Programming for Visual Artists

Aalto University
School of Arts, Design and Architecture

2024/2025
Department of Art and Media

Some things to check

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

Recap

- Control Structures
- Animation

Welcome

- Recap
- Today's Goals

Algortithm Patterns

- Grid-based Patterns
- Sinusoidal Waves Patterns
- Recursive Tree
- Random Walks
- Sinusoidal Animations

Nested Loops

Examples

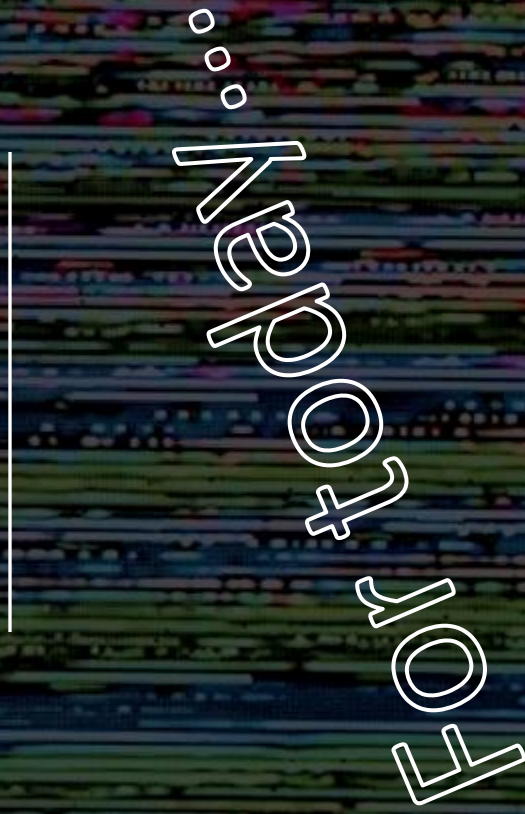
BREAK (10:30–10:45)

Coding tasks

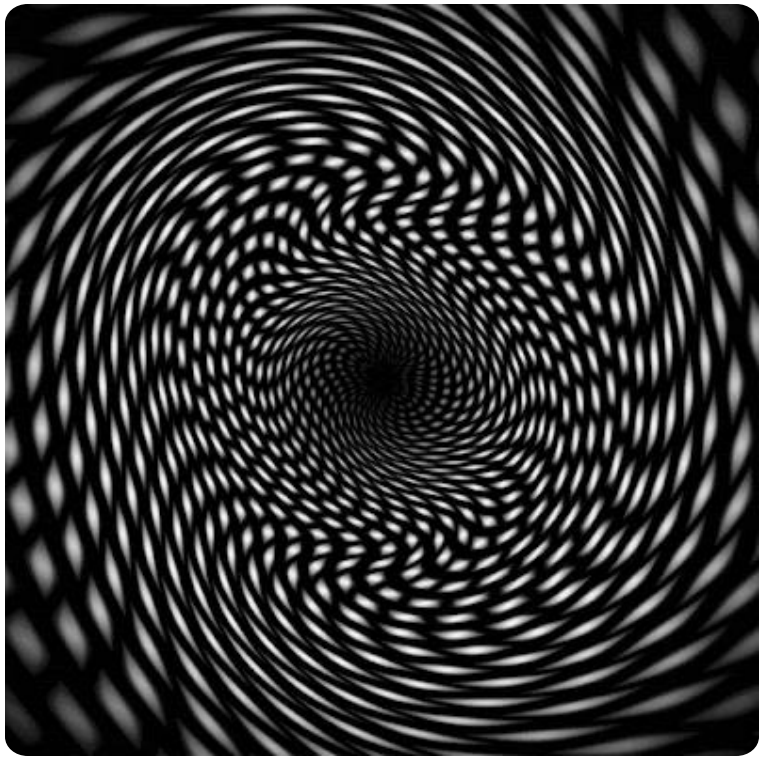
- Generative Grid
- Animated Waves

Q&A

- For tomorrow…



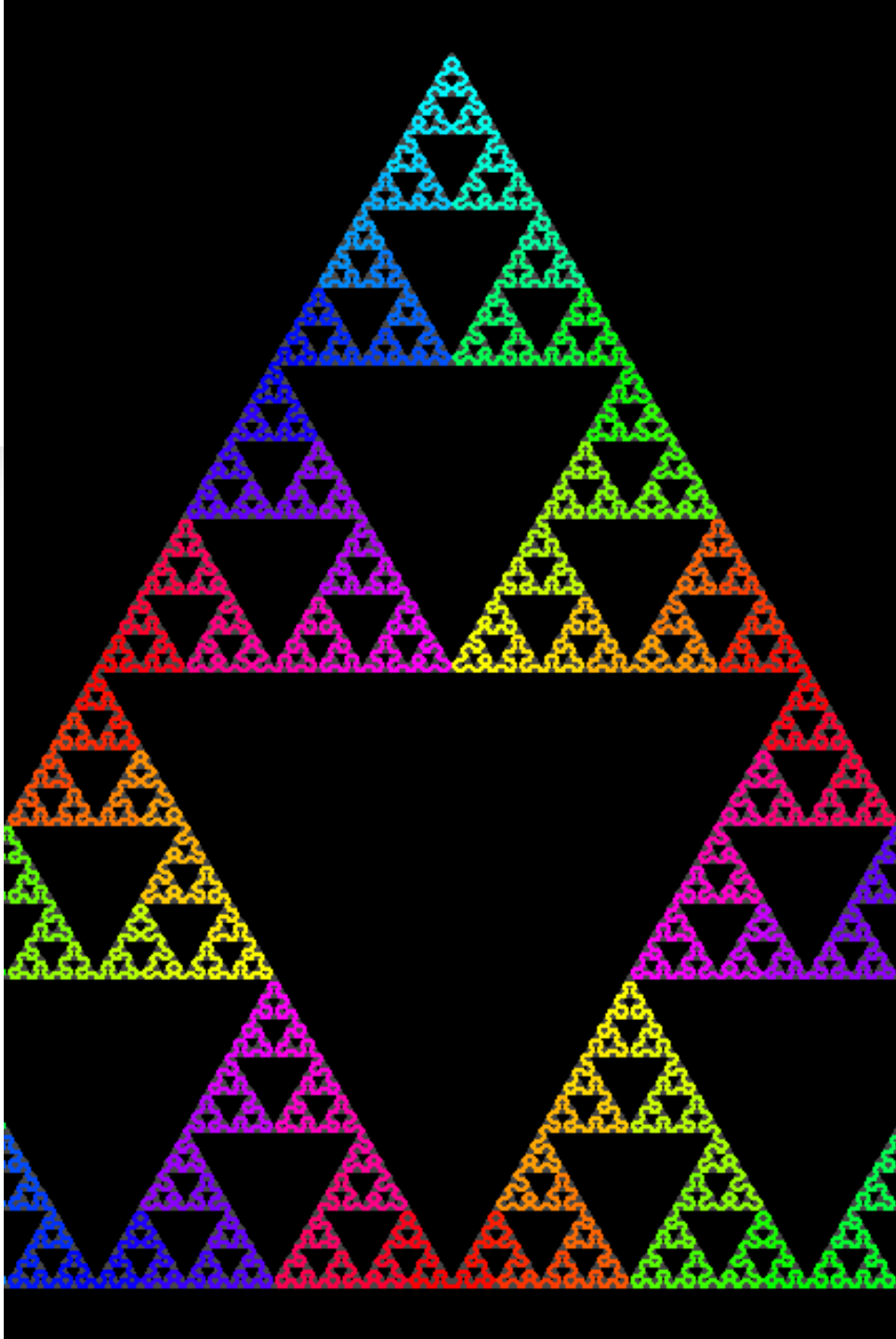
Algorithm Patterns



- An **algorithmic pattern** is a visual composition generated using a set of rules or a mathematical process.
- These patterns are often based on **loops, randomness, transformations, recursion, and mathematical functions** to create dynamic and generative designs.
- They allow for **dynamic, procedural art** that can be modified by adjusting variables, incorporating interactivity, or adding randomness.
- Also, with loops, we have the possibility of **generating art** dynamically.

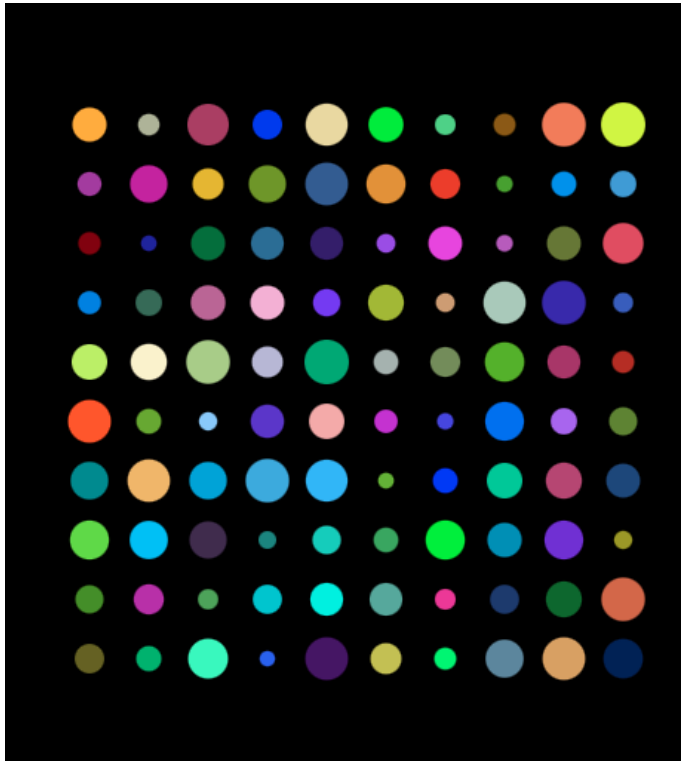
Algorithm Patterns – key features

- **Repetition & Loops** – Using for or while loops to draw elements in a structured way.
- **Randomness & Noise** – Introducing variability with random() or noise().
- **Mathematical Functions** – Using trigonometry (sin(), cos()), linear functions, or fractals.
- **Transformations** – Applying translate(), rotate(), and scale() to manipulate elements.
- **Recursion** – Generating self-repeating patterns (e.g., fractals like Sierpiński Triangle).
- **Color & Transparency** – Varying colors and transparency to add depth.



Algorithm Patterns – examples

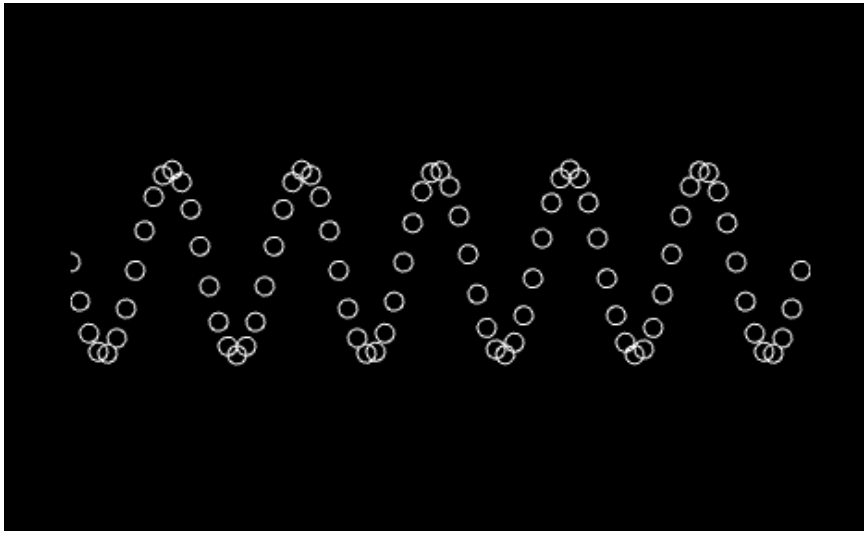
Grid-Based Algorithmic Pattern



```
void setup() {
  size(400, 400);
  background(0);
  noStroke();

  for (int x = 0; x < width; x += 40) {
    for (int y = 0; y < height; y += 40) {
      float size = random(10, 30);
      fill(random(255), random(255), random(255));
      ellipse(x + 20, y + 20, size, size);
    }
  }
}
```

Algorithm Patterns – examples

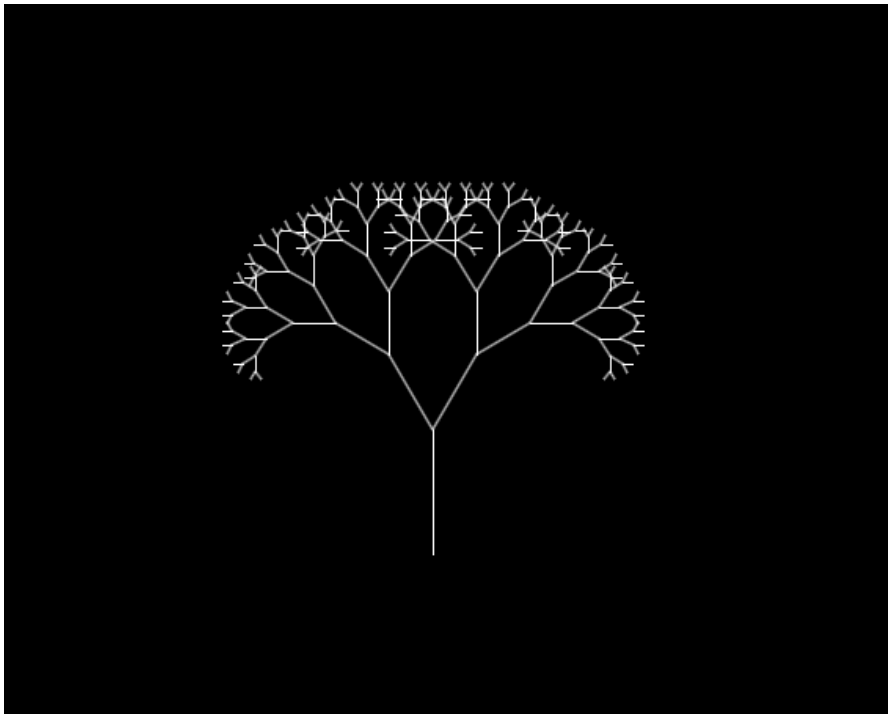
Sinusoidal Wave Pattern



```
void setup() {
  size(400, 400);
  background(0);
  stroke(255);
  noFill();

  for (int x = 0; x < width; x += 5) {
    float y = height / 2 + sin(radians(x * 5)) * 50;
    ellipse(x, y, 10, 10);
  }
}
```

Algorithm Patterns – examples

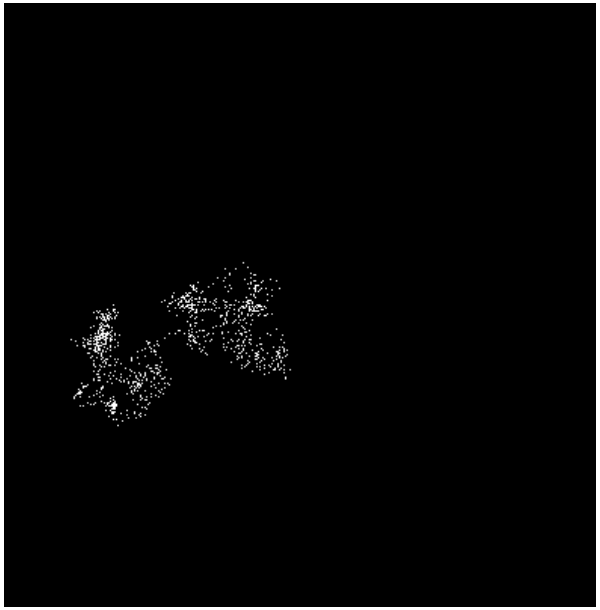
Recursive Tree (Fractal)

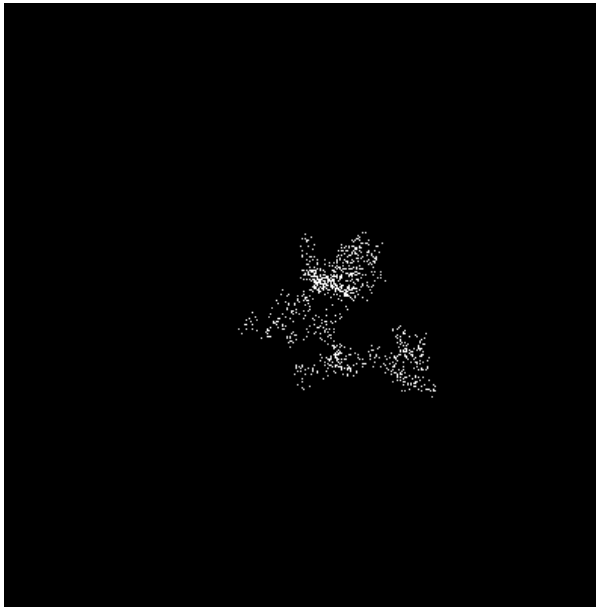


```
void setup() {
  size(400, 400);
  background(0);
  stroke(255);
  drawBranch(width / 2, height, -PI / 2, 80);
}

void drawBranch(float x, float y, float angle, float length) {
  if (length < 5) return;

  float x2 = x + cos(angle) * length;
  float y2 = y + sin(angle) * length;

  line(x, y, x2, y2);

  drawBranch(x2, y2, angle - PI / 6, length * 0.7);
  drawBranch(x2, y2, angle + PI / 6, length * 0.7);
}
```

Algorithm Patterns – examples

Random Walk Patern

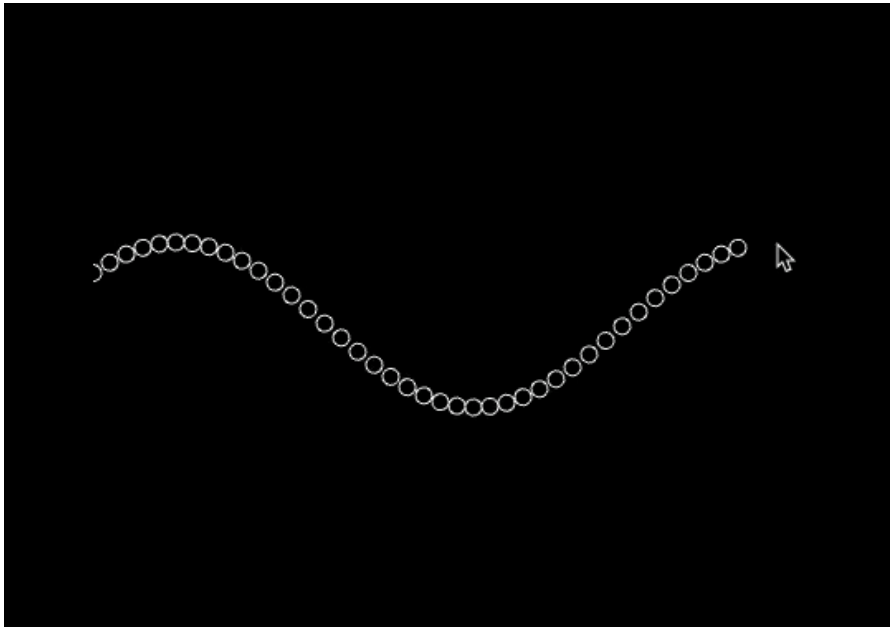




```
void setup() {
  size(400, 400);
  background(0);
  stroke(255);

  float x = width / 2;
  float y = height / 2;

  int i = 0;
  while (i < 1000) {  // Draw 1000 points
    point(x, y);
    x += random(-5, 5);
    y += random(-5, 5);
    i++;
  }
}
```

Algorithm Patterns – examples (with animation)

```
void setup() {
  size(400, 400);
  background(0);
}

void draw() {
  background(0);
  stroke(255);
  noFill();

  for (int x = 0; x < width; x += 10) {
    float y = height / 2 + sin(radians(frameCount + x)) * 50;
    ellipse(x, y, 10, 10);
  }
}
```



Nested loops

A **nested loop** is a loop inside another loop. This means that for each iteration of the **outer loop**, the **inner loop** runs completely.

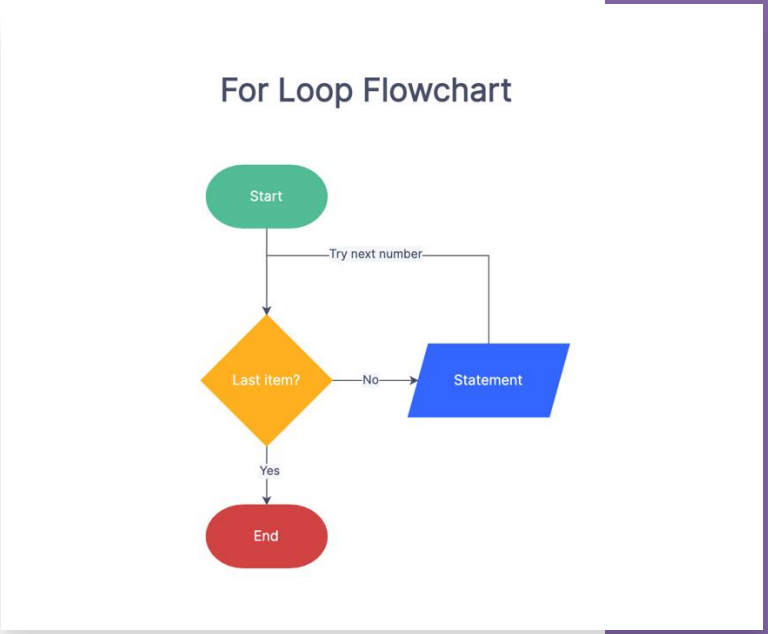
How It Works

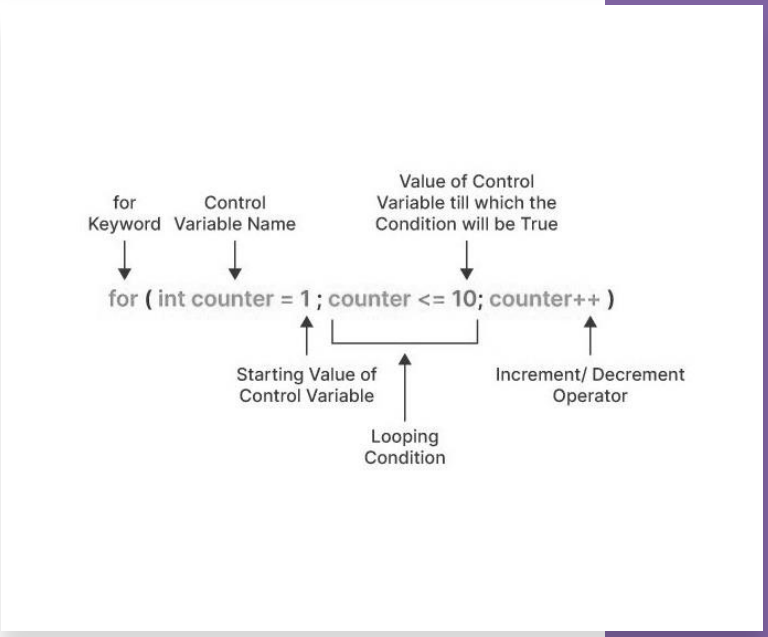
1. The **outer loop** runs once.
2. The **inner loop** runs **completely** for each iteration of the outer loop.
3. The process repeats until the outer loop finishes.

```
for (int i = 0; i < 3; i++) {  // Outer loop
  for (int j = 0; j < 3; j++) {  // Inner loop
    println("i = " + i + ", j = " + j);
  }
}
```

```
i = 0, j = 0
i = 0, j = 1
i = 0, j = 2
i = 1, j = 0
i = 1, j = 1
i = 1, j = 2
i = 2, j = 0
i = 2, j = 1
i = 2, j = 2
```

For Loop sintax

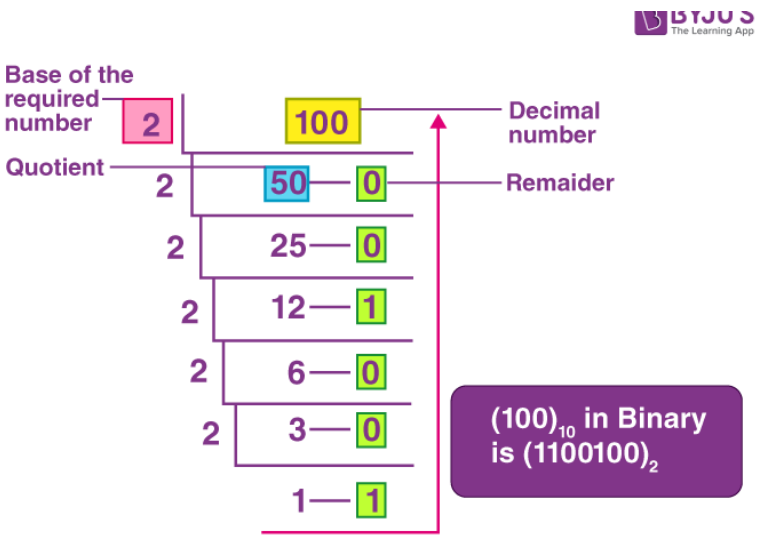


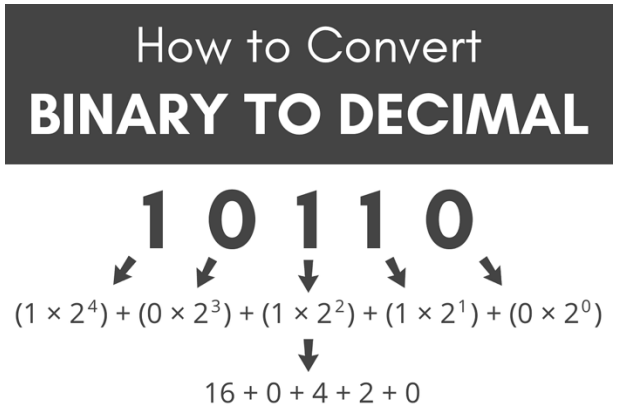


Boolean Logic

Operator	Operation
++ --	increment, decrement
+ -	unary plus, minus
!	boolean not
(<type>)	cast to <type>
* / %	multiplication, division, remainder
+ -	addition/concatenation, subtraction
< <= > >=	relational ordering
== !=	relational equality, inequality
&&	boolean and
||	boolean or
= += -= *= /= %=	assignments

Binary to Decimal // Decimal to Binary

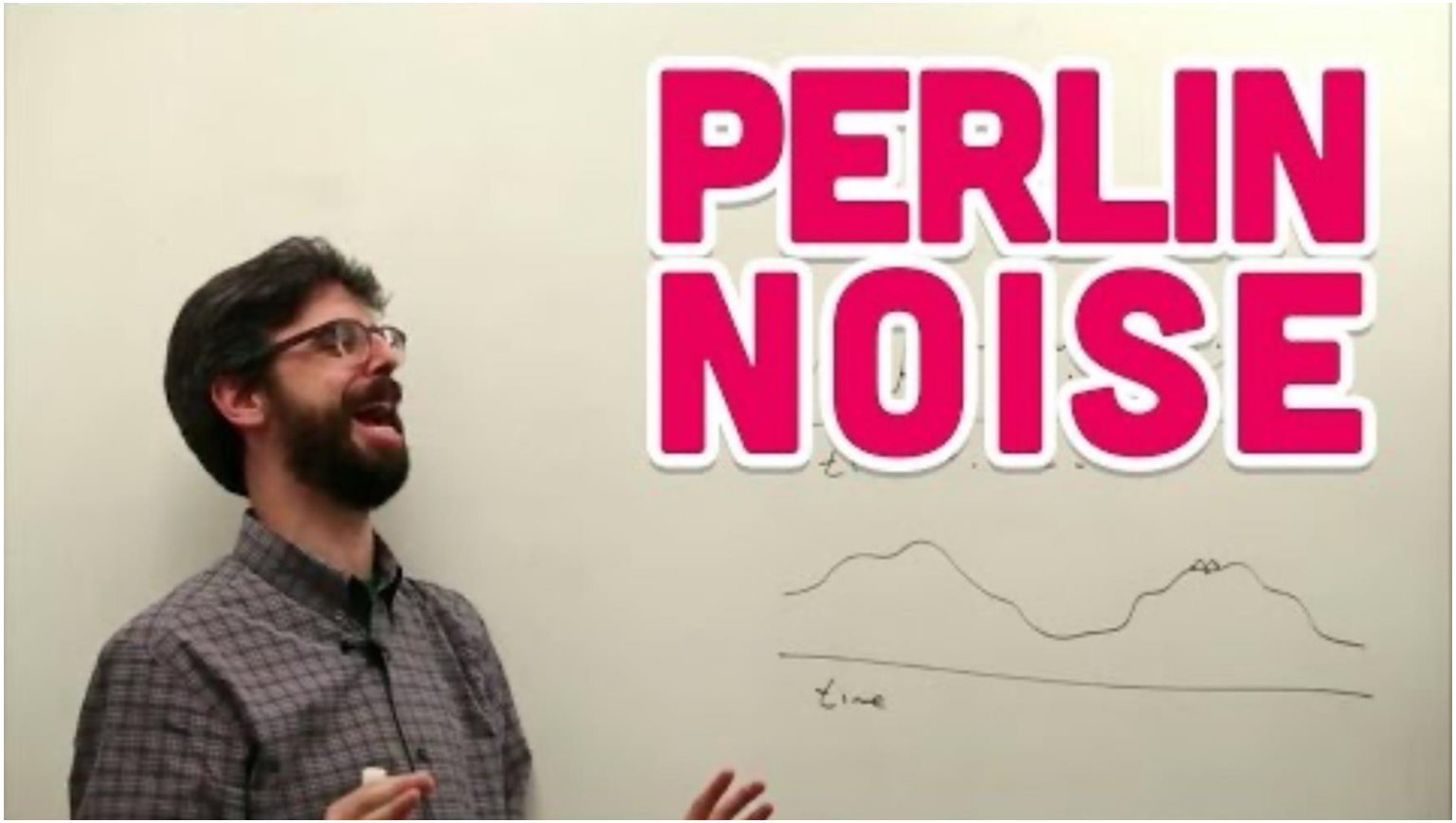




Binary	Decimal	Hex
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	10	A
1011	11	B
1100	12	C
1101	13	D
1110	14	E
1111	15	F

Binary to Decimal

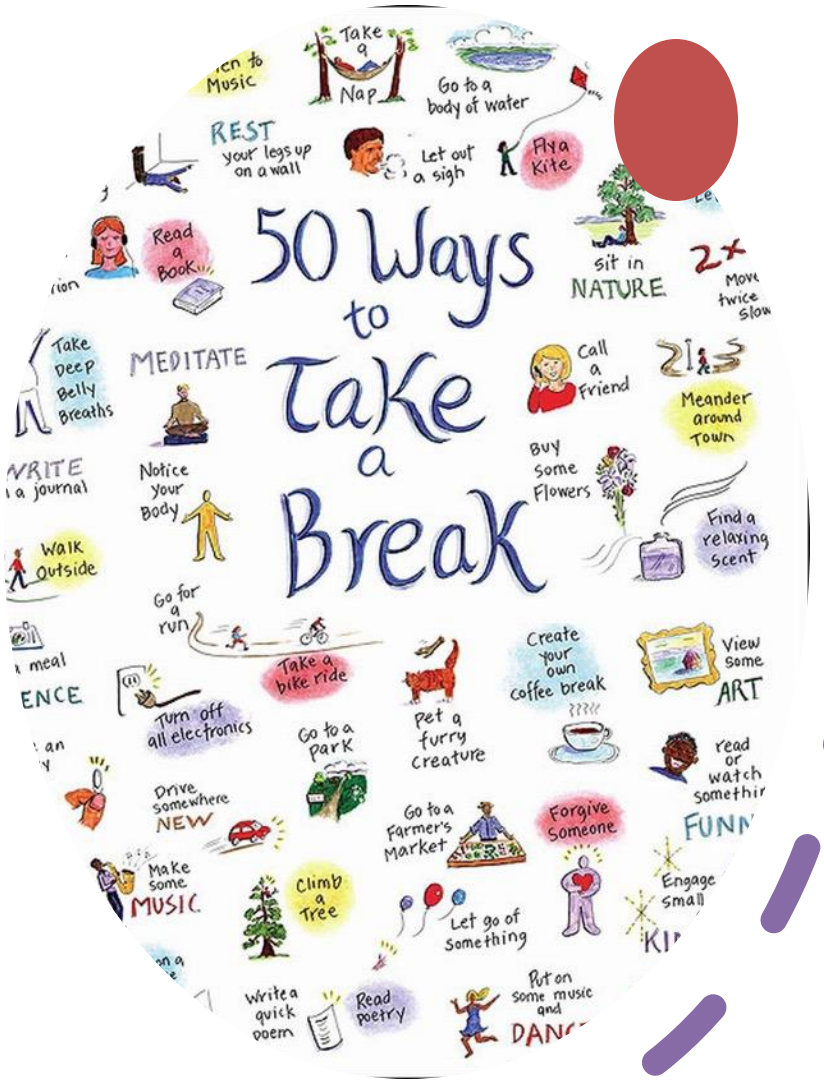
Perlin Noise



Break

15 min.!

Please don’t be late!



Hands-On Exercise!

Objectives:

- Still a grid, but a generative one.
- Things happen when hovering.
- Experiment! Try out things!

Get an example from here: https://github.com/ptiagomp/aalto-programming-visual-artists-24-25/tree/main/Session-04_04032025

(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?

Next week's topics:

- Functions, Modular code.
- Objects and arrays.

Don't forget the assignments!