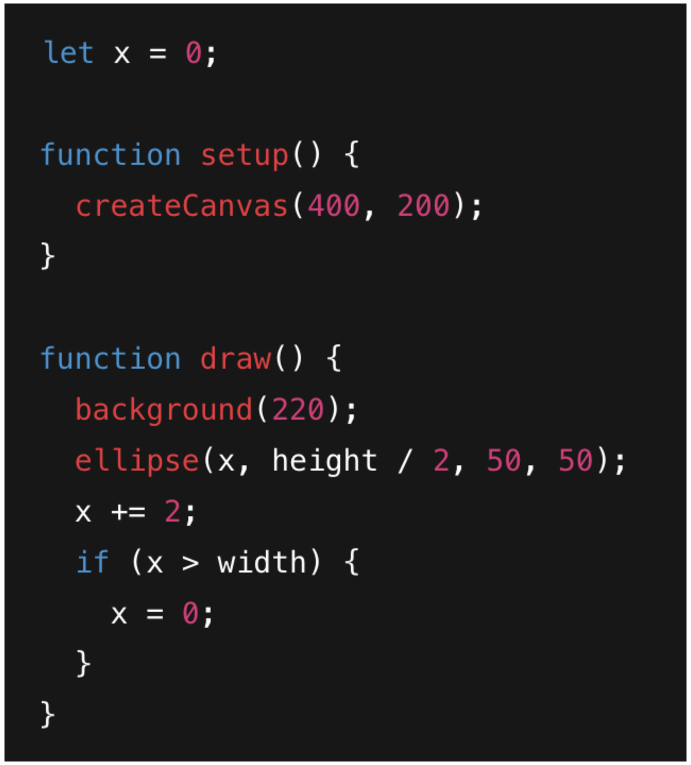
Use Processing if you need **more computational power or hardware integrations**.

p5.js vs Processing – translating code

Ex: Moving Circle

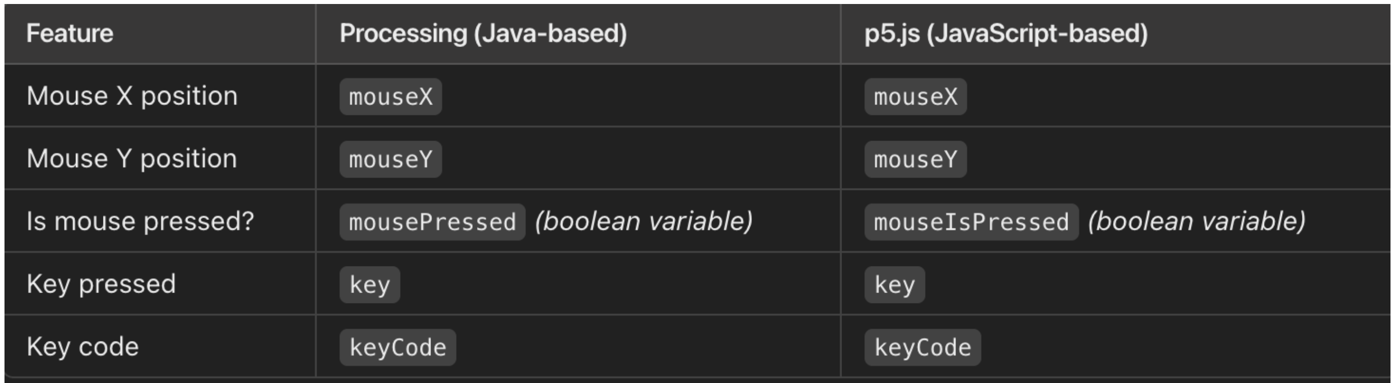


```
float x = 0;

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

void draw() {
  background(220);
  ellipse(x, height/2, 50, 50);
  x += 2;
  if (x > width) {
    x = 0;
  }
}
```

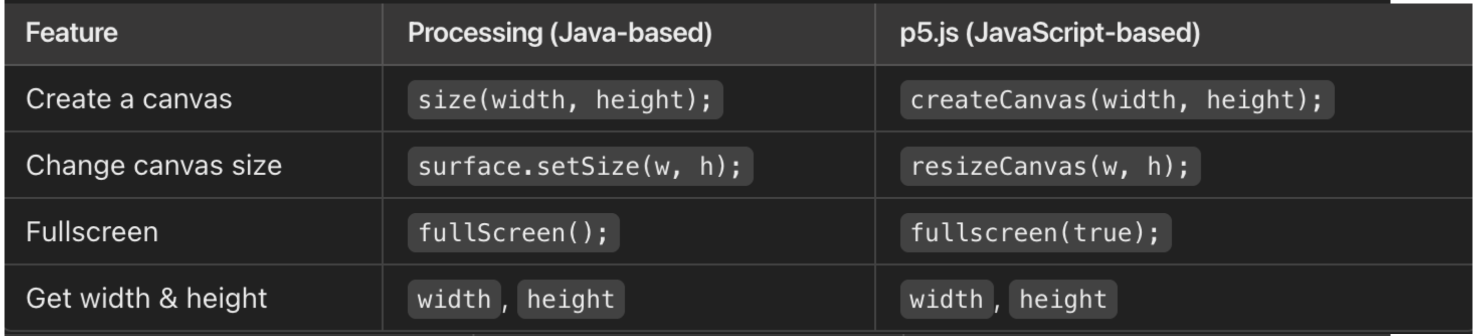


```
let x = 0;

function setup() {
  createCanvas(400, 200);
}

function draw() {
  background(220);
  ellipse(x, height / 2, 50, 50);
  x += 2;
  if (x > width) {
    x = 0;
  }
}
```

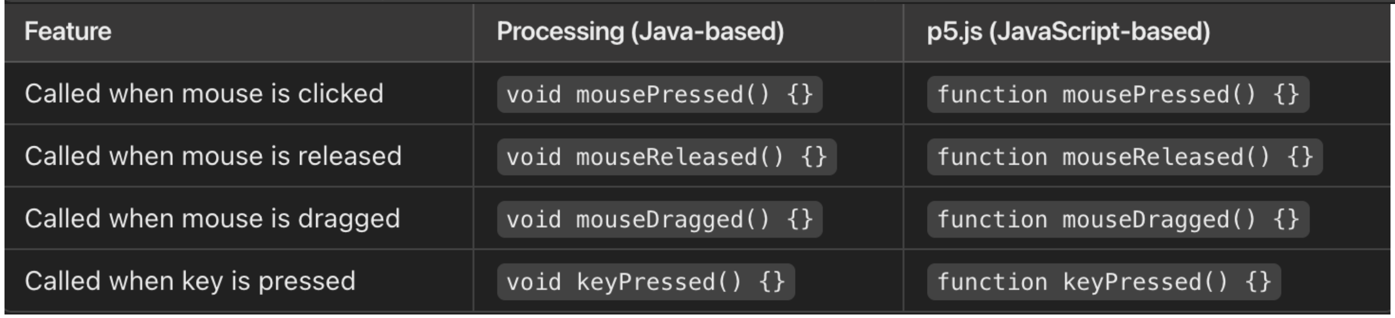
p5.js vs Processing – translating code



Feature	Processing (Java-based)	p5.js (JavaScript-based)
Mouse X position	`mouseX`	`mouseX`
Mouse Y position	`mouseY`	`mouseY`
Is mouse pressed?	`mousePressed` *(boolean variable)*	`mouseIsPressed` *(boolean variable)*
Key pressed	`key`	`key`
Key code	`keyCode`	`keyCode`

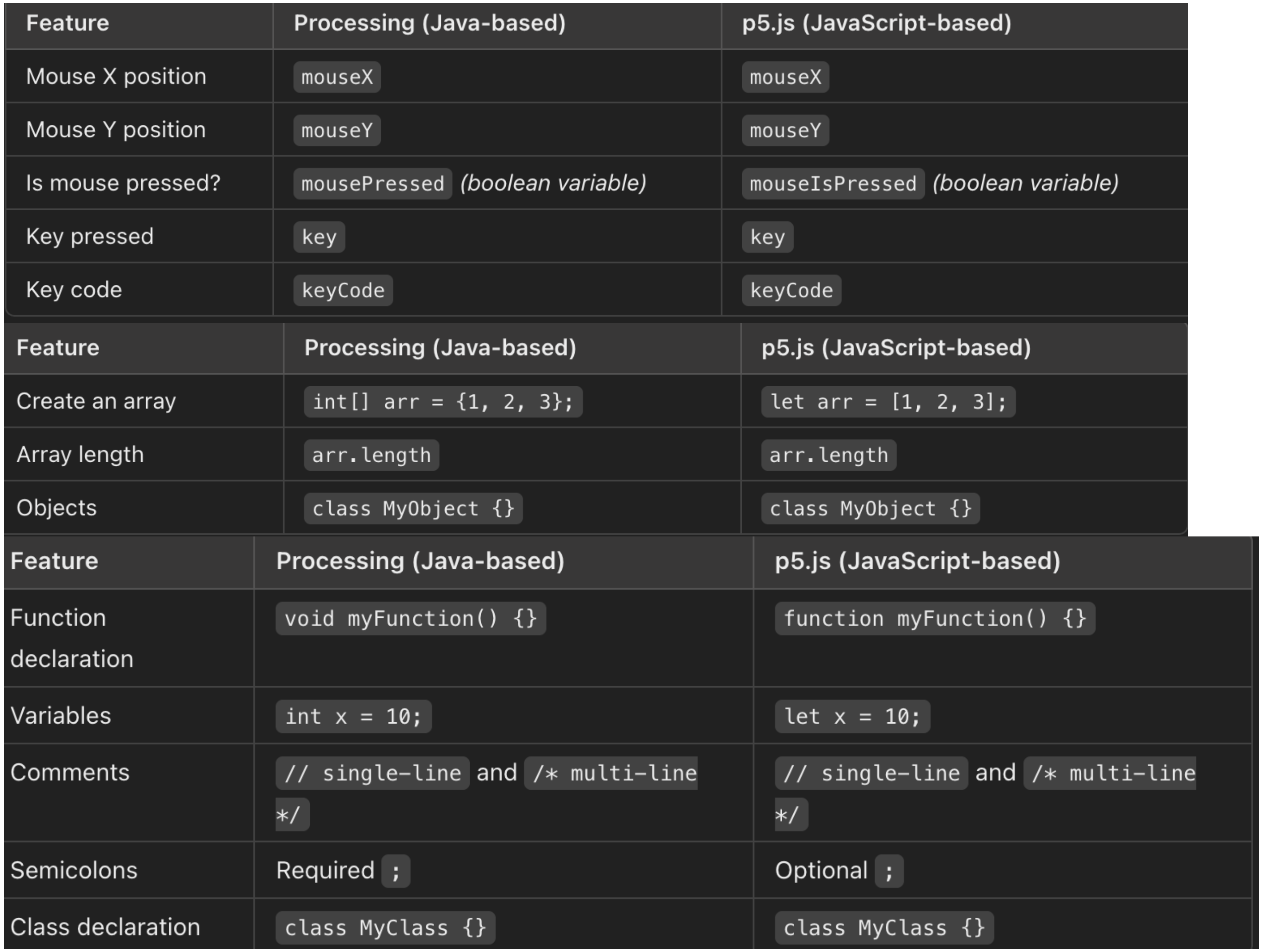


Feature	Processing (Java-based)	p5.js (JavaScript-based)
Create a canvas	`size(width, height);`	`createCanvas(width, height);`
Change canvas size	`surface.setSize(w, h);`	`resizeCanvas(w, h);`
Fullscreen	`fullScreen();`	`fullscreen(true);`
Get width & height	`width`, `height`	`width`, `height`



Feature	Processing (Java-based)	p5.js (JavaScript-based)
Called when mouse is clicked	`void mousePressed() {}`	`function mousePressed() {}`
Called when mouse is released	`void mouseReleased() {}`	`function mouseReleased() {}`
Called when mouse is dragged	`void mouseDragged() {}`	`function mouseDragged() {}`
Called when key is pressed	`void keyPressed() {}`	`function keyPressed() {}`

p5.js vs Processing – translating code



Feature	Processing (Java-based)	p5.js (JavaScript-based)
Mouse X position	`mouseX`	`mouseX`
Mouse Y position	`mouseY`	`mouseY`
Is mouse pressed?	`mousePressed` *(boolean variable)*	`mouseIsPressed` *(boolean variable)*
Key pressed	`key`	`key`
Key code	`keyCode`	`keyCode`

Feature	Processing (Java-based)	p5.js (JavaScript-based)
Create an array	`int[] arr = {1, 2, 3};`	`let arr = [1, 2, 3];`
Array length	`arr.length`	`arr.length`
Objects	`class MyObject {}`	`class MyObject {}`

Feature	Processing (Java-based)	p5.js (JavaScript-based)
Function declaration	`void myFunction() {}`	`function myFunction() {}`
Variables	`int x = 10;`	`let x = 10;`
Comments	`// single-line` and `/* multi-line */`	`// single-line` and `/* multi-line */`
Semicolons	Required `;`	Optional `;`
Class declaration	`class MyClass {}`	`class MyClass {}`

P.3.2.3 Font outline from agents

How long is a letter recognizable as such? In this example, the outlines of a letter serve as the source shape. Each individual nodal point moves like a dumb agent. Over time, the letter becomes illegible and is transformed into something new.

Check this out!

p5.js – web integration

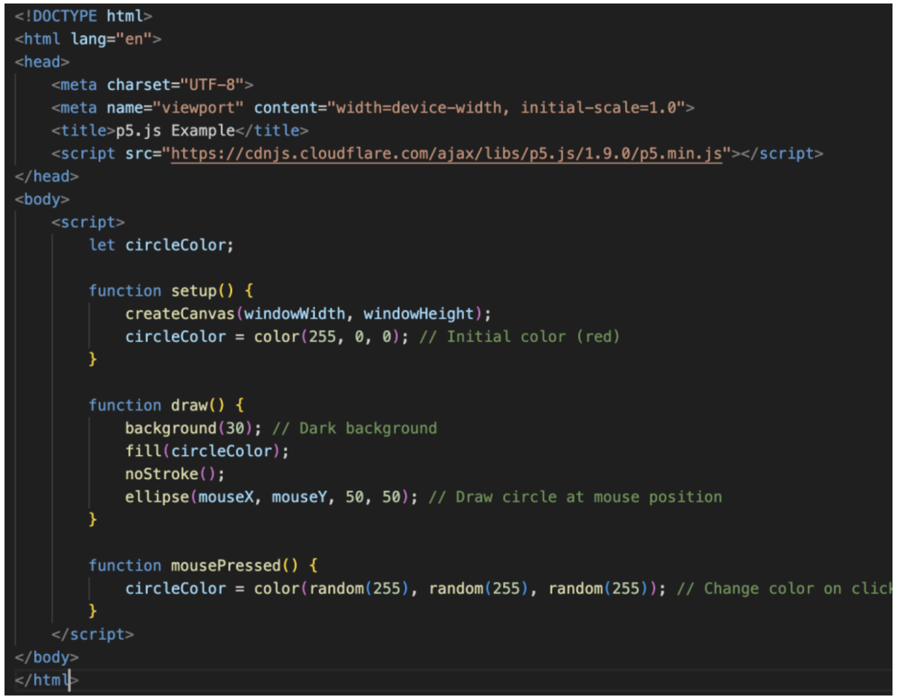
Integrating **p5.js** with **HTML** allows for dynamic and interactive visual experiences on the web.

Since p5.js is a JavaScript library designed for creative coding, it can be embedded directly into an HTML file using a <script> tag or linked via a separate JavaScript file.

This enables interaction between p5.js sketches and standard HTML elements like buttons, sliders, and forms.

Additionally, p5.js can be combined with **CSS** to style elements, ensuring that visuals blend smoothly with web layouts.

Example: Ball that follows mouse
(just save this code as HTML with a IDE like VSCODE e.g. and open the document with a browseer.)



```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>p5.js Example</title>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/p5.js/1.9.0/p5.min.js"></script>
</head>
<body>
    <script>
        let circleColor;

        function setup() {
            createCanvas(windowWidth, windowHeight);
            circleColor = color(255, 0, 0); // Initial color (red)
        }

        function draw() {
            background(30); // Dark background
            fill(circleColor);
            noStroke();
            ellipse(mouseX, mouseY, 50, 50); // Draw circle at mouse position
        }

        function mousePressed() {
            circleColor = color(random(255), random(255), random(255)); // Change color on clic
        }
    </script>
</body>
</html>
```

Processing - Images

Processing is a powerful tool for working with images, offering an easy-to-use API for loading, displaying, and manipulating images.

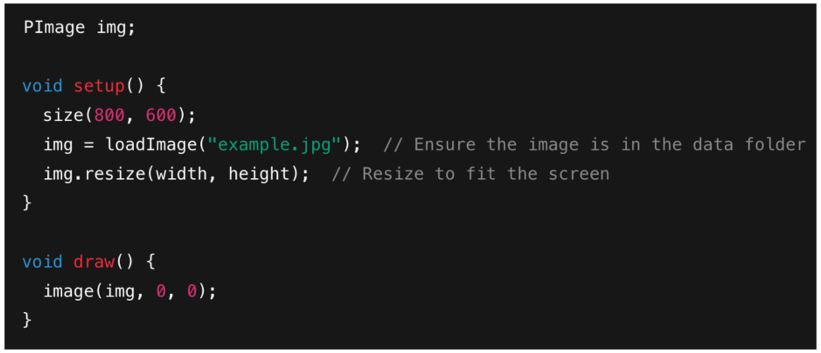
The **PImage** class in Processing provides various functions to process and modify images at the pixel level, enabling artists, designers, and creative coders to experiment with visual effects.

Processing – Images

Loading and Displaying Images
To work with images, they must first be loaded using the loadImage() function and displayed on the screen using image().

The resize() function ensures that the image adapts to different window sizes.

Move "aalto.jpg” (GitHub) to your Processing sketch’s folder.
In Processing, go to
Sketch > Show Sketch Folder



```
PImage img;

void setup() {
  size(800, 600);
  img = loadImage("example.jpg");  // Ensure the image is in the data folder
  img.resize(width, height);  // Resize to fit the screen
}

void draw() {
  image(img, 0, 0);
}
```

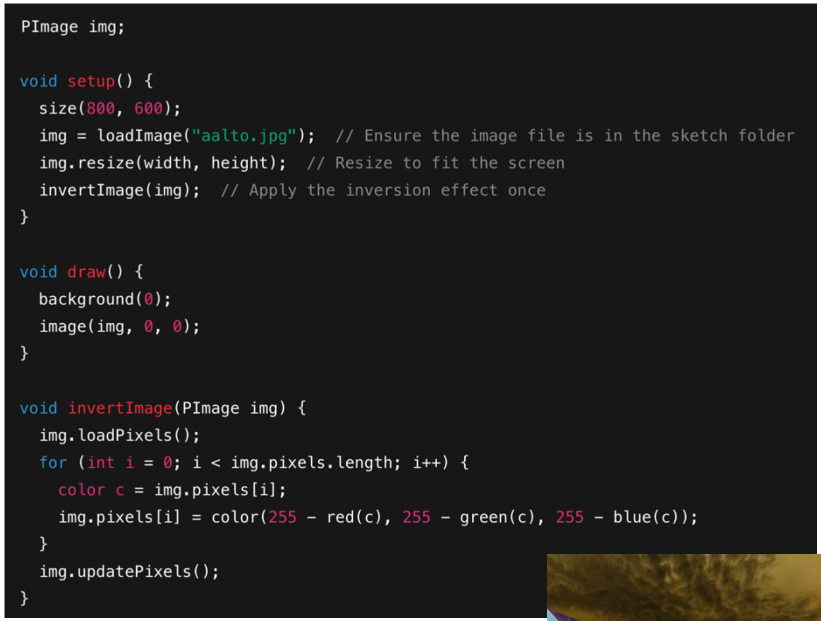
Processing - Images

One of Processing's most powerful features is direct pixel manipulation.

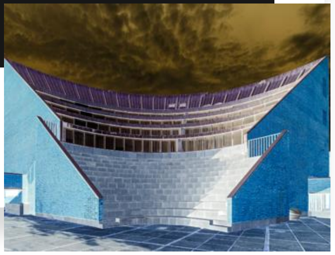
By loading the image’s pixels into an array with loadPixels(), each pixel can be modified, allowing effects like colour changes, filters, or distortions.

For example, inverting an image’s colours.

In this example, the function, takes each pixel, retrieves its red, green, and blue values, subtracts them from 255, and reassigns the new colour.



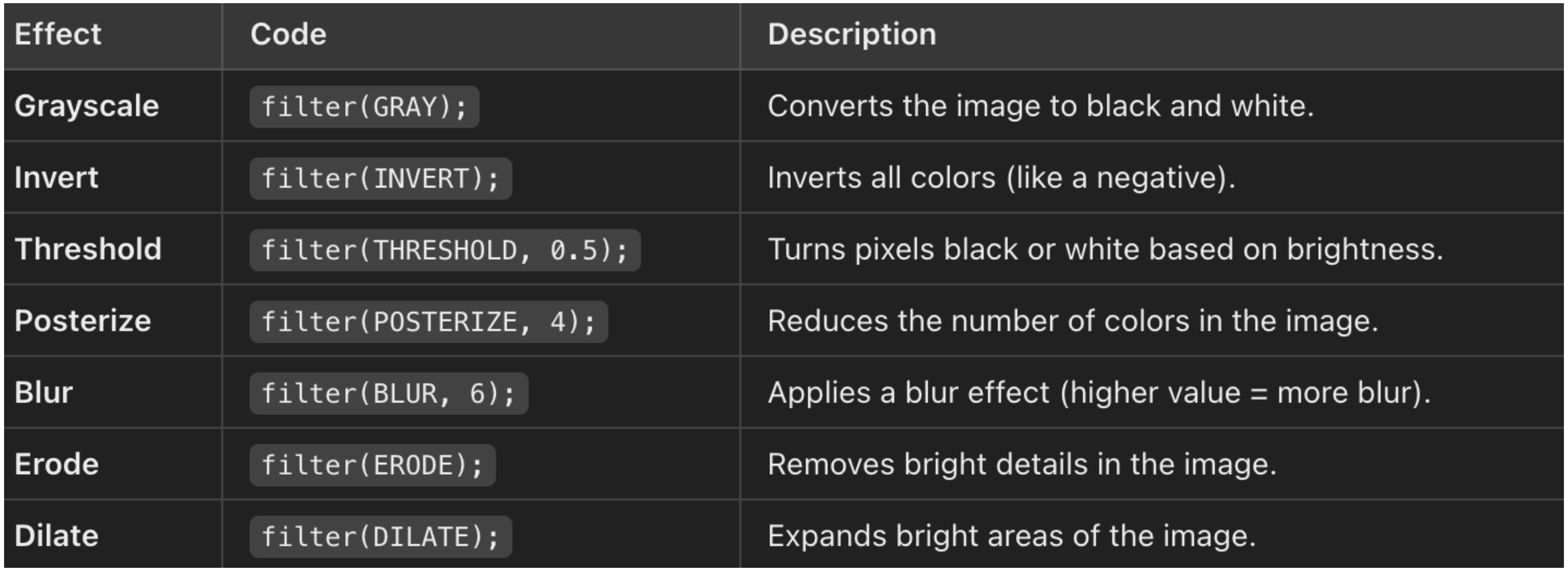
```
PImage img;

void setup() {
  size(800, 600);
  img = loadImage("aalto.jpg");  // Ensure the image file is in the sketch folder
  img.resize(width, height);  // Resize to fit the screen
  invertImage(img);  // Apply the inversion effect once
}

void draw() {
  background(0);
  image(img, 0, 0);
}

void invertImage(PImage img) {
  img.loadPixels();
  for (int i = 0; i < img.pixels.length; i++) {
    color c = img.pixels[i];
    img.pixels[i] = color(255 - red(c), 255 - green(c), 255 - blue(c));
  }
  img.updatePixels();
}
```



Processing – Images

Please check the sample code on github: https://github.com/ptiagomp/aalto-programming-visual-artists-24-25/tree/main/Session-07_17032025/example_imagefilters

Common effects below!



Effect	Code	Description
Grayscale	`filter(GRAY);`	Converts the image to black and white.
Invert	`filter(INVERT);`	Inverts all colors (like a negative).
Threshold	`filter(THRESHOLD, 0.5);`	Turns pixels black or white based on brightness.
Posterize	`filter(POSTERIZE, 4);`	Reduces the number of colors in the image.
Blur	`filter(BLUR, 6);`	Applies a blur effect (higher value = more blur).
Erode	`filter(ERODE);`	Removes bright details in the image.
Dilate	`filter(DILATE);`	Expands bright areas of the image.

Processing - Video

Processing provides a simple and powerful way to handle videos using the processing.video library. This enables users to play, manipulate, and interact with video content in real-time.

In Processing, the Movie object works by continuously sending frames. However, these frames aren't displayed automatically—you need to read and draw them manually in draw().

Just like images (PImage), video frames can be manipulated pixel-by-pixel. Since video is essentially a sequence of images, we can modify each frame dynamically.

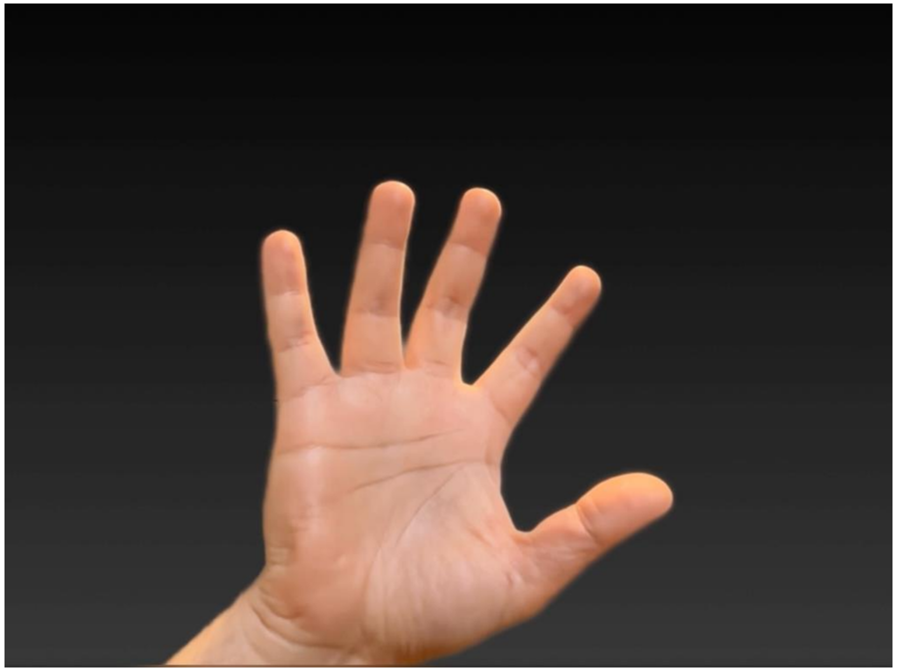
```
import processing.video.*;
```

```
Movie video;

void setup() {
  size(800, 600);
  video = new Movie(this, "test.mp4");  // Load the video file
  video.loop();  // Start looping the video
}
```

```
void draw() {
  background(0);
  if (video.available()) {  // Check if a new frame is ready
    video.read();  // Read and update the frame
  }
  image(video, 0, 0);  // Display the video
}
```

```
void invertVideo(PImage img) {
  img.loadPixels();
  for (int i = 0; i < img.pixels.length; i++) {
    color c = img.pixels[i];
    img.pixels[i] = color(255 - red(c), 255 - green(c), 255 - blue(c));
  }
  img.updatePixels();
}
```

Processing – Video

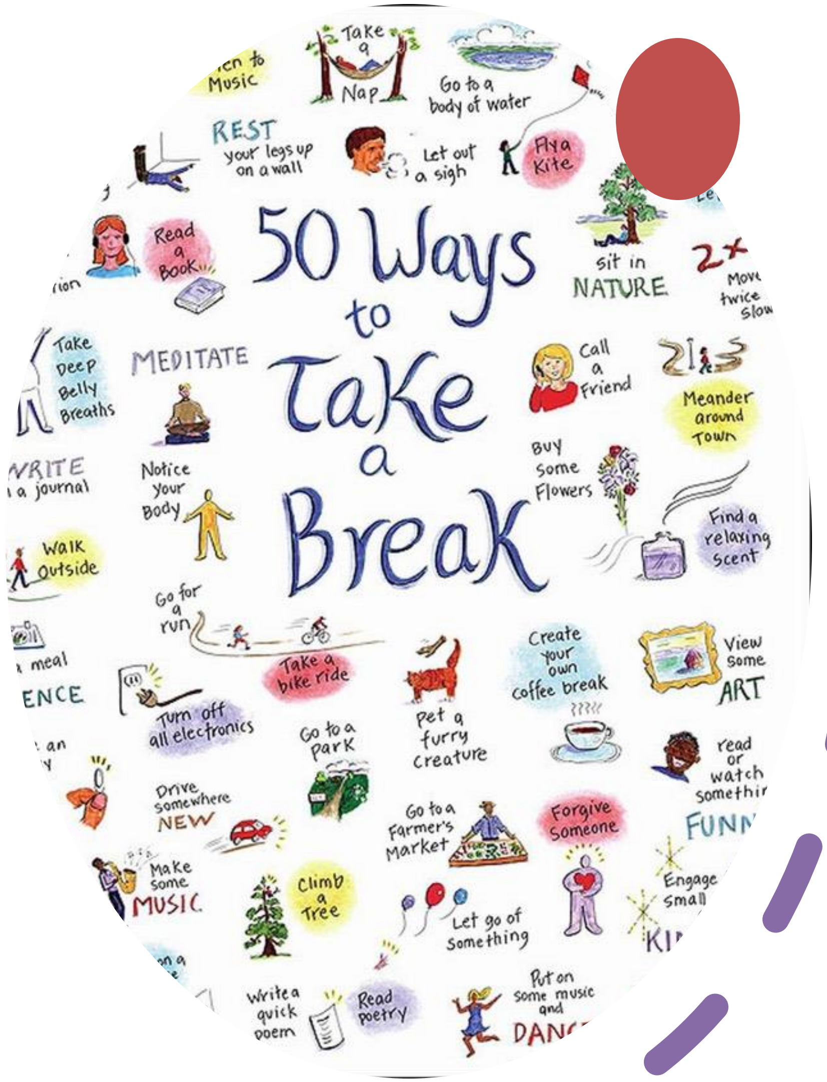


Please check the sample code on github: https://github.com/ptiagomp/aalto-programming-visual-artists-24-25/tree/main/Session-07_17032025/example_videofilters

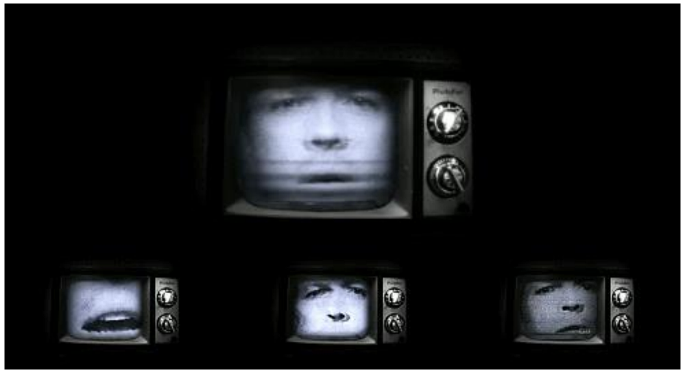
Break

15 min.!

Please don't be late!



Hands-On Exercise!



Objectives:

- Change the codes provided.
- Search for more effects and code them!

Get everything from here: https://github.com/ptiagomp/aalto-programming-visual-artists-24-25/tree/main/Session-07_17032025

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

Discussion & Q&A

Share your feedback!
Am I going too fast or too slow? Is this too easy or too hard?

Tomorrow's topics:

- Webcam usage
- Audio Usage

Don't forget the assignments!