Programming for Visual Artists

2024/2025
Department of Art and Media

Aalto University
School of Arts, Design
and Architecture

Welcome

- Who are we
- Boilerplate
- Tools

Some background on Creative Computation

- Algorithmic Thinking
- Computational Creativity
- Processing Basics

Setup

- Config
  - Processing
  - OpenProcessing
  - GitHub
  - VS Code

First Sketch – "Hello World!"

BREAK (10:30–10:45)

Hands-on Exercise with template

- Change properties, colours, background
- Try out!

Q&A

- Expectations Assessment
- For tomorrow…

…For today

Who are we?!

https://padlet.com/ptiagomp/programmingvisualartists_aalto_2425

https://wordwall.net/resource/86919774

```
void setup() {
    size(600, 400);
    background(255);
}

void draw() {
    // Add your creative code here!
}
```

Boilerplate

In coding, a **boilerplate** refers to a **standardized starting point** that includes pre-written code or structure to help set up a project quickly.
It provides a foundation so that developers (or in this case, visual artists using code) don't have to start from scratch every time.

In our case, it will be also setting "the rules-of-the-game"!

Syllabus

SPECS:

36 hours

Mondays & Tuesdays
9:15–12:00

24.02.2025 to 01.04.2025

Final Goals:

- Learn the basics of computer programming to create screen-based visuals using Java and the Processing library.
- Students complete a creative coding project.

Week 1

Day 1: Monday, 24.02 - Introduction & Setup

- This course's "boilerplate": evaluation, schedule, and projects.
- Theory: Computational creativity, key figures, algorithmic thinking, and Processing basics.
- Practical: Install Processing, OpenProcessing account, GitHub account, Visual Studio Code connection, first sketch, setup() and draw() functions.
- Exercise: Modify a template sketch.

Day 2: Tuesday, 25.02 - Drawing & Interactivity

- Theory: Cartesian coordinate system, color theory, interactivity.
- Practical: Shapes, mouseX, mouseY, dynamic interactions.
- Exercise: Interactive artwork that responds to mouse movements.

Week 2

Day 3: Monday, 03.03 - Animation & Motion

- Theory: Time-based vs. frame-based animation, state variables.
- Practical: Variables, if/else, animated objects.
- Exercise: Bouncing ball animation.

Day 4: Tuesday, 04.03 - Loops & Generative Patterns

- Theory: Algorithmic patterns, loops in generative art.
- Practical: FOR loops, nested loops, dynamic grids.
- Exercise: Create a grid-based generative pattern.

Week 3

Day 5: Monday, 10.03 - Functions for Modular Code

- Theory: Modularity, reusable code components.
- Practical: Defining functions, parameters, return values.
- Exercise: Function-driven generative artwork.

Day 6: Tuesday, 11.03 - Objects & Arrays in Creative Coding

- Theory: Object-Oriented Programming (OOP) in art.
- Practical: Creating objects and arrays, particle systems.
- Exercise: Animation with multiple interacting objects.

Week 4

Day 7: Monday, 17.03 - Transition to p5.js

- Theory: Differences between Processing and p5.js.
- Practical: Setting up p5.js, translating Processing sketches.
- Exercise: Convert a Processing sketch to p5.js.

Day 8: Tuesday, 18.03 - Web Interactivity & Data Sources

- Theory: Event-driven programming, APIs in creative coding.
- Practical: mousePressed(), keyPressed(), fetching external data.
- Exercise: Web-based generative artwork using real-time inputs.

Final Weeks: Project Development & Showcase

Session 9: Monday, 24.03 - Project Development

- Students propose and develop a final project in either Processing or p5.js.
- One-on-one feedback sessions.

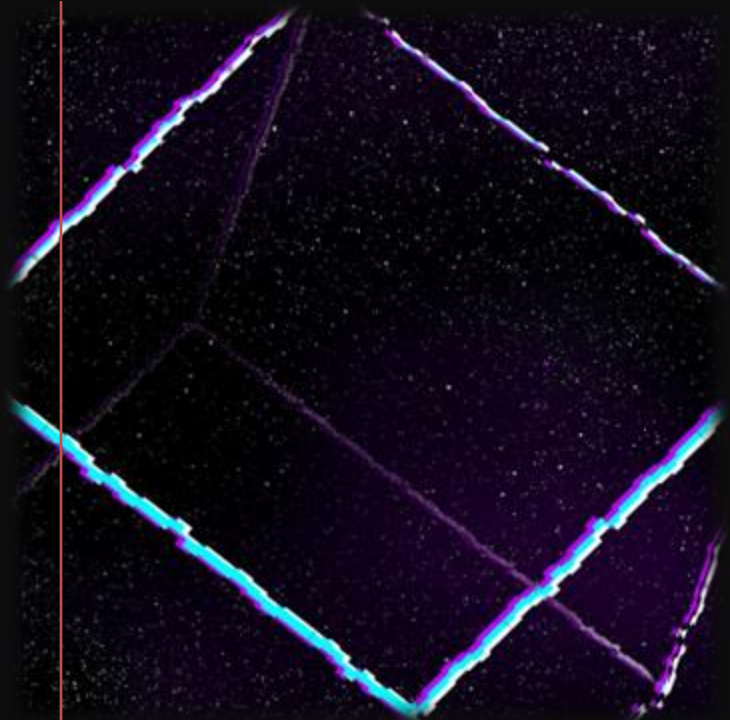### Session 10: Tuesday, 25.03 - Project Development

- Continued project work with guidance.
- One-on-one feedback sessions.

Session 11: Monday, 31.03 - Final Presentations & Critique

- Presentation of final projects.
- Peer and lecturer feedback.

Session 12: Tuesday, 01.04 - Final Presentations & Critique

- Presentation of final projects.
- Peer and lecturer feedback.
- Final reflections on the course's impact on creative practice.

Grading & Attendance

//Grading:

- 4 assignments throughout the 12 sessions, each graded on a scale of 1–5 (60% of the final grade).
- Final Project, graded on a scale of 1–5 (40% of the final grade).
- Each assignment will have detailed evaluation criteria provided in advance.

//Attendance Policy:

- Attendance is mandatory for at least 80% of the sessions.
- The first session is mandatory. Failure to attend the first session will result in automatic unenrollment to accommodate students on the waiting list.

1 - Interactive Sketch - Mouse Movement-based Color and Shape

Opens: Tuesday, 25 February 2025, 12:00 PM
Due: Sunday, 2 March 2025, 11:59 PM

Topics Covered:

Cartesian coordinates, color theory, interactivity, mouse input handling.

Task:

Create an interactive sketch using Processing that dynamically changes colors and shapes based on mouse movement. The program should respond to the in real-time. Experiment with different shapes, colors, and interactions to create an engaging and responsive artwork. You can also use OpenProcessing for

Assessment Criteria:

- Interactivity (30%) – The sketch must respond dynamically to mouse movement (mouseX, mouseY).
- Visual Complexity & Creativity (25%) – The artwork should be visually engaging with well-designed color choices and shape variations.
- Code Structure & Readability (20%) – Code should be well-structured, with clear variable names and comments explaining key parts.
- Implementation of Processing Functions (15%) – Proper use of Processing's setup(), draw(), and event-driven functions (mouseX, mouseY).
- Completion & Functionality (10%) – The program must run without errors and meet the assignment requirements.

Projects / Assigments

2 - Dynamic Circle Grid with Mouse Interaction

Make a submission   Receive a grade   Receive a passing grade

Opens: Tuesday, 4 March 2025, 12:00 PM
Due: Sunday, 9 March 2025, 11:59 PM

Topics Covered:

Loops, generative patterns, grid-based design, mouse interaction.

Task:

Create a dynamic grid of circles in Processing that responds to mouse input. The grid should modify properties such as size, color, spacing, or symmetry based on mouse movement or position. Experiment with different visual patterns to create an interactive composition. You can also use OpenProcessing for this one.

Assessment Criteria:

- Loop Implementation (30%) – Effective use of for loops and nested loops to generate a structured grid.
- Mouse Interaction (25%) – The grid must respond dynamically to mouse movement, altering properties like size, color, or spacing.
- Creativity & Visual Appeal (20%) – The design should be visually interesting, exploring symmetrical or asymmetrical patterns.
- Code Efficiency & Readability (15%) – Well-organized code with clear logic, comments, and meaningful variable names.
- Functionality (10%) – The program must run without errors and meet the assignment objectives.

Projects / Assigments
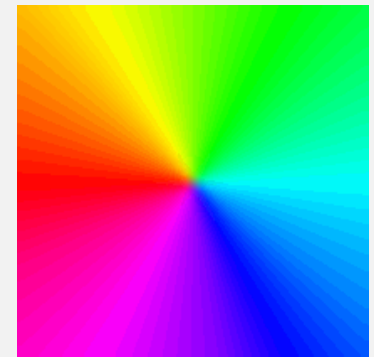
3 - Function-Based Modular Artwork

Make a submission   Receive a grade   Receive a passing grade

Opens: Monday, 10 March 2025, 12:00 PM
Due: Sunday, 16 March 2025, 11:59 PM

Topics Covered:

Functions, modular code design, parameter-driven visual elements.

Task:

Design a generative artwork in Processing that utilizes functions to structure the code modularly. Your program should define at least two functions that generate visual elements with parameters controlling aspects such as size, color, or position. The goal is to make reusable functions that enhance flexibility in design. You can also use OpenProcessing for this one.

Assessment Criteria:

- Function Design & Use (30%) – At least two well-implemented functions that control visual elements.
- Code Modularity & Reusability (25%) – Code should be structured using functions to avoid redundancy.
- Creativity & Artistic Expression (20%) – The generative artwork should explore visually engaging patterns and structures.
- Code Readability & Documentation (15%) – Well-commented and structured code with meaningful function names and parameters.
- Correct Execution (10%) – The program must run as intended, producing a coherent generative artwork.

Projects / Assigments
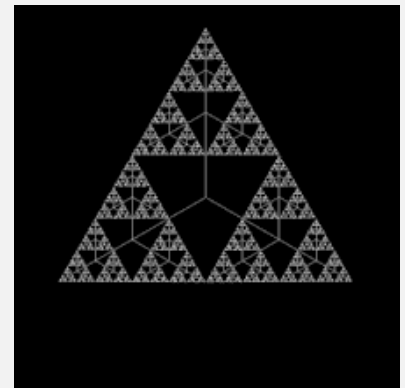
4 - Object-Oriented Animation with Particle Systems

Opens: Tuesday, 11 March 2025, 12:00 PM
Due: Sunday, 23 March 2025, 11:59 PM

Topics Covered:

Object-Oriented Programming (OOP), particle systems, arrays.

Task:

Implement an animated particle system using objects and arrays in Processing. The program should create a set of particles that move, interact, or evolve over time. You can explore behaviors such as attraction, repulsion, fading, or random motion. You can also use OpenProcessing for this one.

Assessment Criteria:

- Object-Oriented Implementation (30%) – Use of classes and objects to manage particles effectively.
- Animation & Motion (25%) – Smooth and visually appealing movement of particles over time.
- Code Organization & Reusability (20%) – Clear class structure, use of methods for behavior updates.
- Complexity & Creativity (15%) – The particle system should explore interesting interactions or behaviors.
- Correct Execution (10%) – The program must function as intended without runtime errors.

Projects / Assigments

Final Assignment: Interactive Generative Project (Processing or p5.js)

Make a submission   Receive a grade   Receive a passing grade

Opens: Monday, 17 March 2025, 12:00 PM
Due: Sunday, 30 March 2025, 11:59 PM

Topics Covered:

All discussed during the course.

Task:

Develop an original interactive generative project in either Processing or p5.js that integrates the key concepts learned throughout the course. The project should demonstrate creativity, technical skill, and a structured approach to coding. The final piece can be an interactive artwork, a generative animation, or a simple creative tool, as long as it meaningfully incorporates user interaction and dynamic elements.

Assessment Criteria:

- Creative Concept & Execution (30%)
  - The project should be visually engaging and creatively developed.
  - There should be a clear artistic or interactive goal.
- Technical Implementation (25%)
  - Effective use of functions, objects, and interactivity.
  - Clean, modular, and efficient code.
- Interactivity & User Experience (20%)
  - Smooth interaction and responsive feedback.
  - Thoughtful design choices that enhance user engagement.
- Code Quality & Documentation (15%)
  - Well-structured code with meaningful variable names and comments.
  - Clear explanation of the project's structure and features in the reflection.
- Presentation & Reflection (10%)
  - Clear explanation of the project, its challenges, and creative choices.
  - Engaging and structured final presentation.

Projects / Assigments

Tools

Visual Studio Code

GitHub

OpenProcess

Processing
IDE / CORE / JAVA

Some background on Creative Computation

- In 1834, André-Marie Ampère introduced the term *cybernetics*, referring to it as "**the future science of government**." Derived from the French *cybernetique* ("**art of governing**") and the Greek *kybernetes* ("**navigator**" or "**steersman**").[1]

- Norbert Wiener popularized *cybernetics* in 1949.[2]

- In 1968, Robert Mueller applied **cybernetics** to *mental entropy*[3], and in 1971, Bruce Lindsay traced its origins back to Ampère's 1834 work.[4]

- Richard Coren proposed a *cybernetics-based evolution theory* in 1998 [5].

[1] Tsien, Hsue S. (1954). Engineering Cybernetics (pg. vii). McGraw-Hill.
[2] Wiener, Norber. (1948). Cybernetics: or Control and Communication in the Animal and the Machine (pgs. 11-12). Cambridge, Mass.: The MIT Press.
[3] Mueller, Robert E. (1968). The Science of Art: The Cybernetics of Creative Communication,(pg. 67). Rapp & Whiting.
[4] Lindsay, Bruce. (1971). "The Larger Cybernetics", Zygon, 6(2):126-34.
[5] Coren, Richard L. (1998). Evolutionary Trajectory: the Growth of Information in the History and Future of Earth (ch. 7: Entropy, pgs 111-26). CRC Press.

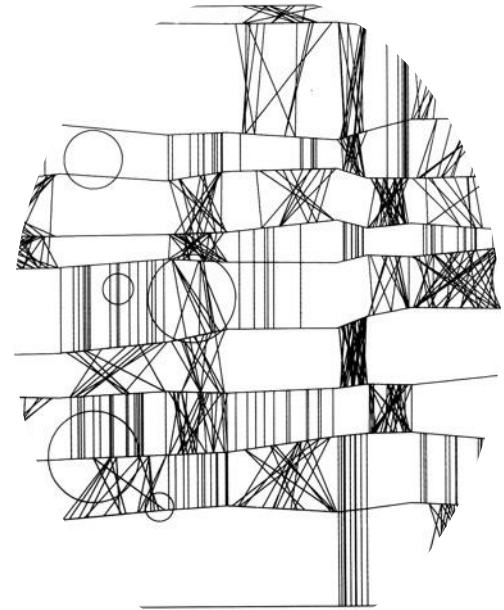# Some background on Creative Computation

Early Use of Computers in Art

In the late 1950s and early 1960s, early experiments with computer-generated imagery emerged in academic and research settings, led by artists with backgrounds in mathematics, engineering, or architecture.

Frieder Nake (b. 1938): A mathematician and artist, Nake created some of the earliest algorithmic drawings using a plotter printer and mathematical rules.

Georg Nees (1926–2016): One of the first to publicly exhibit computer-generated images.[6]

Michael Noll (b. 1939): Conducted research at Bell Labs and explored computational aesthetics.

KUGEL UND ELLIPSOID IM UNTERRAUM   1-APR-86 13:06:01

6 Nees, Georg (1969). Generative Computergraphik.
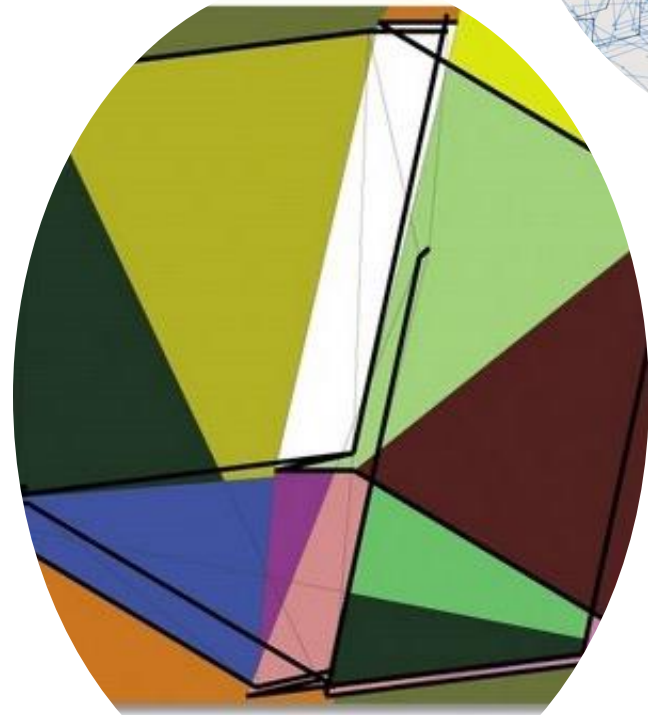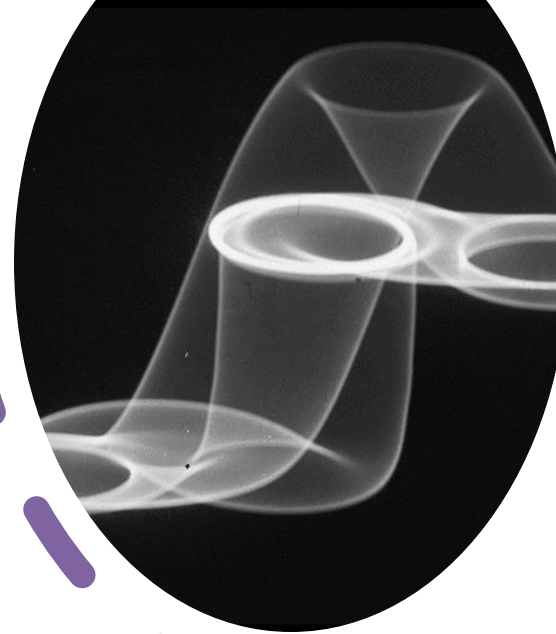
Some background on Creative Computation

The Emergence of Generative Art as an Aesthetic Movement

Algorithmic art represents a conceptual shift in artistic practice, where artists formulate rules instead of directly crafting visual forms. Like conceptual art, this approach emphasizes the importance of ideas and structure alongside the final visual outcome.

Vera Molnár (1924–2023): Used early computers to explore geometric abstraction and randomness.

Manfred Mohr (b. 1938): Developed visual works based on logical structures and hypercubes.

Herbert W. Franke (1927–2022): Combined scientific knowledge and artistic practice, exploring computational aesthetics.[7]

[7] Franke, H. W. (1971). *Computer Graphics – Computer Art*. Phaidon Press.

Some background on Creative Computation

The Role of Research Institutions and Technology

Many algorithmic artists worked at universities and laboratories, where they had access to mainframe computers and early plotting devices.

Bell Labs (USA): A hub for early computer art experiments, hosting artists like Noll and Kenneth Knowlton.

Siemens Research (Germany): Supported early exhibitions of generative art.[8]

Cybernetic Art in Britain: Gordon Pask and Roy Ascott explored interactive and rule-based art forms.

One of the most significant technological advances of this period was the **plotter printer**, which allowed for precise, mechanical drawing of algorithmic compositions. These machines, typically used for engineering and scientific purposes, became key tools for generative artists.

[8] Galanter, P. (2003). *What is Generative Art? Complexity Theory as a Context for Art Theory*. GA 2003.

Some background on Creative Computation

**Exhibitions and Recognition of Algorithmic Art**

The 1960s and 1970s saw the first major exhibitions of computer-generated art, helping to establish the field within the broader art world.

1965: First Algorithmic Art Exhibition
(*Generative Computergrafik*, Stuttgart)

1968: Cybernetic Serendipity (ICA, London)[9] - Featured early computer-generated works and interactive installations.

1970s: Ars Electronica (Austria) - Began as a platform for exploring digital and media art, later becoming a major festival.

These exhibitions helped to legitimize algorithmic art as a serious artistic practice, moving it beyond the confines of technical research.
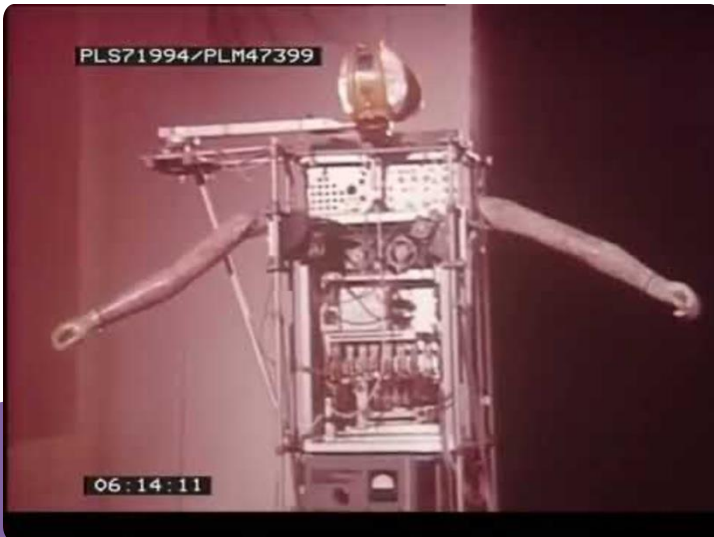
9 Reichardt, J. (1968). Cybernetic Serendipity: The Computer and the Arts. Studio International.

Some background on Creative Computation

Impact and Legacy of Algorithmic Art

By the late 1970s, algorithmic art had established itself as a distinct field within digital art. The ideas developed in this era—rule-based design, randomness, and computer-assisted creativity—laid the groundwork for later innovations in generative art, AI-based creativity, and contemporary digital aesthetics.
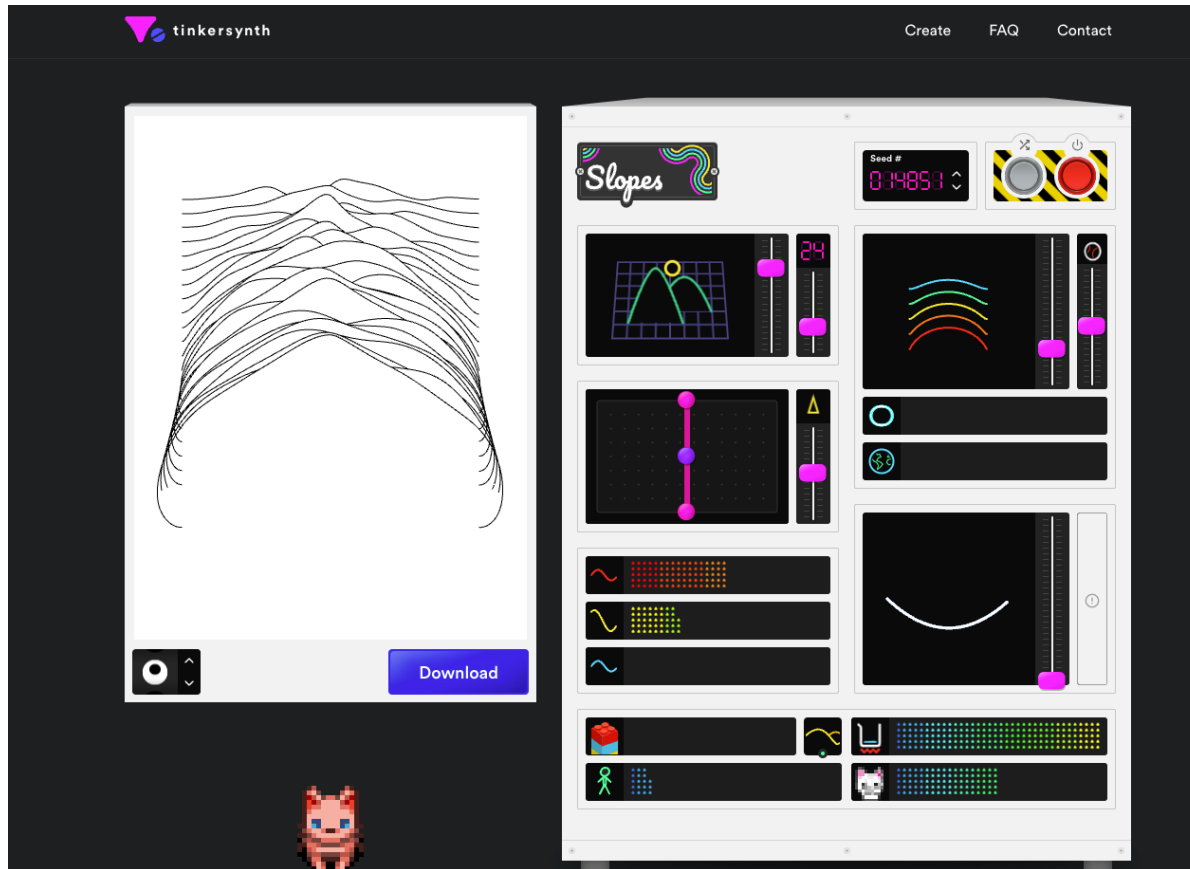
Key Influences on Later Art Movements:

Creative Coding (Processing, p5.js): Algorithmic principles were revived with tools like Processing, co-created by Casey Reas back in 2001.

AI-Generated Art (GANs, DeepDream, DALL·E): The exploration of rules and randomness continues in modern AI-driven creative systems.

NFT and Blockchain Art: The idea of algorithmically generated uniqueness is now prominent in digital art markets.

Grau, O. (2003). Virtual Art: From Illusion to Immersion. MIT Press.

Let's try out something!

https://tinkersynth.com/

Algorithmic Thinking

Best tea I've ever had!

Algorithmic thinking is a **step-by-step problem-solving approach** that breaks down tasks into a sequence of well-defined instructions. It is a fundamental concept in computer science, but it is also applicable in everyday life, creative processes, and decision-making.

Key Elements of Algorithmic Thinking:

- **Decomposition** – Breaking a complex problem into smaller, manageable parts.
- **Pattern Recognition** – Identifying recurring structures or similarities in problems.
- **Abstraction** – Ignoring unnecessary details to focus on relevant concepts.
- **Sequencing & Automation** – Arranging steps in a logical order to achieve a goal.
- **Optimization** – Improving solutions to be more efficient or effective.
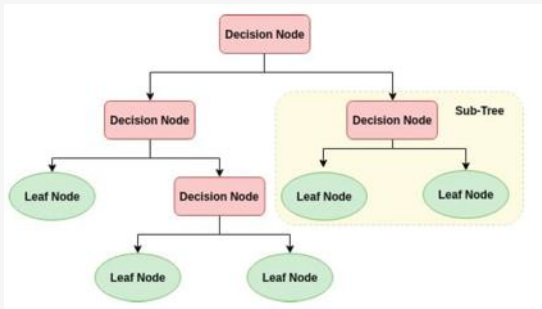
Simple Example:
Making a cup of tea can be described algorithmically:

1. Boil water.
2. Add a tea bag to a cup.
3. Pour hot water into the cup.
4. Let the tea steep for 3–5 minutes.
5. Remove the tea bag and enjoy.

Each step is **precise**, **ordered**, and **repeatable!** Just like an algorithm in programming.

Algorithmic Thinking

$$2 - x^2 = -4 - x^2 + 6x$$
$$2 + 4 = 6x$$
$$6 = 6x$$
$$x = \frac{6}{6} = 1$$

Algorithmic Thinking in Different Fields:

- **Computing**: Writing a program to sort numbers.
- **Mathematics**: Solving equations using step-by-step rules.
- **Art & Design**: Creating generative patterns with rules and randomness.
- **Daily Life**: Following a cooking recipe or assembling furniture.

Parts of an Algorithm

- Input (*constants and variables)

- Process (Steps/Computation)

- Decision-Making (Conditions and Branching)

- Iteration (Loops/Repetition)

- Output

- Termination (End Condition)

Example "Find Max Number from list" Algorithm Steps:

1. Input: A list of numbers.

2. Set the first number as the maximum.

3. Compare each number in the list with the maximum.

4. If a number is greater, update the maximum.

5. Repeat until all numbers are checked.

6. Output the maximum number.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms*. MIT Press.

Programming Languages

Machine Language (Low-Level)

- The **lowest-level programming language** directly understood by the CPU.
- Composed entirely of **binary (0s and 1s)** or **hexadecimal** values.
- Extremely **fast and efficient**, as it does not require translation.
- **Not human-readable** and difficult to debug or modify.
- **CPU-specific**, meaning it differs between processor architectures (e.g., Intel x86 vs. ARM).

```
10110000 01100001
```

This means in Assembly: **MOV AL, 61h**
(*Move hexadecimal value 61 into register AL*).

High-Level Language

- **Abstracted from hardware**, making it easier to read and write.
- Uses **human-friendly syntax** (e.g., Python, C++, Java).
- Requires a **compiler or interpreter** to convert into machine code.
- **Portable across multiple CPU architectures** (e.g., a Python program runs on both Intel and ARM).
- Used for **general application development, AI, web development, and automation**.

```
x = 10
y = 20
sum = x + y
print(sum)    # Output: 30
```

Processing "language"

- Processing is not a programming language itself but rather a **software sketchbook and development environment** built on **Java**.

- It provides a simplified syntax, and a graphics-oriented API designed to make programming more accessible, especially for artists, designers, and beginners.

- While Processing is primarily Java-based, there are also **alternative modes** that allow users to write code in **JavaScript (p5.js)** and **Python (Processing.py)**.
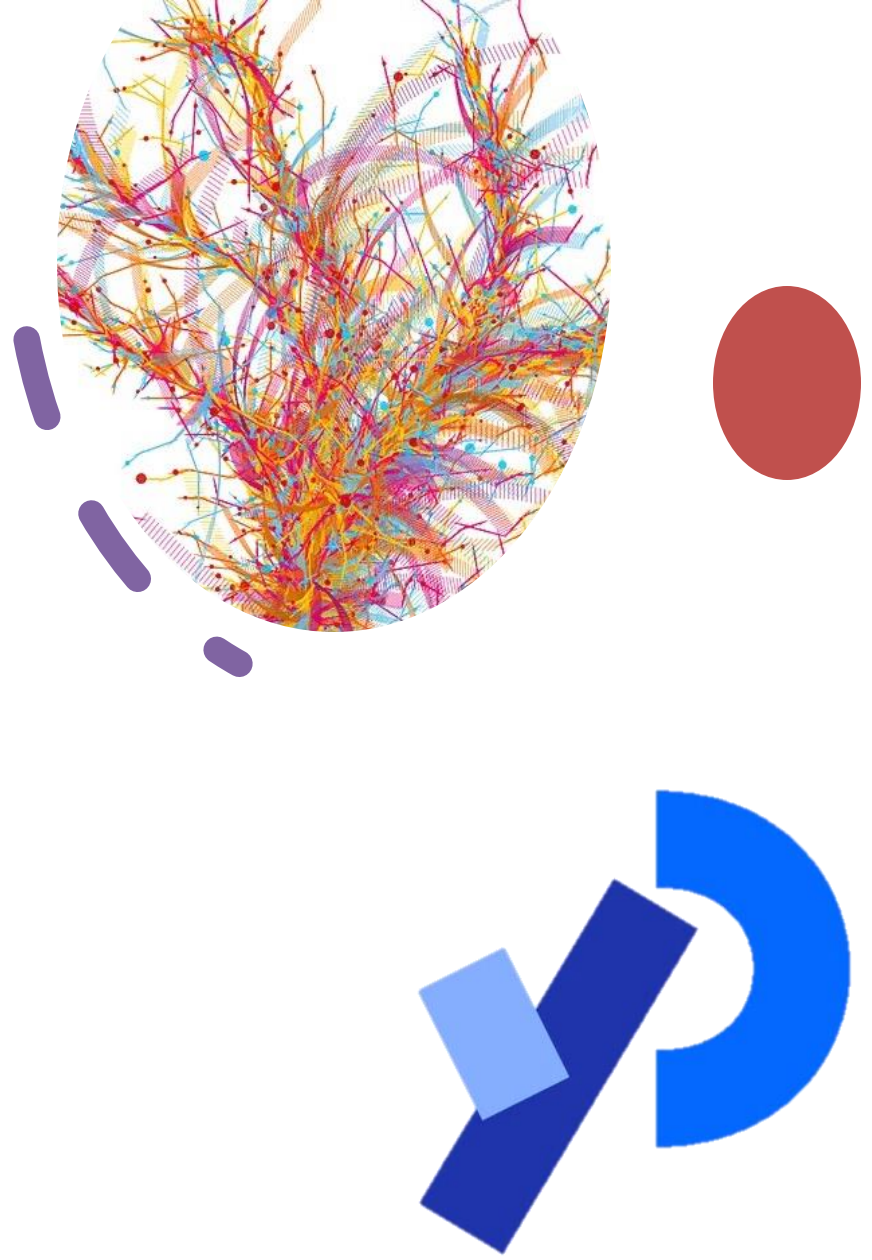
Processing Workflow

Initialization (setup())

- Runs **once** when the program starts.
- Used for setting up the canvas size, initializing variables, loading assets (images, fonts, data), etc.

```
void setup() {
    size(400, 400);   // Set canvas size
    background(255);  // Set background color
}
```

```
void draw() {
    ellipse(mouseX, mouseY, 50, 50);  // Draw circle at mouse position
}
```

Processing Workflow

Continuous Execution (draw())

- Runs **continuously** in a loop after setup(), approximately **60 times per second** (default frame rate).

- Used for animations, real-time interactions, and continuously updating visuals.

```
void mousePressed() {
    background(random(255));  // Change background color on click
}
```

Processing Workflow

Event Functions (User Interactions)

- These functions respond to user input like mouse clicks, key presses, etc.

- **mousePressed()** – Runs when the mouse button is clicked.

Processing Workflow

```
void mousePressed() {
    background(random(255));  // Change background color on click
}
```

```
void keyPressed() {
    println("Key pressed: " + key);
}
```

Event Functions (User Interactions)

- These functions respond to user input like mouse clicks, key presses, etc.
- **mousePressed()** – Runs when the mouse button is clicked.
- **keyPressed()** – Runs when a key is pressed.

Processing Workflow

```
void setup() {
    size(400, 400);
    frameRate(30);  // Limit to 30 FPS
}

void draw() {
    line(random(width), random(height), random(width), random(height));
}

void mousePressed() {
    noLoop();  // Stop the draw loop when clicked
}
```

Control Functions

- **noLoop()** – Stops draw() from running continuously.
- **loop()** – Restarts draw().
- **frameRate(x)** – Sets the number of times draw() runs per second.

```
void drawCircle(float x, float y, float d) {
    ellipse(x, y, d, d);
}


void draw() {
    drawCircle(mouseX, mouseY, 50);
}
```

Processing Workflow

Custom Functions

- You can define your own functions and call them within setup() or draw().

Setup

*plugins

*libraries

```
void setup() {
    size(400, 200);  // Set canvas size
    background(0);    // Set background color to black
    fill(255);        // Set text color to white
    textSize(32);     // Set text size
    textAlign(CENTER, CENTER);  // Center the text
}


void draw() {
    text("Hello, World!", width / 2, height / 2);  // Display text in the center
}
```

First Sketch

size(400, 200); → Sets the window size to **400x200 pixels**.

background(0); → Sets the background to **black**.

fill(255); → Sets the text color to **white**.

textSize(32); → Sets the font size to **32 pixels**.

textAlign(CENTER, CENTER); → Centers the text horizontally and vertically.

text("Hello, World!", width / 2, height / 2); → Displays **"Hello, World!"** in the center of the screen.

```
void setup() {
    size(400, 200);  // Set canvas size
    background(0);    // Set background color to black
    fill(255);        // Set text and ellipse color to white
    textSize(24);     // Set text size
    textAlign(CENTER, CENTER);  // Center the text
}

void draw() {
    background(0);  // Clear the screen each frame

    // Draw an ellipse in the center
    ellipse(width / 2, height / 2, 150, 100);

    // Display "Hello, World!" inside the ellipse
    fill(0);  // Set text color to black for contrast
    text("Hello, World!", width / 2, height / 2);
}
```

Second "First Sketch"

Creates a **400x200** pixel window.

Draws a **white ellipse** in the center.
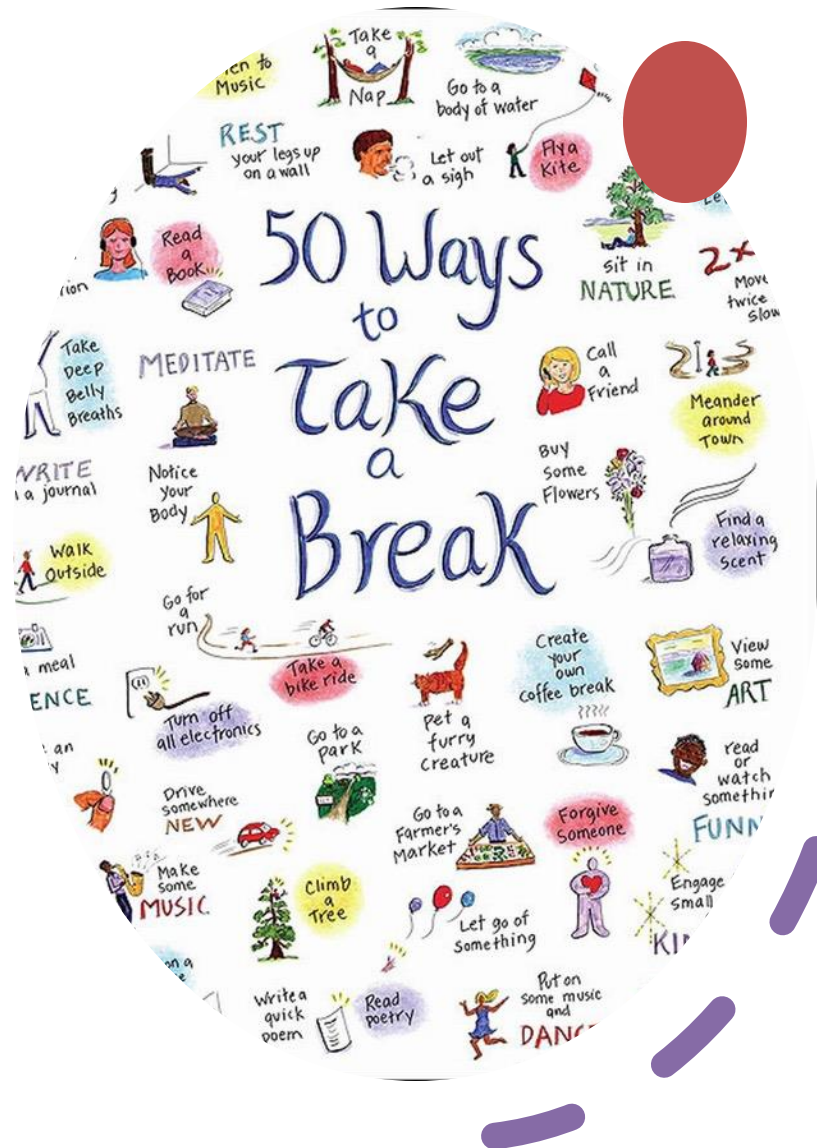
Displays **"Hello, World!"** inside the ellipse.

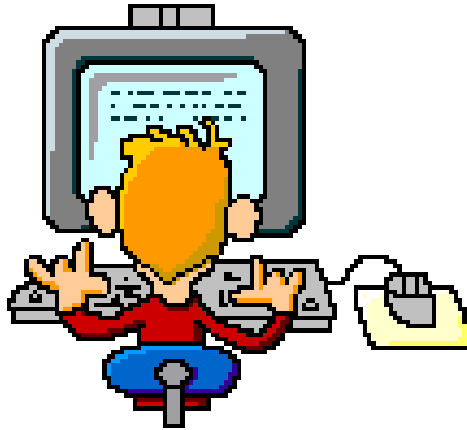The background refreshes in draw(), keeping it clean.

Break

15 min.!

Please don't be late!

Objectives:

- Change shape properties, colours, and **background**.

- Experiment! Try out things!

Get it from here:
https://github.com/ptiagomp/aalto-programming-visual-artists-24-25/tree/main/Session-01_24022025

(if bored, check for the "extra" file!)

Hands-On Exercise!

Discussion & Q&A

Share your feedback!

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

Tomorrow's topic:

Basic drawing & interactivity