

WEEK 7

AMBIENT:

RANDOM, NOISE, MATHS

# Ambient Music

A genre of music that puts an emphasis on tone and atmosphere over traditional musical structure or rhythm.

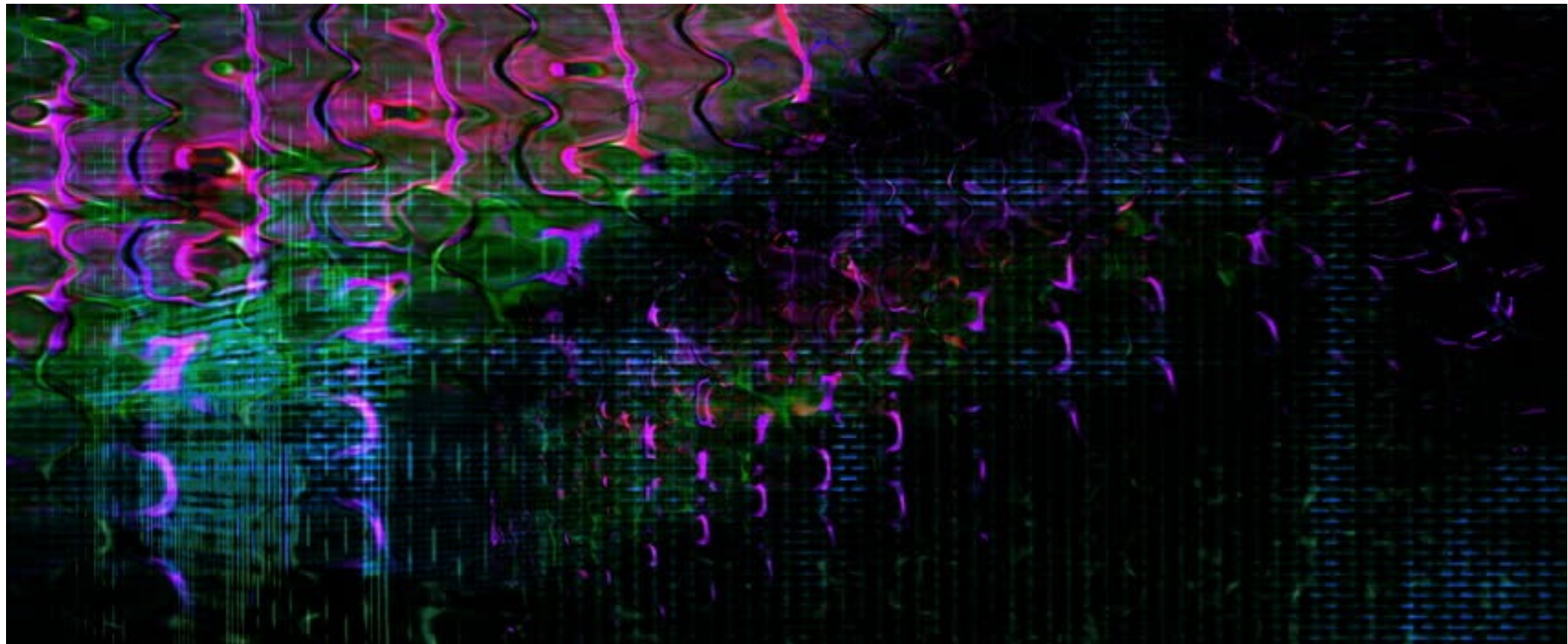
"Ambient music must be able to accommodate many levels of listening attention without enforcing one in particular; it must be as ignorable as it is interesting."

- According to Brian Eno, one of its pioneers

# Ambient Art

“...focused on the creation of a representation of what might be termed the ‘**mood**’ of a place, and in the modifications that occur as users interact **indirectly** with the artefact.”

Beale, Russell. "Ambient art: information without attention." HCI International. 11th International Conference on Human-Computer Interaction, Las Vegas, Nevada, USA. 2005.







# Properties of Ambient Art

- No fixed end point, runs continuously
- Never static, constantly changing
- Does not require continual attention from audience
- Seeks to create a mood / environment
- May explore relationships between multiple senses, or modalities, e.g., sight/sound

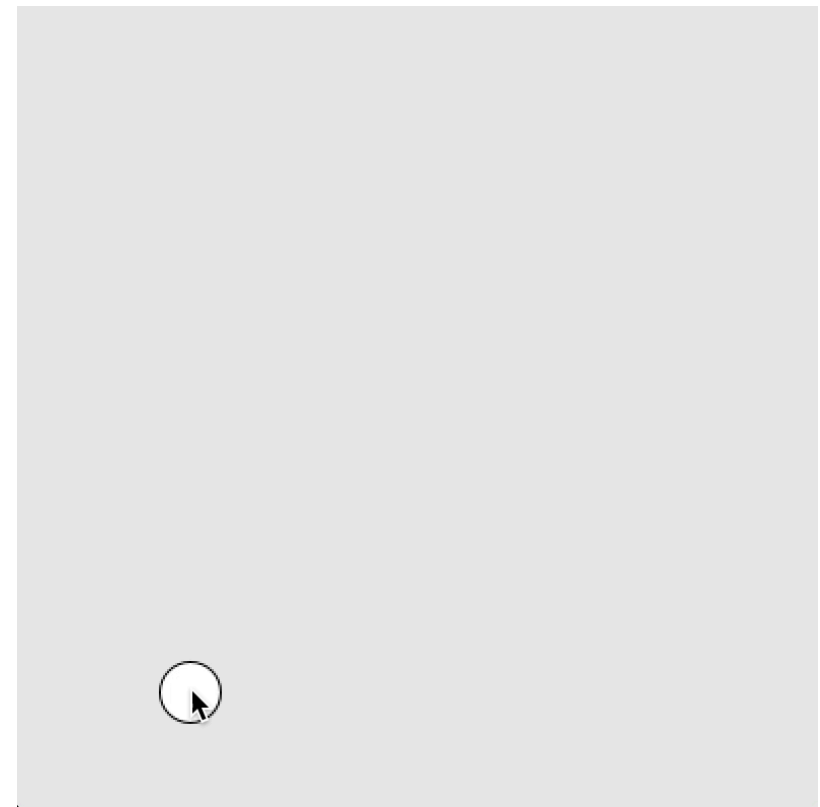
# Structure of Ambient Art

- Uses a dynamic or fluid structure, often via mathematical functions, generative systems, external data sources, or user interaction
- Structure is not immediately recognizable, but perceived subconsciously, or over longer durations of time
- Maintains a loose mapping between data and representation
- May explore emergent behavior to create interest or surprise

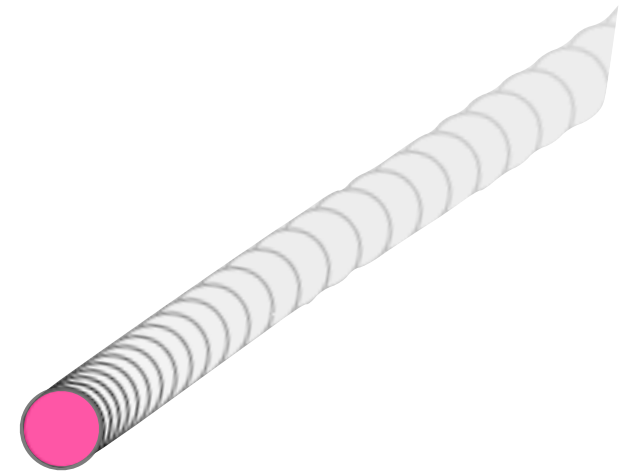
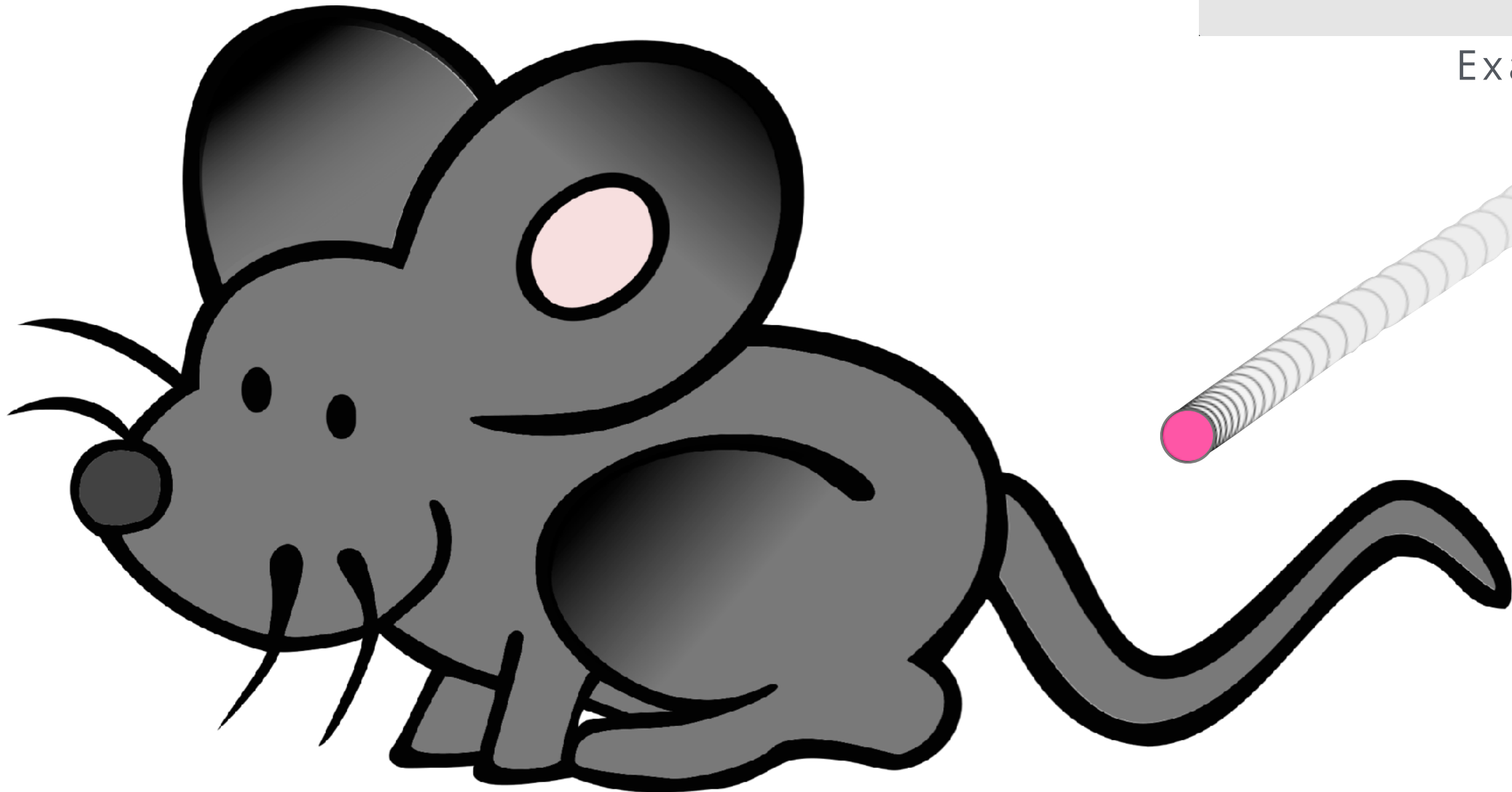
# Techniques for Ambient Art

- Randomness
- Mathematical functions: noise, sin/cos, modulo
- Generative algorithms / systems, often derived from natural systems: flocking/schooling, automata, physical simulation, evolutionary systems
- Emergent behaviors

let's play follow the mouse...



Example 1a



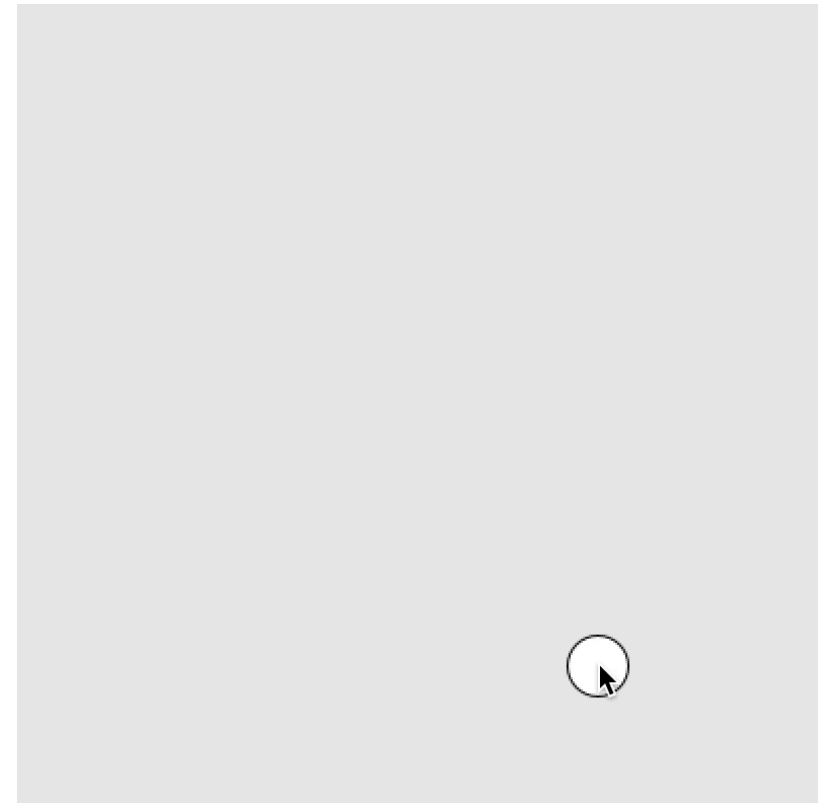


# Natural Motion

`lerp(start, stop, amt)`

- Linear Interpolation
- calculates a number between two numbers at a specific increment
- amt: float between 0.0 and 1.0

```
// find a location between two numbers
lerp(0, 100, .1); // 10
lerp(0, 100, .2); // 20
lerp(0, 100, .9); ?
lerp(0, 100, 1); ?
```



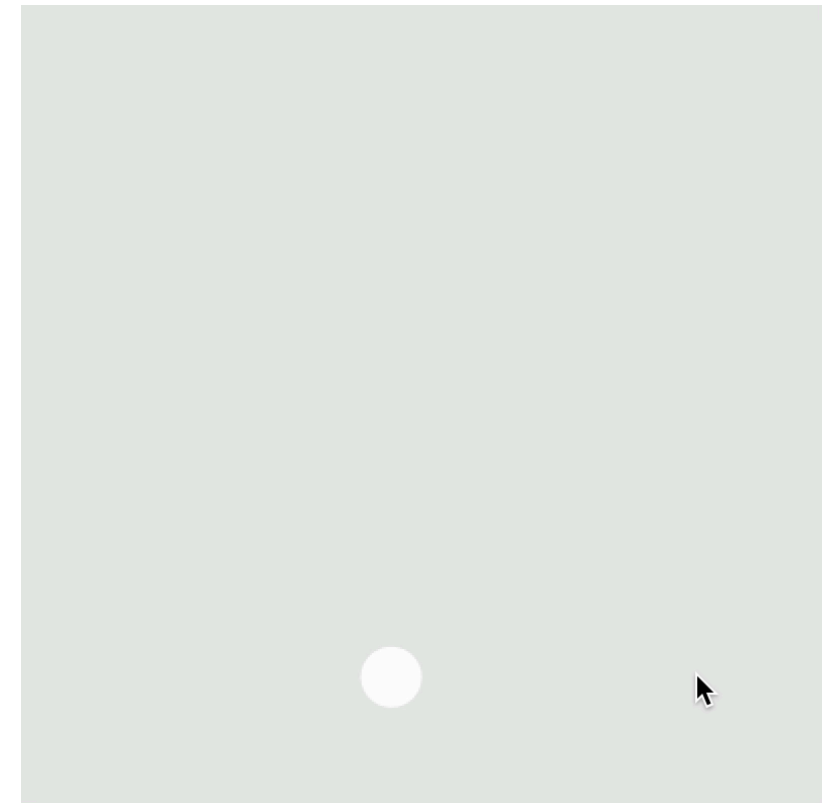
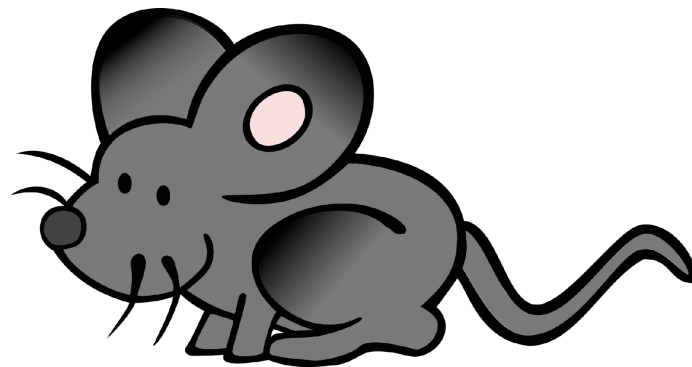
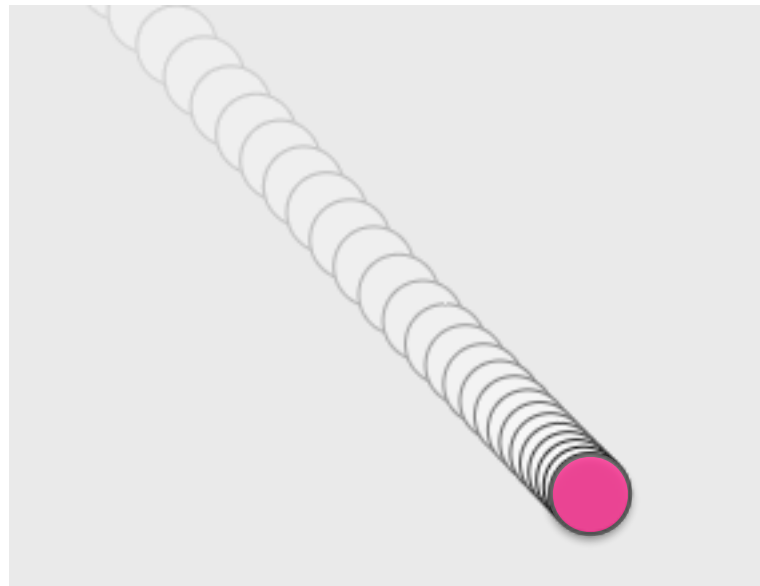
Example 1b

```
float x=0, y=0;

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

void draw() {
  background(230);
  // now follow the mouse naturally
  x = lerp(x, mouseX, .05);
  y = lerp(y, mouseY, .05);
  ellipse(x, y, 30, 30);
}
```

What if we want the color of our circle to change as we near the mouse...



Example 1c

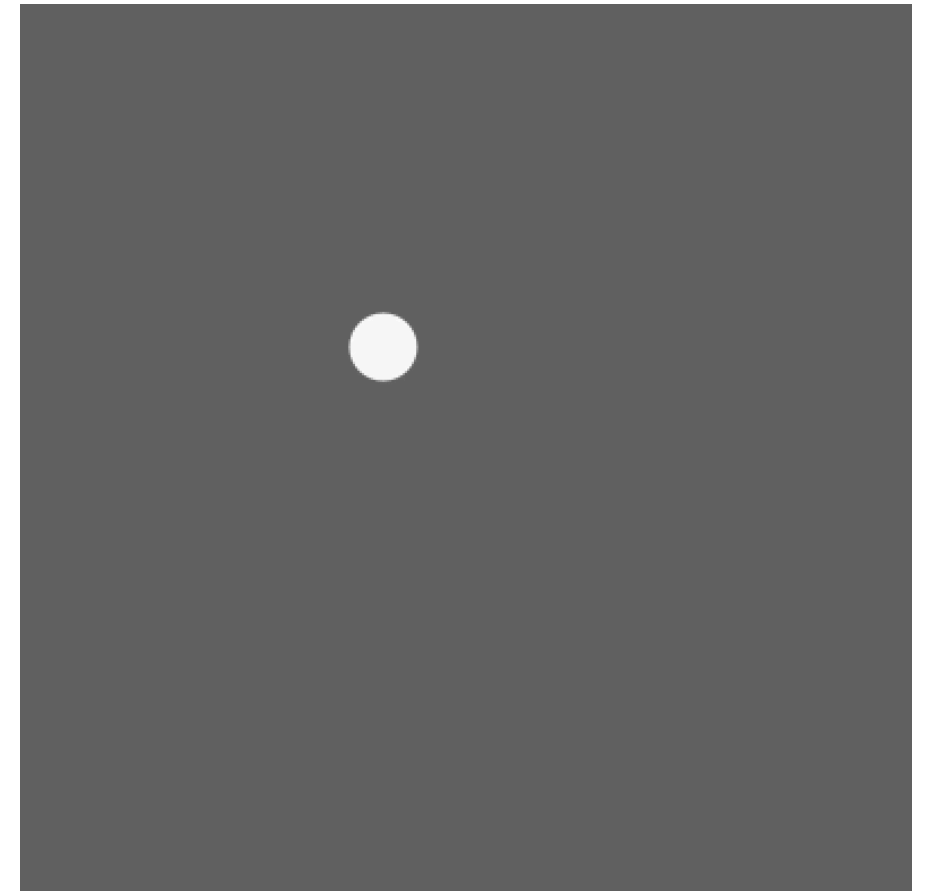
How can we tell when we are close?

```
// dist() takes 2 points (x1, y1, x2, y2)
```

```
float d = dist(x, y, mouseX, mouseY);
```

# Exercise 1

- Use `lerp()` and `dist()` to move a circle smoothly around the screen, from random point to random point
- The circle's color should gradually change, getting brighter as it nears its destination each time
- Start with Example 1c





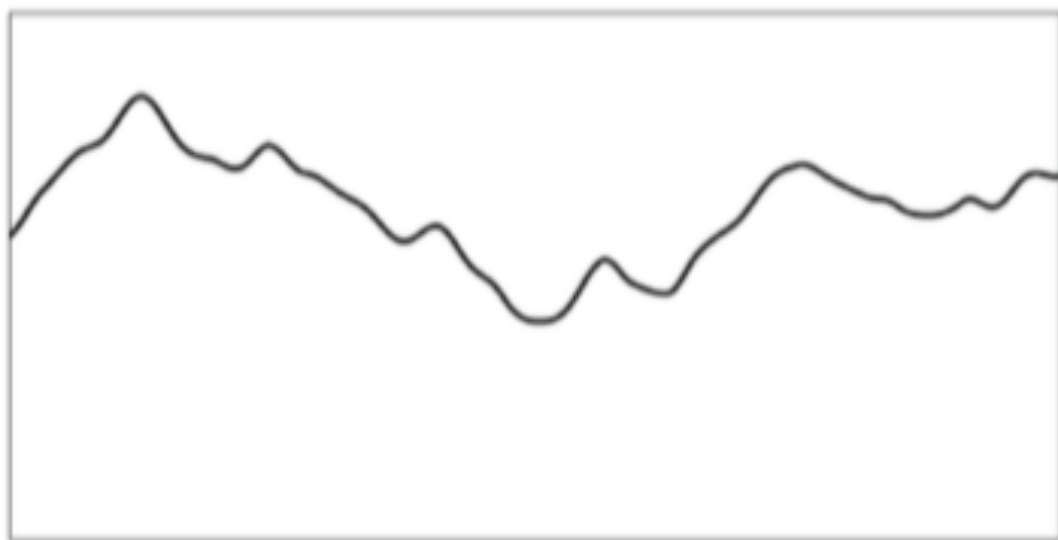
Example 1d

# Random Numbers vs Perlin Noise

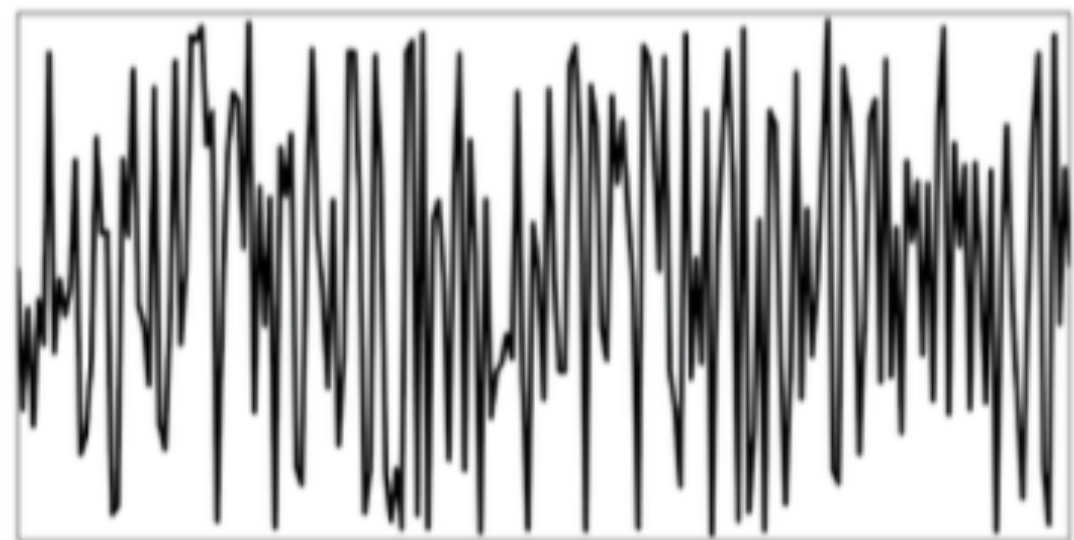
**Random numbers:** a series of numbers with no perceptible pattern and no relationship between them (independent of each other)

**Perlin noise:** a naturally ordered (i.e., “smooth”) sequence of random numbers

- Invented by Ken Perlin for producing procedure textures




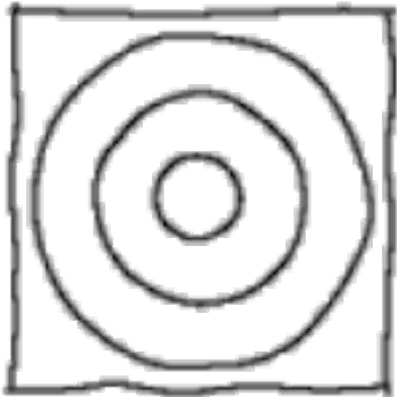

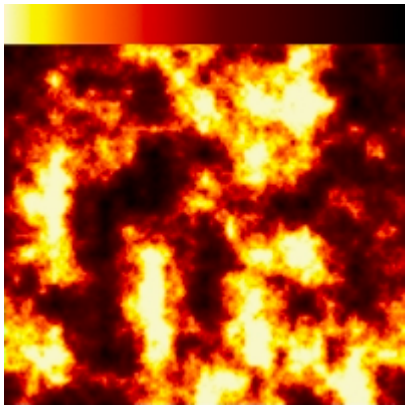
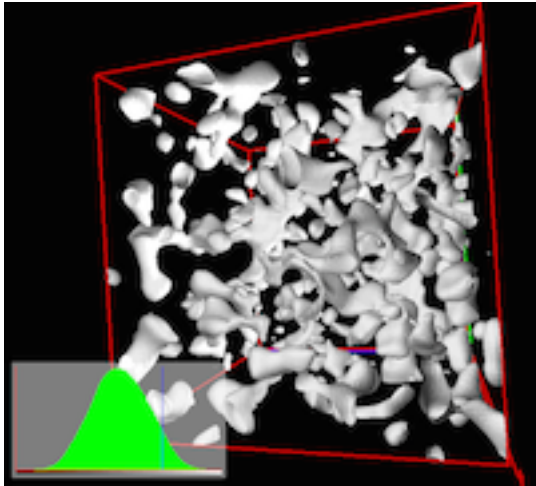

Perlin noise  
over time



Random numbers  
over time



# Applications of Perlin Noise in different dimensions

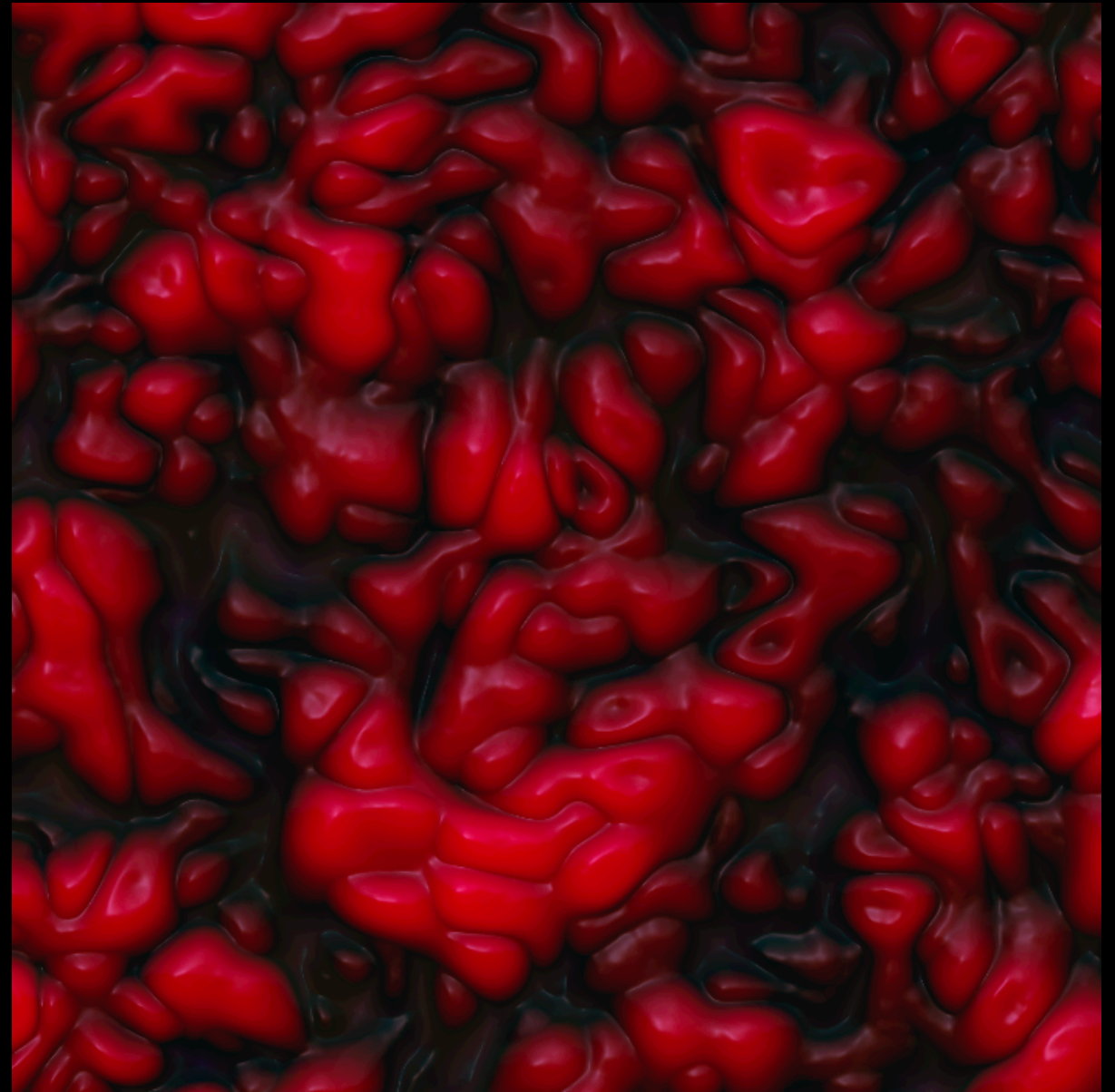
Noise Dimension	Raw Noise (Grayscale)	Use Case
1		 <p>Using noise as an offset to create handwritten lines.</p>
2		 <p>By applying a simple gradient, a procedural fire texture can be created.</p>
3		 <p>Perhaps the quintessential use of Perlin noise today, terrain can be created with caves and caverns using a modified Perlin Noise implementation.</p>



Generated with Perlin noise...



A virtual landscape



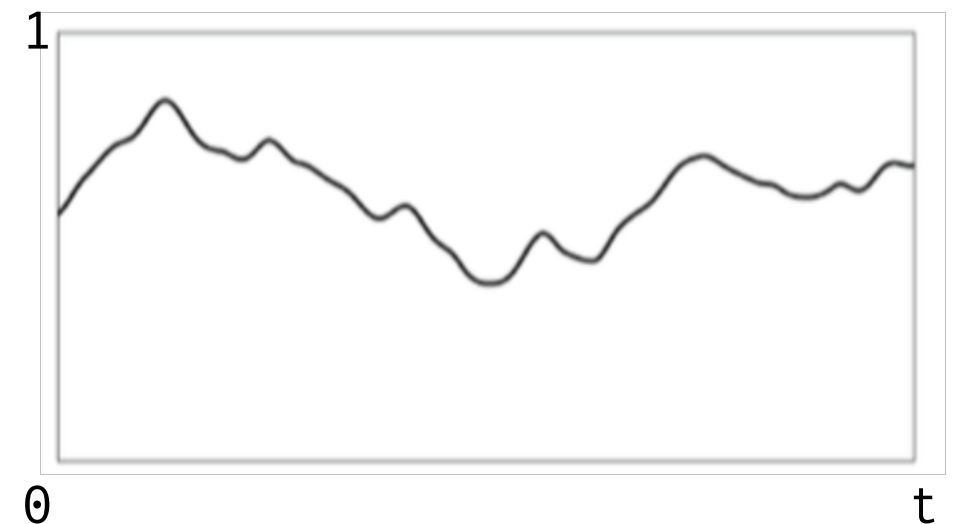
An organic surface

# Perlin Noise in Processing

`noise()` function:

“a random sequence generator producing a more natural ordered, harmonic succession of numbers compared to the standard `random()` function..”

1D Perlin noise `noise(t)`: a linear sequence of values (between 0 and 1) over time `t`



`float t = 0.0;`      Example 2

```
void draw() {  
    float noisevalue = noise(t);  
    println(noisevalue);  
}
```

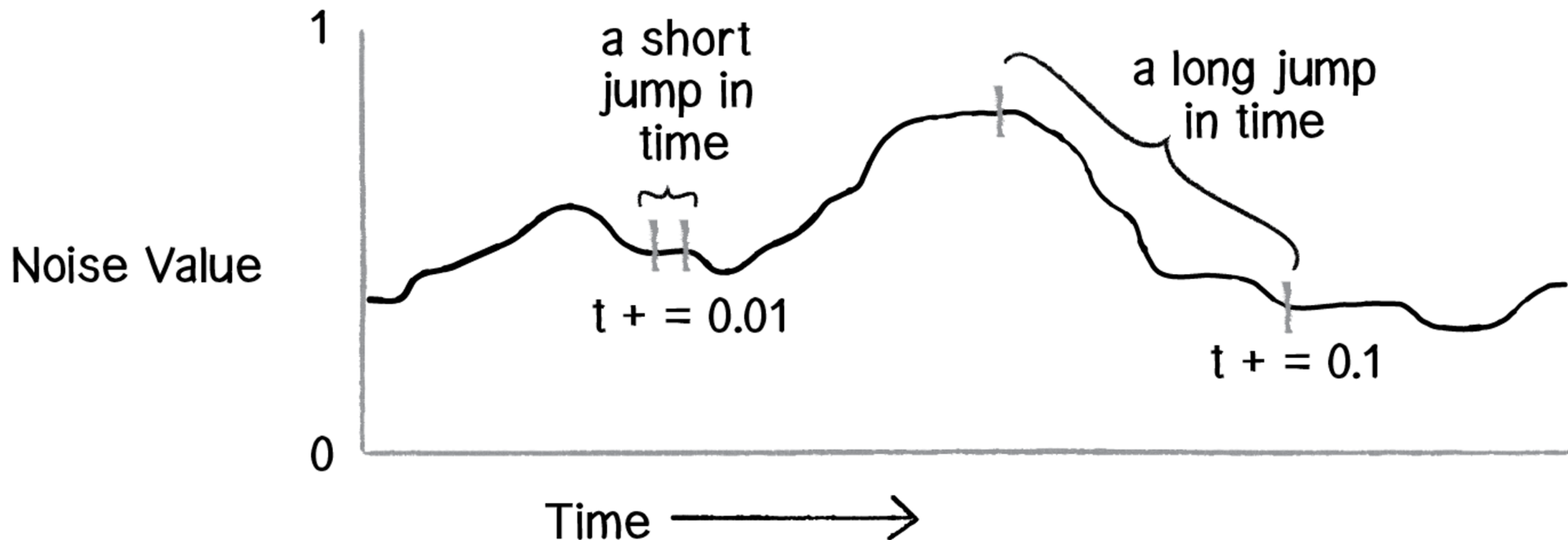
```
// t += 1;  
// t += 0.01;  
// t += 0.0001;  
}
```

Numbers move up and down randomly but stay close to the value of their predecessor

The smaller the increment, the smoother the resulting noise sequence



# Perlin Noise in Processing



- The initial value of  $t$  doesn't matter. What it matters is the **step size** (i.e., the change to  $t$ ).
- If we make large jumps in time, then we are skipping ahead and the values will be more random.
- Steps of 0.005-0.03 work best for most applications

# Natural Movement of Line

## Random vs Noise

```
void setup() {  
  size(400, 400);  
}  
  
void draw() {  
  background(230);  
  
  float x = random(0, width);  
  line(x, 0, x, height);  
}
```

Example 3a

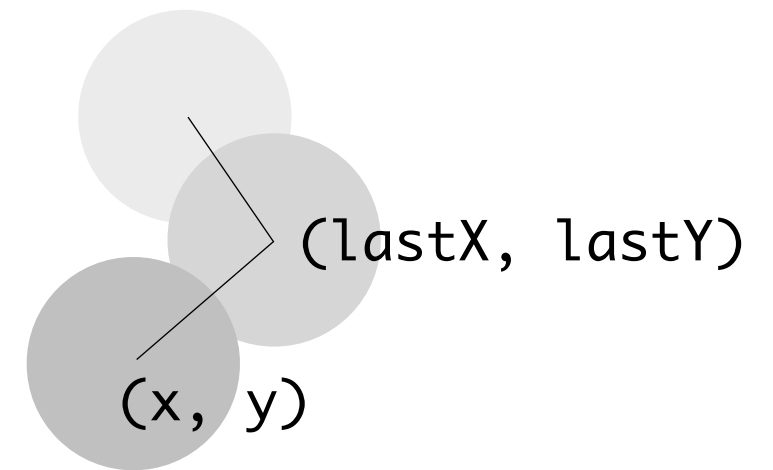
```
float num = 0;  
  
void setup() {  
  size(400, 400);  
}  
  
void draw() {  
  background(230);  
  
  float x = noise(num) * width;  
  line(x, 0, x, height);  
  num = num + 0.01;  
}
```

Example 3b



# Example 4

```
void draw() {  
  fill(255, 32);  
  rect(0, 0, width, height);  
  
  stroke(0);  
  
  float x = noise(num) * width;  
  if (y > lastY)  
    line(lastX, lastY, x, y);  
  
  noStroke();  
  fill(0, 32);  
  ellipse(x, y, 15, 15);  
  
  lastX = x;  
  lastY = y;  
  
  y = (y + 5) % height;  
  num = num + 0.01;  
}
```

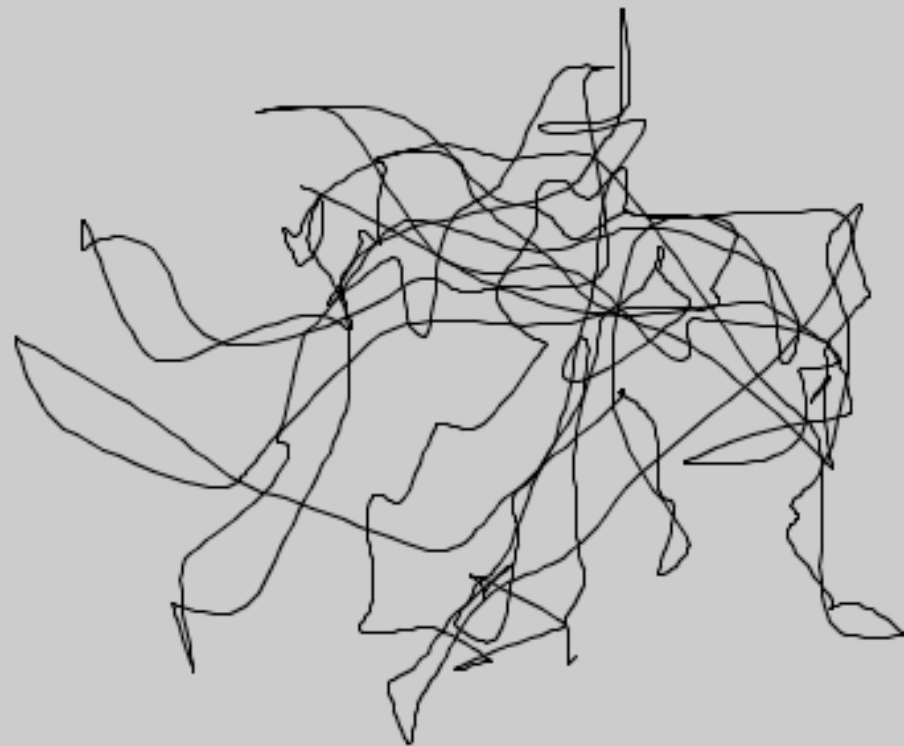


# Random Walk with Natural Motion



```
x2 = x1 + random(-5, 5);  
y2 = y1 + random(-5, 5);
```

Example 5a



```
x2 = (noise(t)) * width;  
y2 = (noise(t + 100)) * width;
```

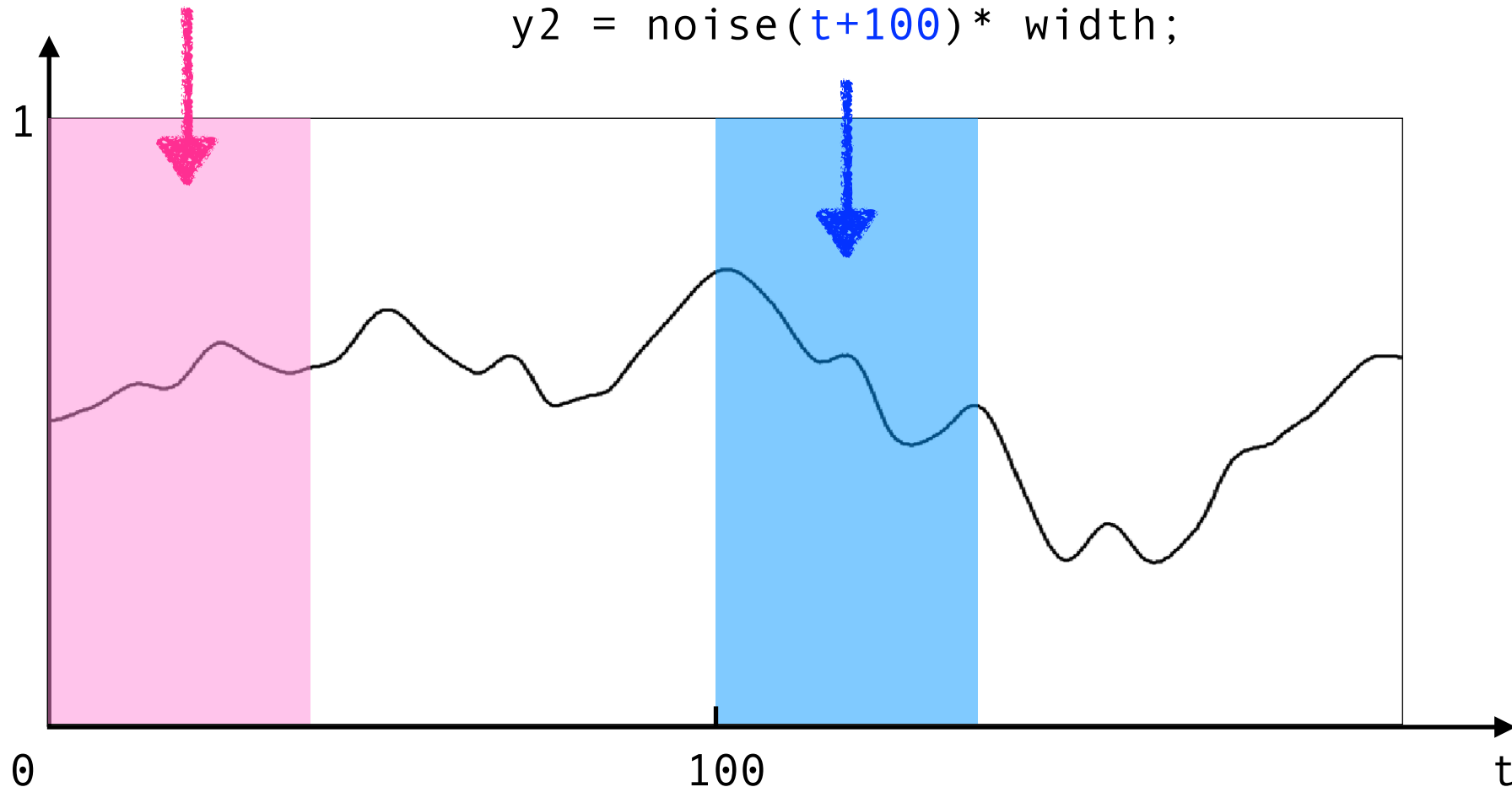
↑  
to avoid x2 always being equal to y2

Example 5b

# Noise Space in Example 5b

```
x2 = noise(t) * width;
```

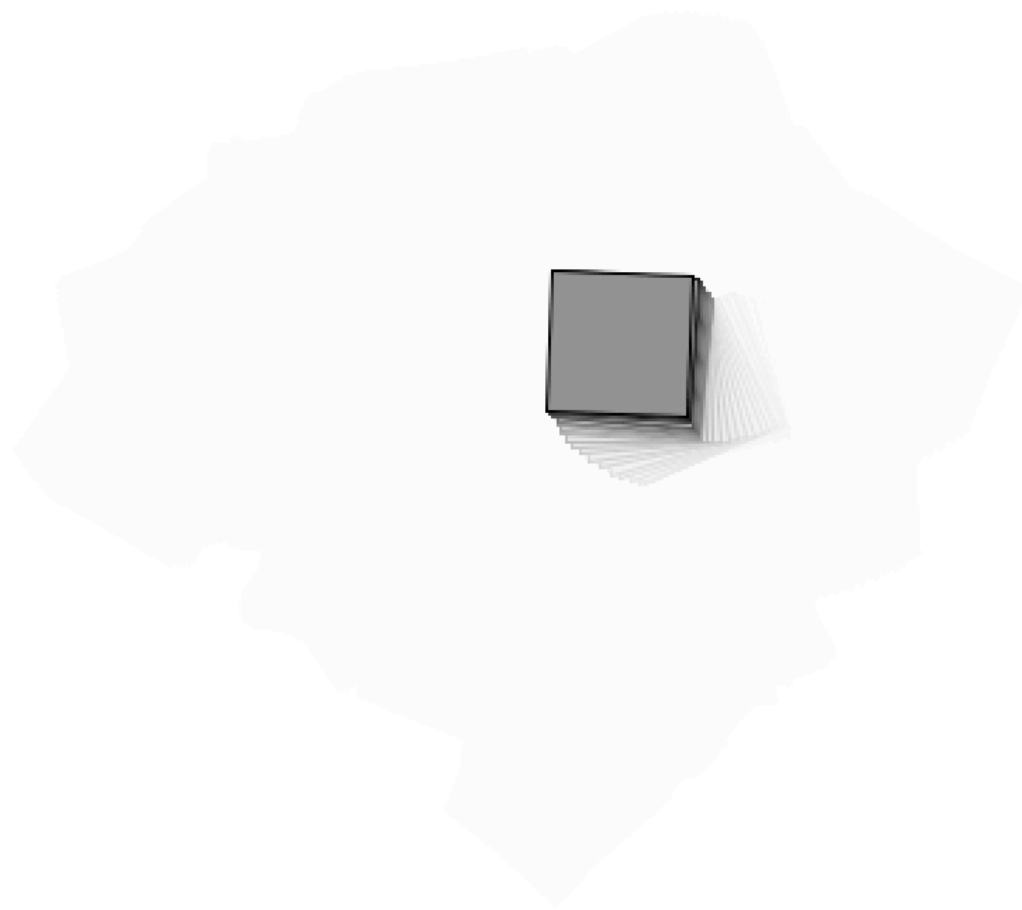
```
y2 = noise(t+100) * width;
```



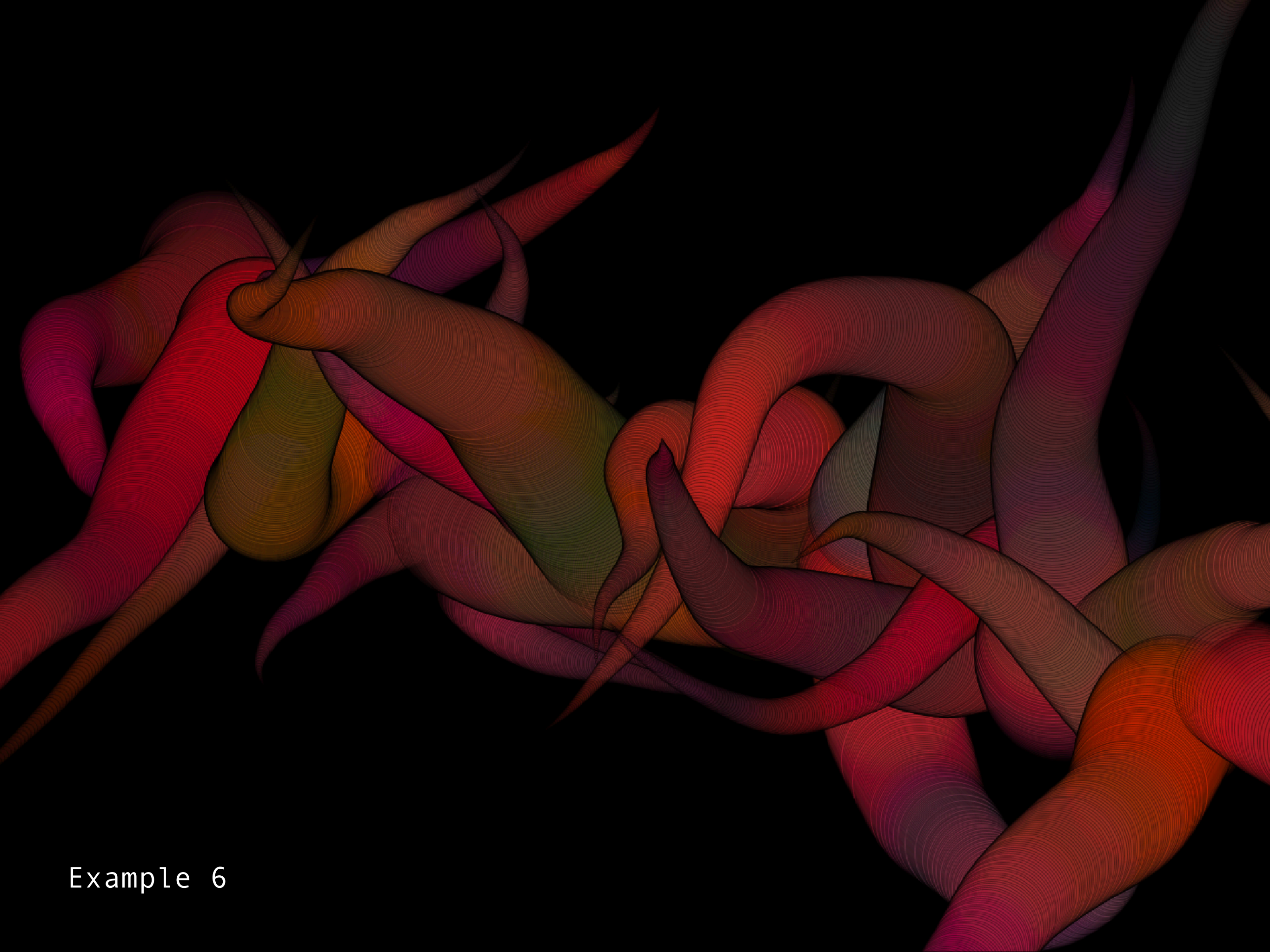
Why does `x2` start at 0 and `y2` at 100? While these numbers are arbitrary choices, we have very specifically initialized our two time variables with different values. This is because **the noise function is deterministic** (once it is initialized every time): it gives you the same result for a specific time `t` each and every time. If we ask for a noise value at the same time `t` for both `x` and `y`, then `x2` and `y2` would always be equal, meaning that the Walker object would only move along a diagonal. Instead, we simply use two different parts of the noise space, starting at 0 for `x2` and 100 for `y2` so that `x2` and `y2` can appear to act independently of each other.

# Exercise 2

Based on Exercise 2 Template, make the square move (rotate and translate) more smoothly by using `noise()`.







Example 6



# Example 6

- every parameter has its own noise space and step size (update rate).

```
float t_x = 10, t_y = 20, t_r=30, t_g=40, t_b=50, t_a=60;
```

```
.  
.   
.
```

```
t_x += 0.005;  
t_y += 0.005;  
t_r += 0.01;  
t_g += 0.005;  
t_b += 0.005;  
t_a += 0.0001;
```

# noise() in More Dimensions

To use noise in multiple dimensions, we create one variable for each dimension that starts at some fixed number and increases by a small amount each loop/frame.

```
float n1, n2;

void draw()
{
    background(230);

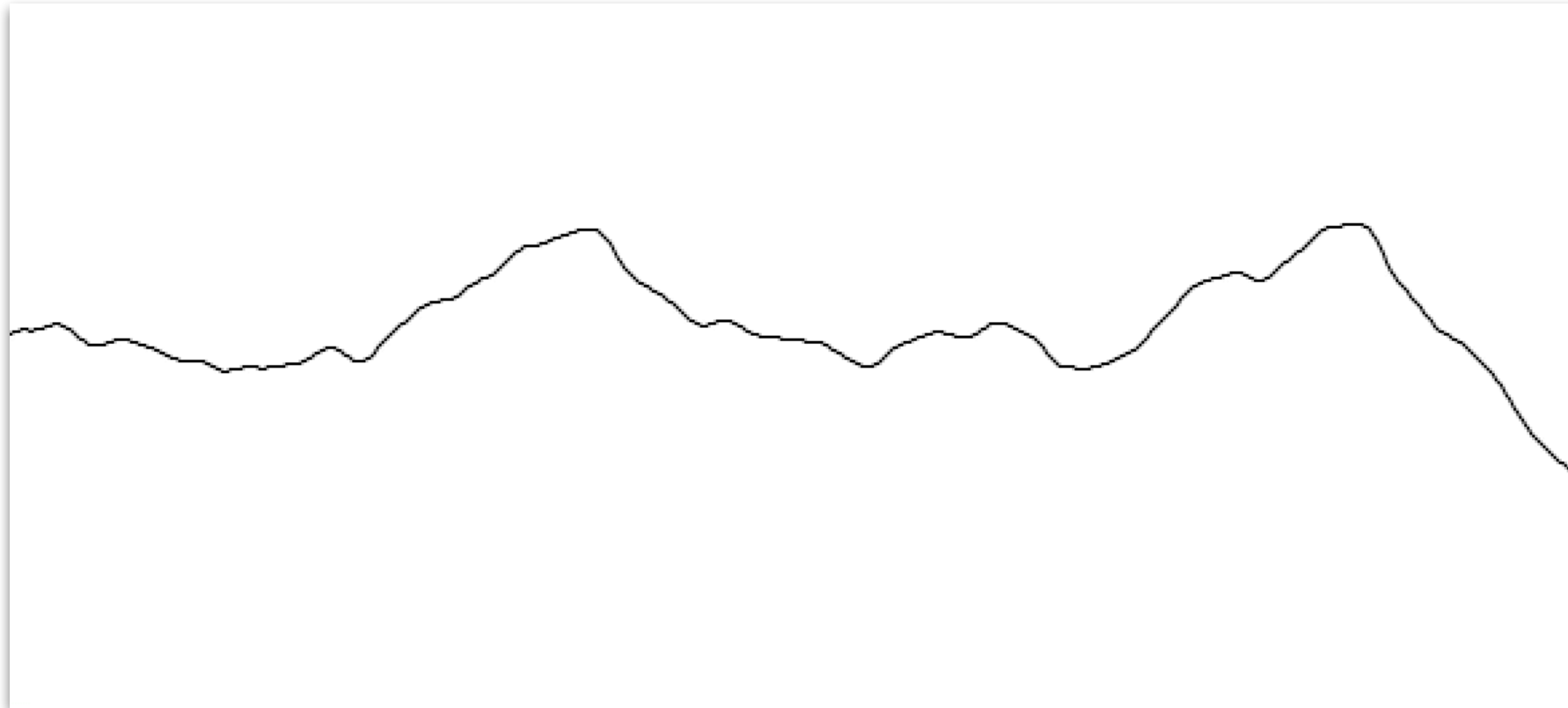
    float result = noise(n1, n2);

    ...
    n1 = n1 + 0.01;
    n2 = n2 + 0.02;
}
```

# noise() in More Dimensions

```
// lets draw points across the screen with noise  
  
size(600, 400);  
  
background(255);  
  
float x = 0; // change with horizontal position  
for (int i = 0; i < width; i++) {  
    point(i, 150 + noise(x) * 200);  
    x += 0.01;  
}
```

# noise() in More Dimensions



```
float x = 0;
for (int i = 0; i < width; i++) {
  point(i, 150 + noise(x, y) * 200);
  x += 0.01; ← shape
}
y += 0.003; ← time
```

Example 7c & 7d