Tangible Code：
Rubyで「見て・触れて・変えてわかる」コードのしくみ

11/7 RubyWorld Conference 2025

Miyuki Koshiba

koshiba 🌱 chobishiba

SmartBank, Inc.

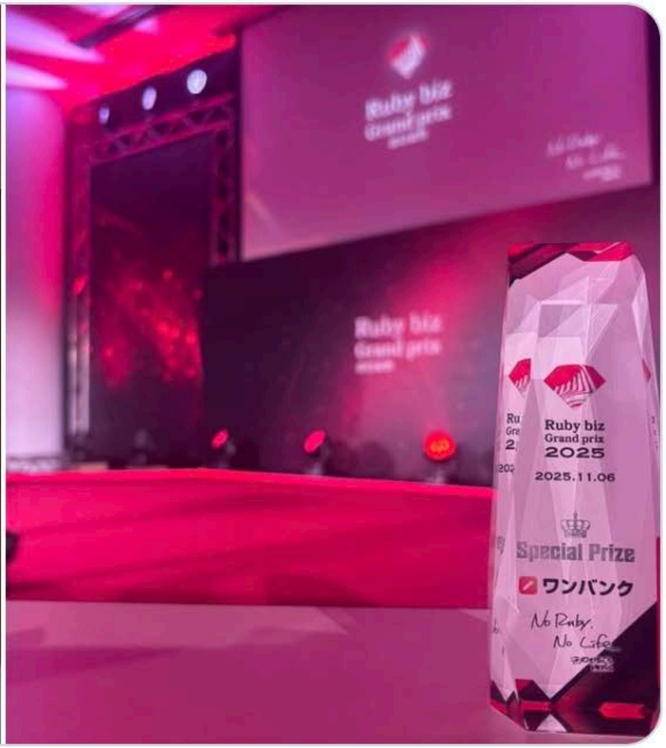
サーバーサイドエンジニア / エンジニアリングマネージャー

@chobishiba　　@ksbmyk


8:43
ホーム
ふりかえり詳細
3
今日のふりかえりをする
今月の支出
28,000 円
入金
マイカード
12,000 円
最近の履歴
1月1日（金）
ATMで入金
+3,000 円
Onebank
VISA
頑張らなくていいお金の管理
— AI家計簿アプリ —
ワンバンク

SmartBank Tech
@SmartBankTech

ワンバンクがRuby bizグランプリで特別賞を受賞させていただきました💎✨🤝



ワンバンク

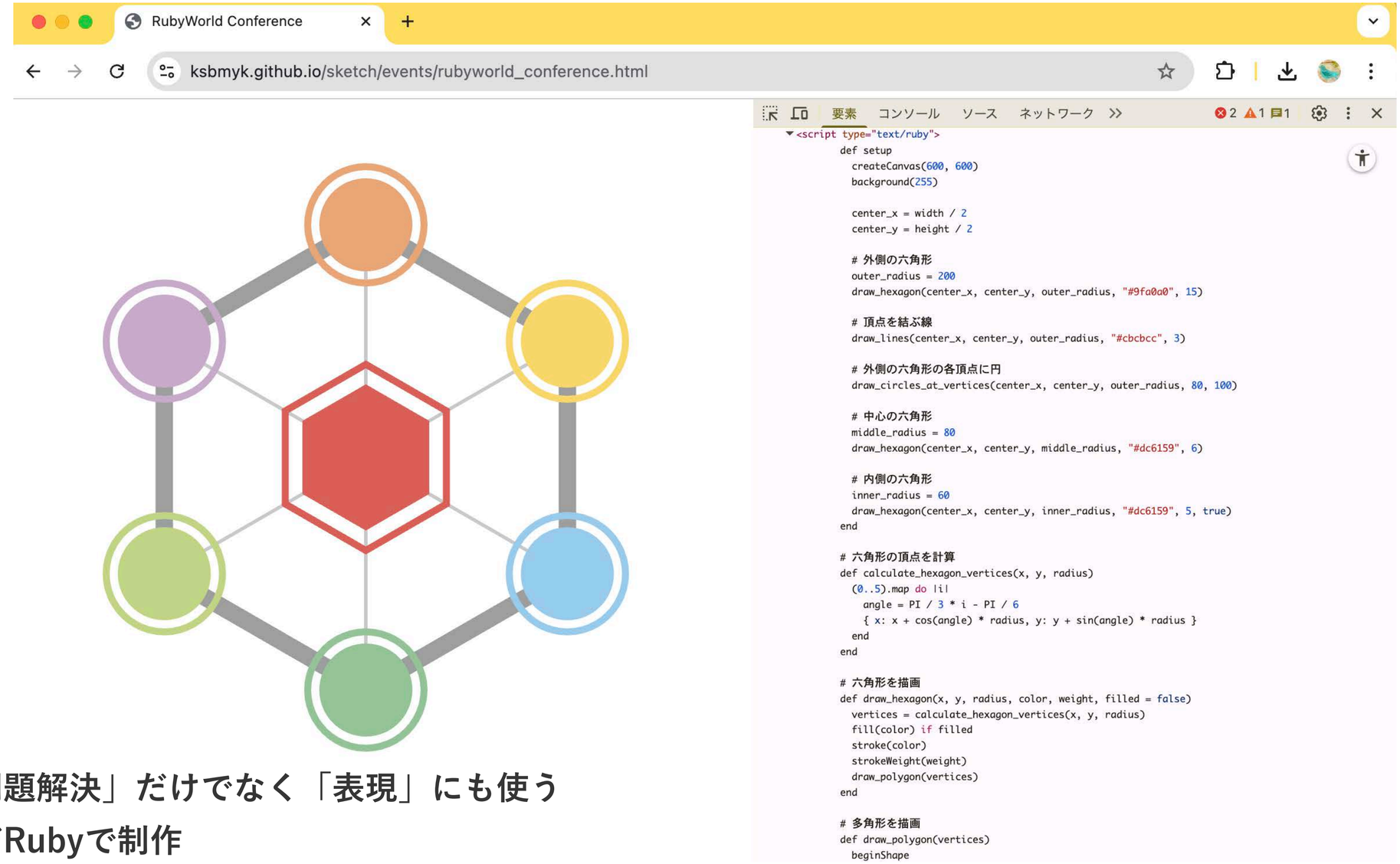
ワンバンク

ワンバンクは「がんばらなくても自然とお金が整う」次世代型のAI家計簿です。AIアシスタントが支出を学習し、次の行動をやさしくサポートします。面倒な設定なしでゲーム感覚で続けられるから、お金の不安が自然とクリアになります。他にも、ペアカードによるカップルのお金管理や、あと払いチャージで柔軟な支払いも可能。日常の支払いをスムーズにする機能が充実しています。

株式会社スマートバンク

- 武蔵野美術大学通信教育課程
  - デザイン情報学科 デザインシステムコース 卒業
- クリエイティブコーディング活動
  - 作品制作、ワークショップ、登壇

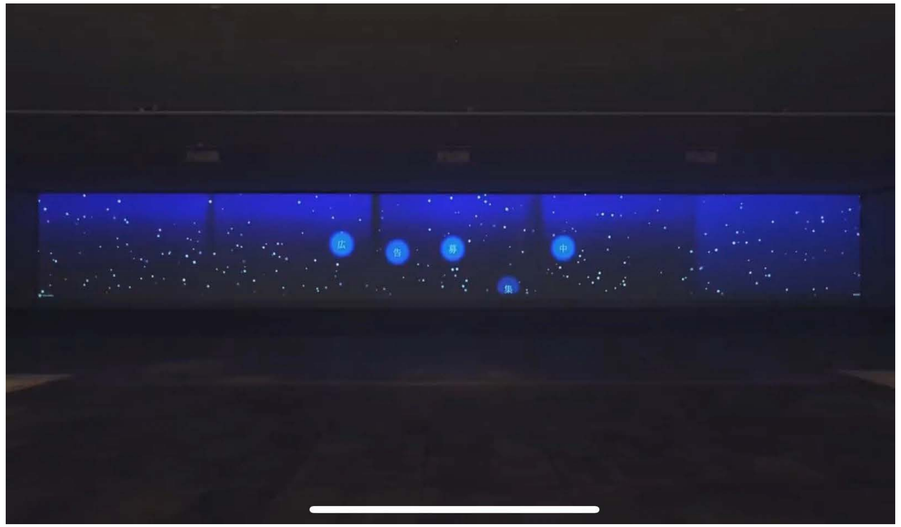
クリエイティブコーディング



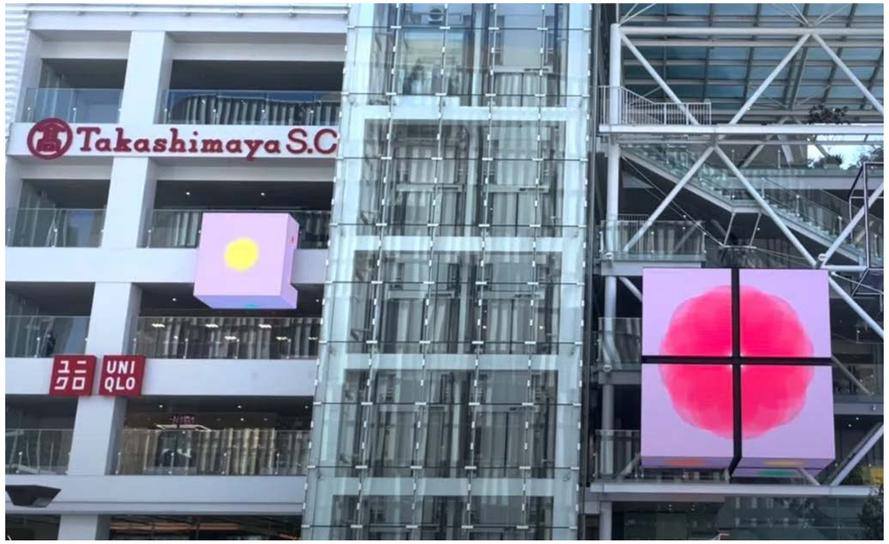
コードを「問題解決」だけでなく「表現」にも使う

慣れ親しんだRubyで制作

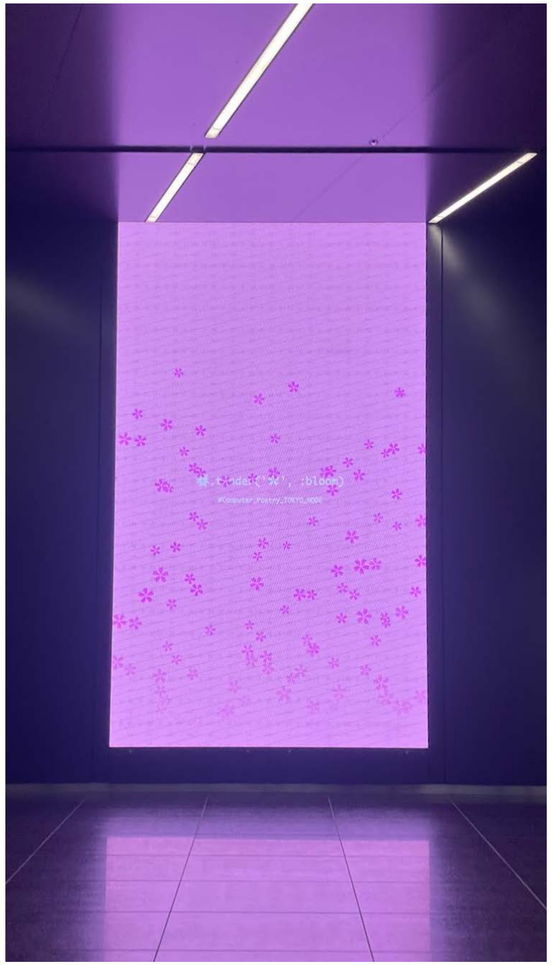
公共空間での展示



KITTE 大阪



玉川高島屋S.C.



虎ノ門ヒルズ ステーションタワー

個人でも公募・応募を通じて展示可能
通りすがりの人々が作品に偶然触れる機会

ワークショップ・登壇

ワークショップ

2025.1.17 東京Ruby会議12 前夜祭
（ブラウザでビジュアル作品制作）

mruby Girls
Matsue 1st

2025.11.8 mruby Girls. Matsue 1st
（マイコンでプログラミング体験）

登壇

YAPC::Hakodate 2024

RubyWorld Conference 2024

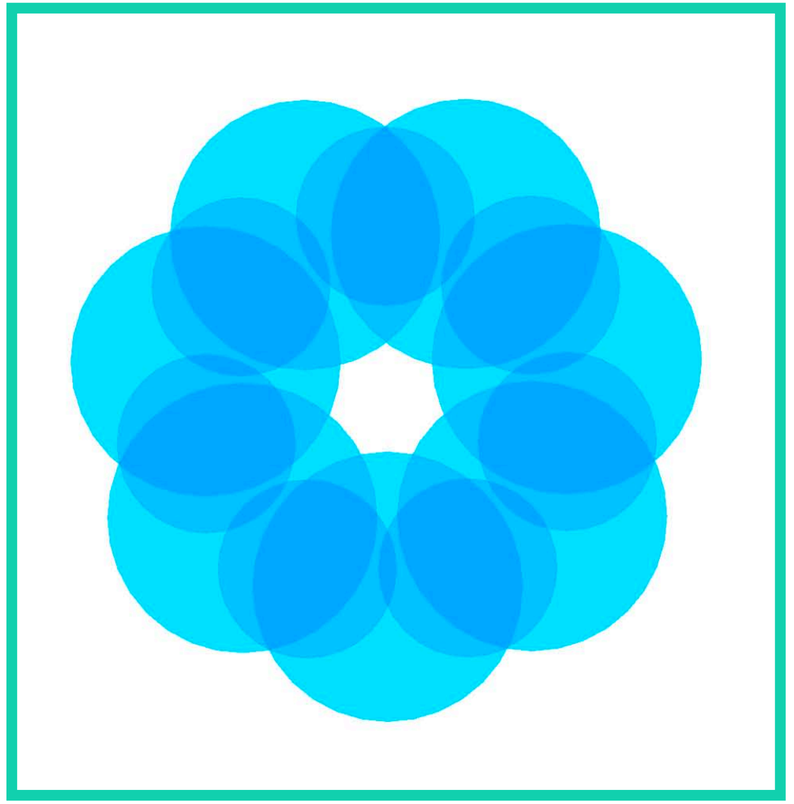
https://regional.rubykaigi.org/tokyo12/schedule/
https://mrubygirls.github.io/events/2025-11-08-matsue_1st

https://speakerdeck.com/chobishiba/creative-coding-creating-your-own-world
https://speakerdeck.com/chobishiba/creative-coding-and-learning-ruby

Tangible Code

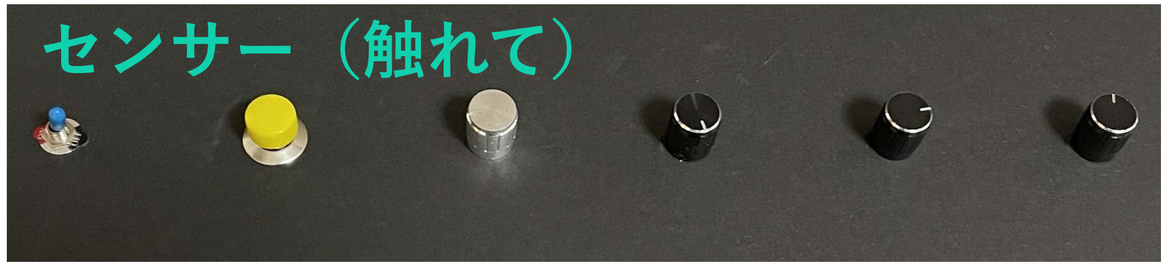
```
require "processing"
using Processing

def setup
  @circle_count = 4; @distance = 100; @circle_size = 150
  @hue_value = 200; @is_dark_mode = false
  @angle_offset = 0; @is_button_push = false

  size(displayWidth, displayHeight)
  colorMode(HSB, 360, 100, 100, 255)
  noStroke
end

def draw
  # 背景色と色の混ぜ方を設定
  @is_dark_mode = false
  if(@is_dark_mode)
    background(0, 0, 0)
    blendMode(ADD)
  else
    background(0, 0, 100)
    blendMode(MULTIPLY)
  end

  # 色を設定
  @hue_value = 190
  fill(@hue_value, 80, 100, 150)

  # 回転角度を設定
  @is_button_push = false
  @angle_offset += @is_button_push ? 0.05 : 0.01

  # 円の個数を設定
  @circle_count = 14
  @circle_count.times do |i|
    angle = TWO_PI / @circle_count * i + @angle_offset
    # 中心からの距離を設定
    @distance = 122
    x = cos(angle) * @distance
    y = sin(angle) * @distance
    # 円のサイズを設定
    @circle_size = 148
    # 円を描く
    circle(x, y, @circle_size + (i.even? ? 30 : -30))
  end
end
```

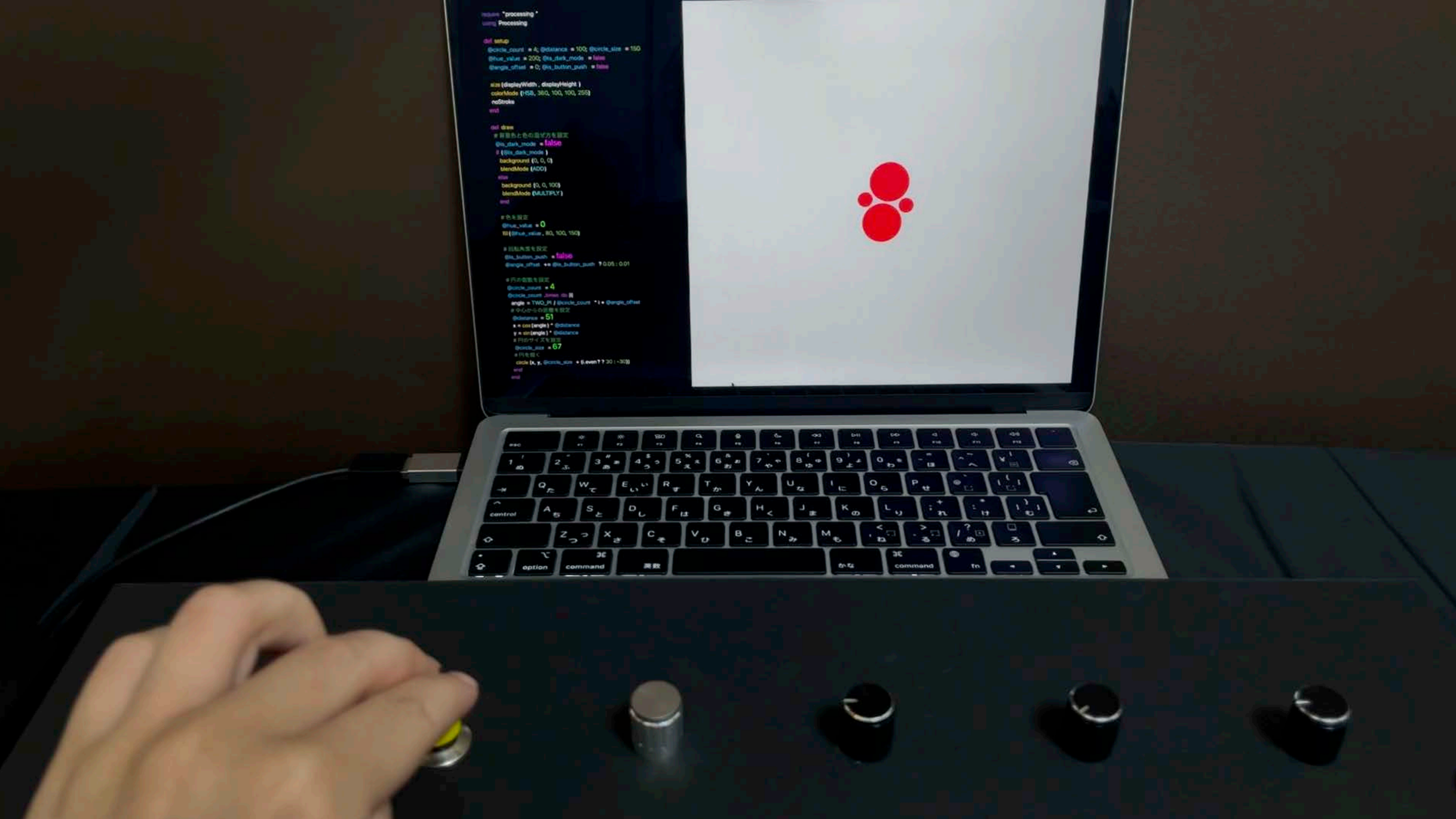
ビジュアルアート（見て）



コード（変えて）

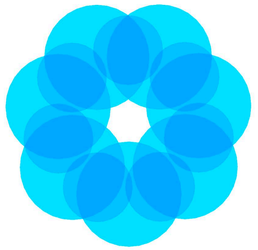
センサー（触れて）





- Tangible Code
  - センサーでコードに触れる
  - リアルタイムに視覚的フィードバック

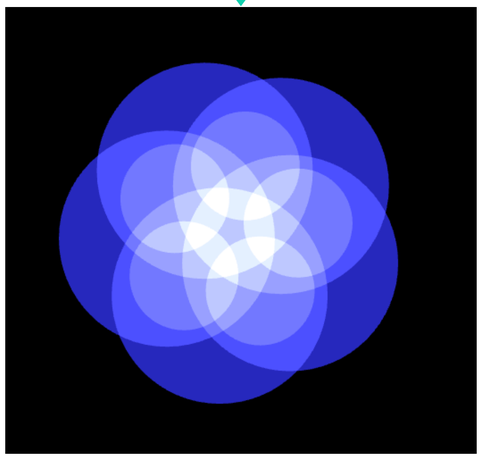
```
require "processing"
using Processing

def setup
  @circle_count = 4; @distance = 100; @circle_size = 150
  @hue_value = 200; @is_dark_mode = false
  @angle_offset = 0; @is_button_push = false

  size(displayWidth, displayHeight)
  colorMode(HSB, 360, 100, 100, 255)
  noStroke
end

def draw
  # 背景色と色の混ぜ方を設定
  @is_dark_mode = false
  if(@is_dark_mode)
    background(0, 0, 0)
    blendMode(ADD)
  else
    background(0, 0, 100)
    blendMode(MULTIPLY)
  end

  # 色を設定
  @hue_value = 190
  fill(@hue_value, 80, 100, 150)

  # 回転角度を設定
  @is_button_push = false
  @angle_offset += @is_button_push ? 0.05 : 0.01

  # 円の個数を設定
  @circle_count = 14
  @circle_count.times do |i|
    angle = TWO_PI / @circle_count * i + @angle_offset
    # 中心からの距離を設定
    @distance = 122
    x = cos(angle) * @distance
    y = sin(angle) * @distance
    # 円のサイズを設定
    @circle_size = 148
    # 円を描く
    circle(x, y, @circle_size + (i.even? ? 30 : -30))
  end
end
```



アルゴリズムアートにおける問い

「コード自体も鑑賞対象になりえないか」

```
def draw
  blendMode(BLEND)
  background(0, 0, 0)
  blendMode(ADD)
  translate(width / 2, height / 2)
  fill(239, 80, 100, 189)
  10.times do |i|
    angle = TWO_PI / 10 * i
    x = cos(angle) * 37
    y = sin(angle) * 37
    circle(x, y, 91 + (i.even? ? 30 :
-30))
  end
end
```



- コードで作っている意味を出したい
- コードも見せたい、面白いと思ってもらいたい

- 美しいと感じるコード や 読みづらいと感じるコード
- TRICK “奇妙なコード”を楽しむ文化
- ライブコーディングするジェネ系VJ

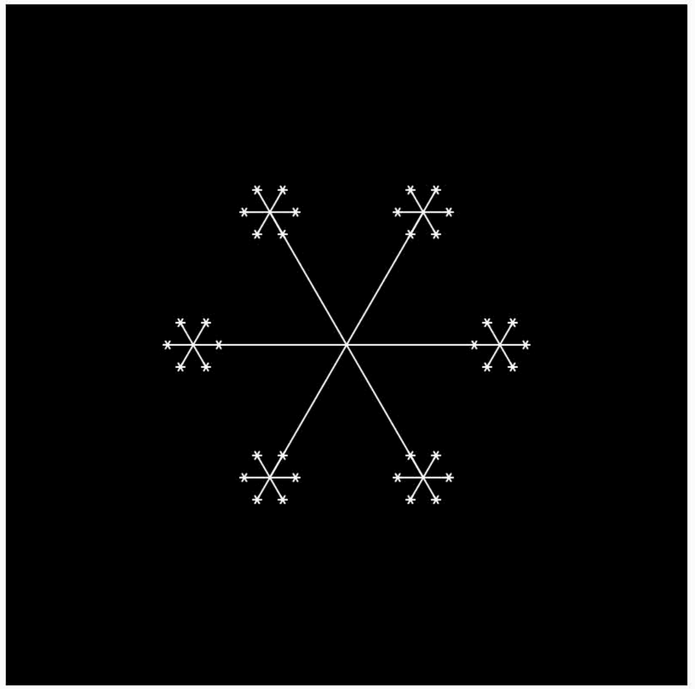
コードでできてるならコードを見たいと思いませんか？

- Web公開：完成作品のコードは構造が伝わりにくい
- 静的展示：コードとビジュアルを並べても注目されにくい
- 登壇発表：コード→実行結果の順だと反応が良い

動的なプロセスが重要では？

試行錯誤 - コードをどう見せるか

```
function setup() {
  createCanvas(400, 400); // キャンバスの作成
}

function draw() {
  background(240); // 背景の設定
  translate(width / 2, height / 2); // キャンバスの中心に原点を移動
  snowflake( - 90 + , - 6 + );
}

// 雪の結晶を描く
function snowflake(len, division) {
  if (len > 2) {  // 再帰的に描く長さが2より大きい場合
    for (let i = 0; i < 6; i++) { // 6つの辺を描く（正六角形）
      line(0, 0, len, 0);
      push(); // 現在の描画状態を保存
      translate(len, 0); // 線の終わりに移動
      rotate(PI / 3); // 60度回転
      snowflake(len / division, division); // 再帰的に雪の結晶を描く
      pop(); // 描画状態を元に戻す
      rotate(PI / 3); // 次の辺に向けて回転
    }
  }
}
```



Web版プロトタイプ

✅ 値を制御できる　❌ 物理的な触感がない

✅ リアルタイム変化　❌コードに触れている実感が薄い

NaClさんのスポンサーブース

- ロータリーエンコーダーを使ったキーボード
- 物理的に触れる → 光・液晶が即座に反応
- 「これだ」と確信

- センサーでコードに「触れる」
- リアルタイムに視覚的フィードバック
- コード実行で作品が生まれることを感じる
- 手を加えることで自分のものにできる

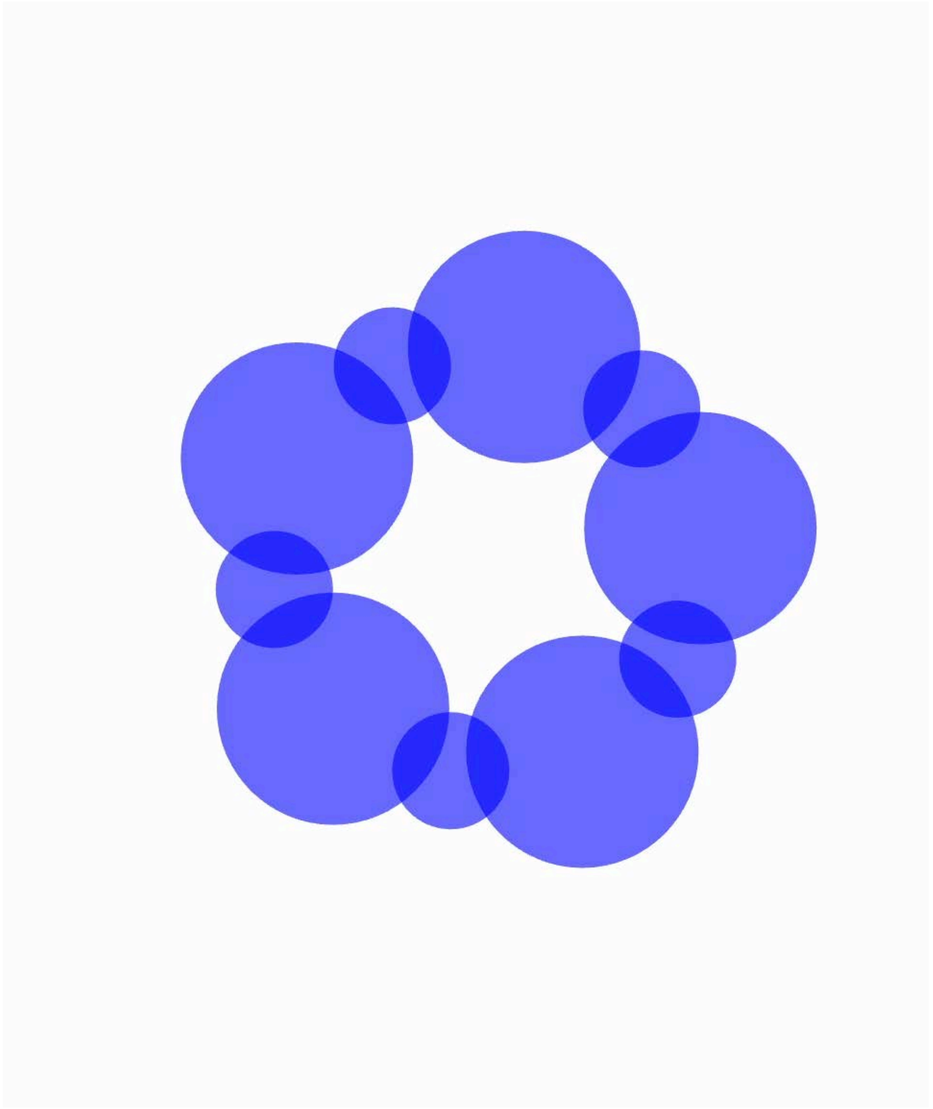
『コードに触れる体験』を感覚的で、楽しいものに

Arduino版制作と検証

- 目標
  - すべてRubyで実装
- 現実的な課題
  - 電子工作初心者
  - 卒業制作の時間制約
  - PicoRubyの情報がまだ少ない

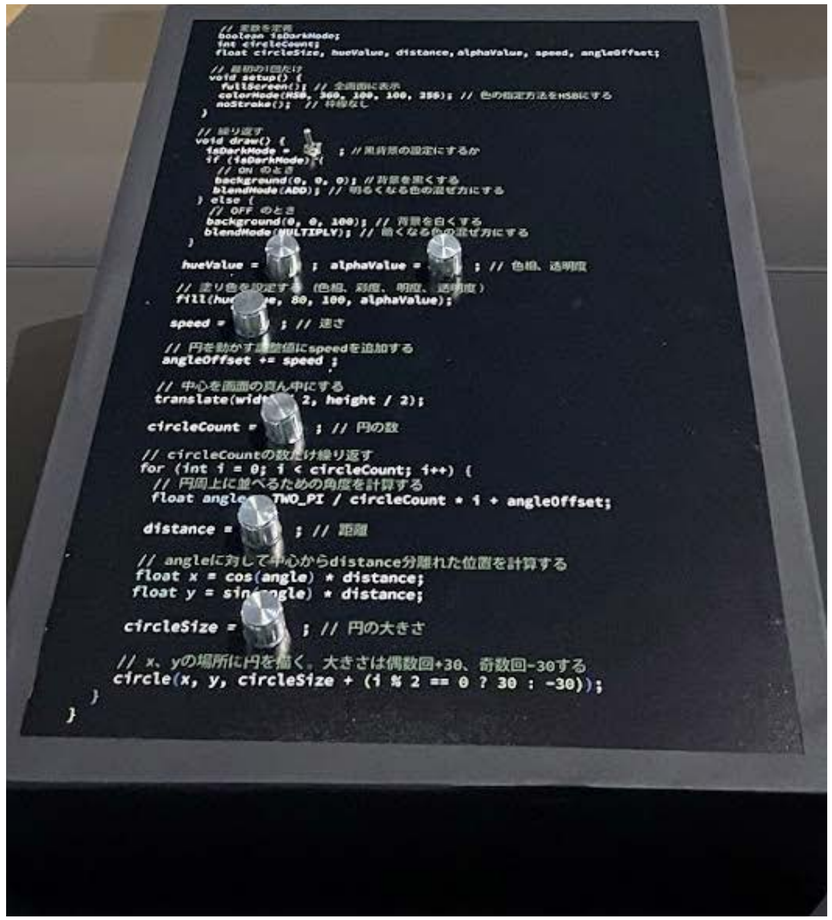
- まずArduino + Processingで制作
  - 情報が豊富→時間内に完成させられる
- Arduino版で完成と手応えを確認 → Ruby版に挑戦

- 基本形
  - 2種類の円が中央を中心に回転
- センサーで様々なパラメータを変化
  - 色、明るさ、大きさ
  - 回転速度、中心からの距離、円の個数

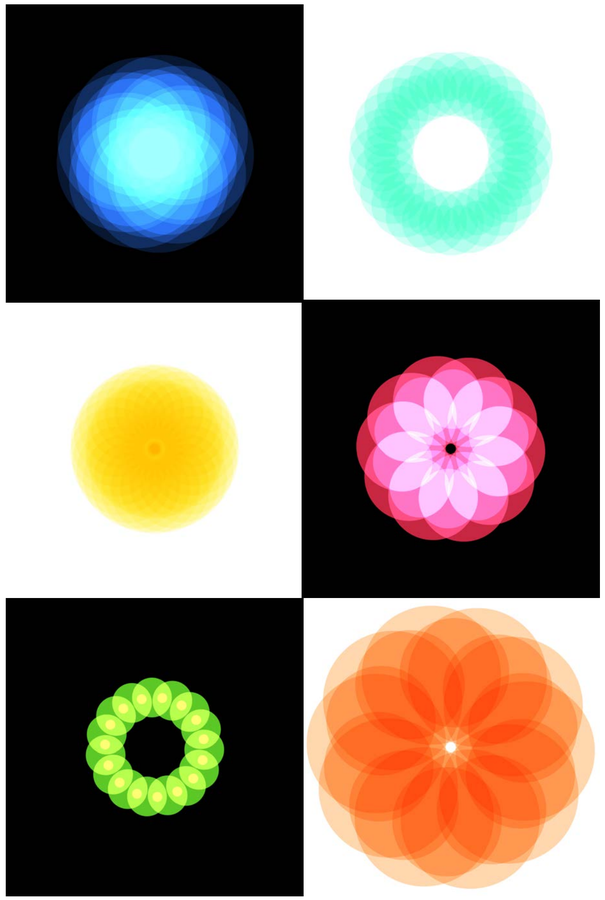


- 人に鑑賞してもらうための3つの設計
  - 設計1:パネルにコードを印刷し、センサーを埋め込む
  - 設計2:コードはシンプルに、変化は大きく
  - 設計3:常に動いている状態

- コードと変数を物理的に一体化
  - コードを印刷したパネルにセンサーを埋め込む
  - 変数を操作している実感
- 物理センサーの価値
  - 手応え、重さ、カチッという感触
  - 直感的な操作性
- 期待する効果
  - コードに触れる、変数を操作する感覚



- コードはシンプルに
  - 読む気の起きる量
  - 複雑すぎると読む前に諦める
- 変化は大きく
  - パラメータ変更で視覚的な大きな変化
  - 複数パラメータで多様な表現
- 期待する効果
  - 「ちょっと触ってみよう」から「もっと試してみたい」へ

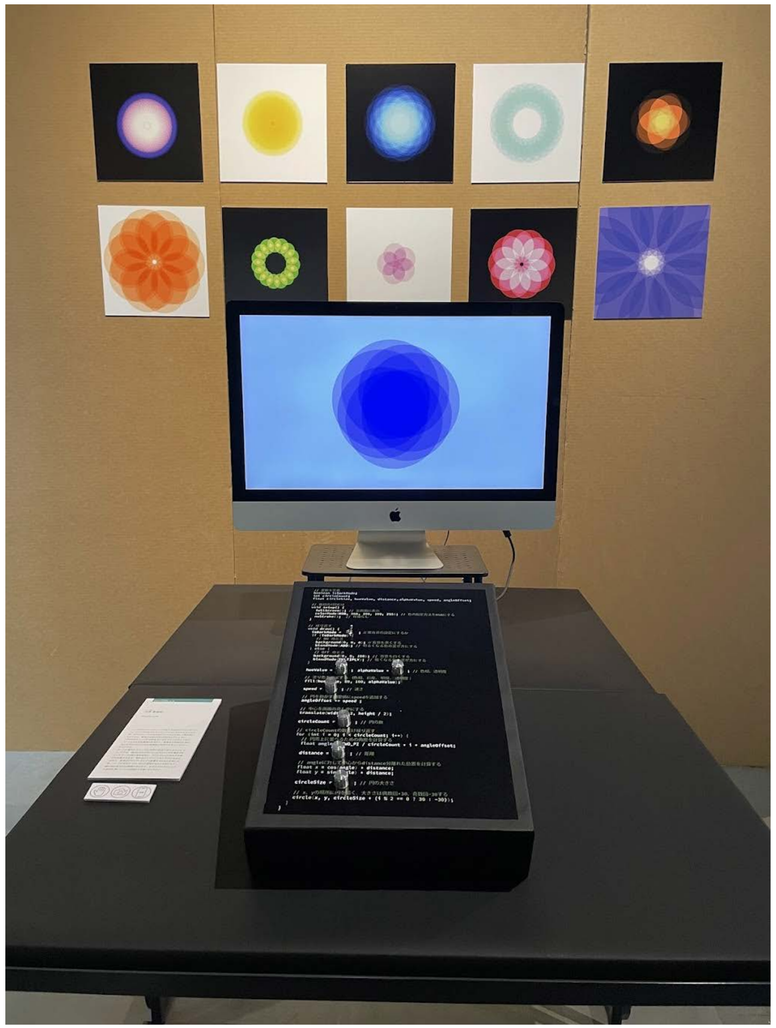


- 展示では「最初の一歩」が最大のハードル
  - 体験開始を自然に誘導したい
- 待機中も回転し続ける
  - 動いている → 注意を引く
  - 止まっていない → いきなり動き出さない
  - リセット不要 → すぐに始められる
- 期待する効果
  - 「触ってみよう」への自然な誘導

2025年3月 武蔵野美術大学 卒業制作展

- 来場者：未就学児～70代
- 観察方法
  - 滞在時間の記録
  - 行動パターンの観察
  - 鑑賞者との会話

3つの設計それぞれを検証



✅ 成功した点

- 「えっどうなってるの？」という驚きの反応
- アート作品としての存在感
- 手応え、重さ、感触への好反応

❌ 課題

- 視線が手元（パネル）と前方（画面）に分散
- 仕組みが見えず、コードが動いている実感が持ちにくい

学び：アート体験には効果的、学習体験には視線の統合が必要

✅ 成功した点

- ほとんどの人が3分以上滞在
- 短時間の試行から、深い探索まで多様な関わり方
- 「組み合わせたらどうなる？」という探索行動

❌ 課題（講評より）

- センサーの種類が少ない
- 変数とセンサーの対応が直感的でない
- センサー値の変化が分かりにくい

学び：シンプルなコードでも十分な探索時間を生む
パラメータの種類と操作方法の対応が重要

✅ 成功した点

- 多くの人が立ち止まる
- 「触ってみよう」への自然な誘導
- リピート体験

❌ 課題

- 長時間稼働での安定性
- 再起動の仕組みが必要

学び：待機状態の設計が体験開始のハードルを下げる
予想以上に繰り返し体験される作品になった

- 発見1：コードより「仕組み」への関心が予想以上に高い
  - センサー→ビジュアルの流れを理解したい
  - 「どういう仕組み？」という質問
- 発見2：自分好みの状態を作れる楽しさ
  - 心地よい組み合わせで止められる「ずっと見ていられる」
- 学び
  - 展示して初めて多様な楽しみ方が見えた

✅成功した点

- 常に動く → 体験開始のハードル低減
- シンプルなコード → 読む気を起こす
- 物理センサー → 身体性のある体験

❌改善が必要な点

- 視線の分散 → 学習には統合が必要
- センサーと変数の対応 → より直感的な選択
- 方向性の明確化 → アートか教育か

講評での指摘：「誰に、どういう体験をしてほしいのか」

次への方針：学習体験に重点を置き、Ruby版での設計へ

Ruby版の実装

- Rubyが好きだから
  - 本当にRubyだけでできるか確かめたかった
- 一つの言語で完結することの価値
  - センサー制御〜ビジュアル生成まで
  - 言語を切り替えずに開発・学習できる

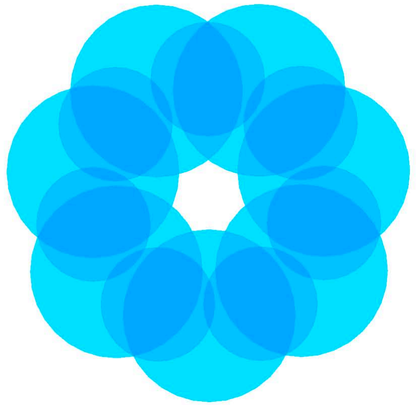
- Arduino版での学び
  - 視線の統合が必要
  - センサーと変数の対応（より直感的な選択）
  - 方向性：学習体験に重点

- Ruby版の方針転換
  - コード表示に表示
  - ロータリーエンコーダ採用（色相環に沿って変化）
  - センサー値を画面に表示（数値の変化を可視化）

物理センサーは維持。手応え、感触の重要性

コード・変数の値・ビジュアルを同時に表示

```
require "processing"
using Processing

def setup
  @circle_count = 4; @distance = 100; @circle_size = 150
  @hue_value = 200; @is_dark_mode = false
  @angle_offset = 0; @is_button_push = false

  size(displayWidth, displayHeight)
  colorMode(HSB, 360, 100, 100, 255)
  noStroke
end

def draw
  # 背景色と色の混ぜ方を設定
  @is_dark_mode = false
  if(@is_dark_mode)
    background(0, 0, 0)
    blendMode(ADD)
  else
    background(0, 0, 100)
    blendMode(MULTIPLY)
  end

  # 色を設定
  @hue_value = 190
  fill(@hue_value, 80, 100, 150)

  # 回転角度を設定
  @is_button_push = false
  @angle_offset += @is_button_push ? 0.05 : 0.01

  # 円の個数を設定
  @circle_count = 14
  @circle_count.times do |i|
    angle = TWO_PI / @circle_count * i + @angle_offset
    # 中心からの距離を設定
    @distance = 122
    x = cos(angle) * @distance
    y = sin(angle) * @distance
    # 円のサイズを設定
    @circle_size = 148
    # 円を描く
    circle(x, y, @circle_size + (i.even? ? 30 : -30))
  end
end
```



センサーを操作すると値が変化する

すべてRubyで構成

【物理層】センサー（スイッチ・可変抵抗）

↓ GPIO

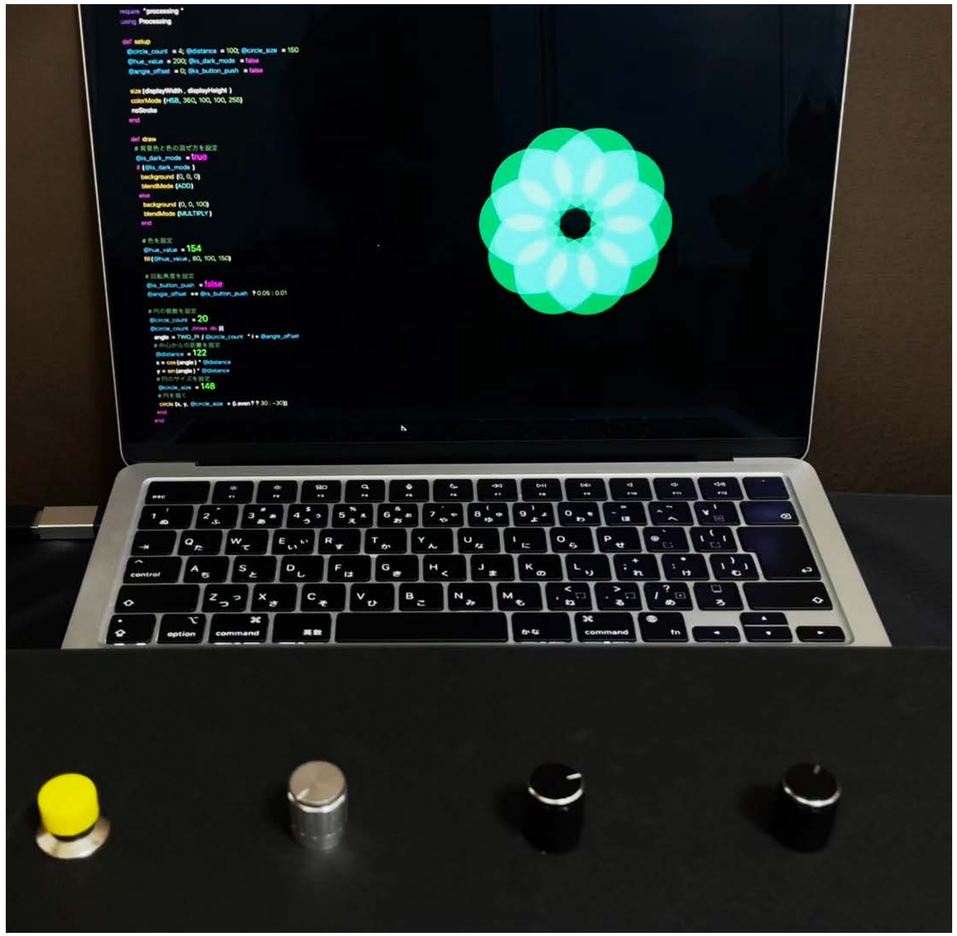
【制御層】PicoRuby（Raspberry Pi Pico）

↓ シリアル通信

【表現層】processing gem（PC）

↓

リアルタイムビジュアル



すべてRubyで構成

【物理層】センサー（スイッチ・可変抵抗）

↓ GPIO

【制御層】PicoRuby（Raspberry Pi Pico）

↓ シリアル通信

【表現層】processing gem（PC）

↓

リアルタイムビジュアル

- スイッチ：ボタンを押す→ON/OFF
- 可変抵抗：つまみを回す→0 〜 4095の値
- ロータリーエンコーダー：回し続けられる
  →値が増減

スイッチ

可変抵抗

ロータリー
エンコーダー

すべてRubyで構成

【物理層】センサー（可変抵抗・スイッチ）

↓ GPIO

【制御層】PicoRuby（Raspberry Pi Pico）

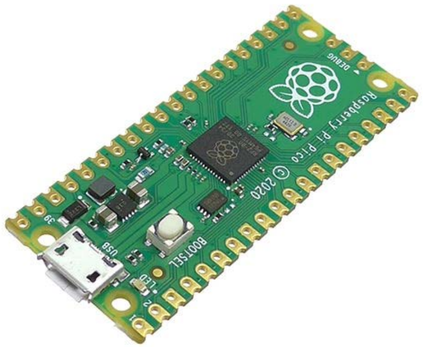
↓ シリアル通信

【表現層】processing gem（PC）

↓

リアルタイムビジュアル

- マイコンでセンサー値を読み取り
- シリアル通信（USB経由）



すべてRubyで構成

【物理層】センサー（可変抵抗・スイッチ）

↓ GPIO

【制御層】PicoRuby（Raspberry Pi Pico）

↓ シリアル通信

【表現層】processing gem（PC）

↓

リアルタイムビジュアル

- センサー値→ビジュアルのパラメータ

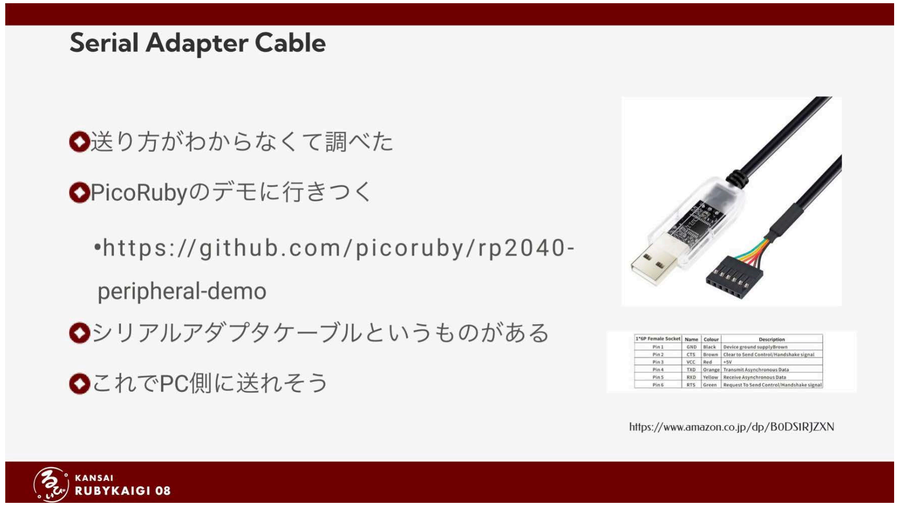
- 実装の過程で2つの大きな壁
  - 1. 情報不足
    - PicoRubyでどう実装する？
  - 2. 予期しない動作
    - Arduino版では起きなかった問題

課題

- PicoRubyでどう実装するか
- サンプルコードが少ない
- PCへのデータ転送方法が不明

解決アプローチ

- 公式ドキュメント、技術同人誌、GitHubを調査
- UART（シリアル通信）での実装方法を見つけた
- 試行錯誤しながら実装



問題

- ビジュアルがカクつく

原因

- センサー値が安定しない → ノイズが混入している

変更前：値をそのまま使用

```
value = adc.read
uart.write("P,#{value}\n")
```

変更後：複数回読み取って平均化

```
readings << adc.read
readings.shift if readings.length > 5
filtered = readings.sum / readings.length
uart.write("P,#{filtered}\n")
```

- 結果：カクつきが改善、安定した動作
- 学び：センサー値のノイズ対策の重要性

基本的な実装の流れ

1. ノイズ対策：移動平均フィルタ

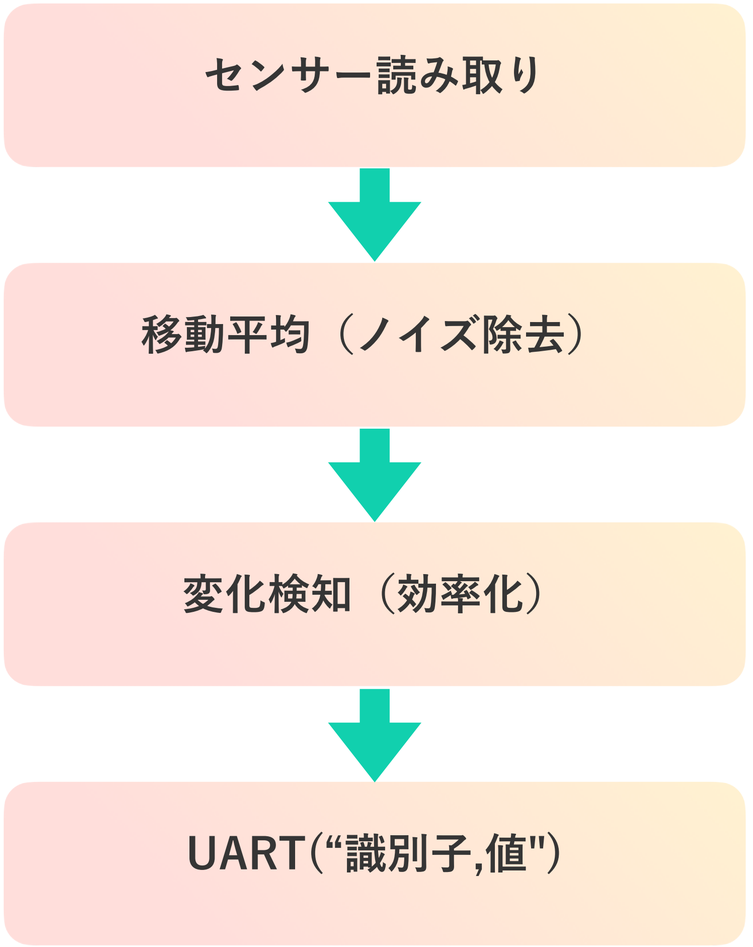
- 5回読み取って配列に格納、平均値を計算

2. 変化検知＋効率的な送信

- 閾値を超えた変化があった時だけ送信

3. 通信プロトコルの設計

- "識別子,値\n" 形式を採用
- 複数センサーの識別が容易



```ruby
# 初期化
pot = ADC.new(26)
uart = UART.new(unit: :RP2040_UART0, ...)
pot_readings = []

loop do
  # ノイズ対策：移動平均
  pot_readings << pot.read_raw
  pot_readings.shift if pot_readings.length > 5
  filtered = pot_readings.sum / pot_readings.length

  # 変化検知して送信
  if (filtered - last_value).abs > 50
    uart.write("P,#{filtered}\n")
  end
end
```

見慣れたRubyの書き方

- 配列操作（<<、shift、each）
- オブジェクト生成
- シリアル通信

組み込み開発も、いつものRuby

完全版: https://github.com/ksbmyk/sketch/blob/main/tmp/rwc2025/pico.rb

```ruby
# 初期設定
def setup
  @uart = UART.open(port, 115200)
  @circle_size = 150
  size(1000, 1000)
end
# 描画
def draw
  handle_serial_data
  circle(width/2, height/2, @circle_size)
end
# データ受信
def handle_serial_data
  data = @uart.gets
  return if data.nil?
  values = data.chomp.split(',')
  if values[0] == "P"
    @circle_size = map(values[1].to_i, 0, 4095, 10, 500)
  end
end
```

見慣れたRubyの書き方

- メソッド定義
- インスタンス変数
- 文字列操作（chomp、split）

ビジュアル生成も、いつものRuby

	Arduino + Processing版	Ruby版
制御層	Arduino (C++)	PicoRuby (Ruby)
表現層	Processing (Java)	processing gem (Ruby)
思考の切替	必要	不要

展開と価値

物理版の成功と課題

✅ センサーで触れる身体性

✅ 展示で手応え確認

❌ 会場に来た人にしか届かない

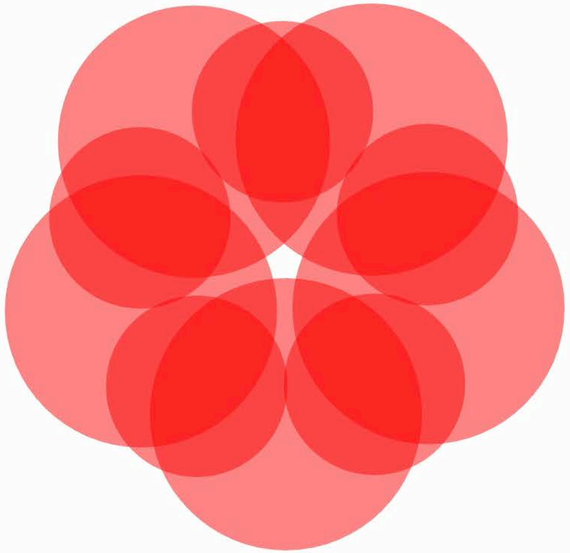
❌ 特別なハードウェアが必要

学びを活かしたWeb版

- ruby.wasmで実装
- 手元のデバイスで体験可能

Web版デモ

```
$angle_offset = 0
def setup
  createCanvas(windowWidth, windowWidth)
  colorMode(HSB, 360, 100, 100, 255)
  noStroke
end

def draw
  blendMode(BLEND)
  $is_dark_mode = false
  if ($is_dark_mode)
    background(0, 0, 0)
    blendMode(ADD)
  else
    background(0, 0, 100)
    blendMode(MULTIPLY)
  end

  translate(width / 2, height / 2)
  $hue_value =   0
  $alpha_value = 150
  fill($hue_value, 80, 100, $alpha_value)
  $speed = 0.06
  $angle_offset += $speed

  $circle_count =  10
  $circle_count.times do |i|
    angle = TWO_PI / $circle_count * i + $angle_offset
    $distance = 100
    x = cos(angle) * $distance
    y = sin(angle) * $distance
    $circle_size = 150
    circle(x, y, $circle_size + (i.even? ? 30 : -30))
  end
end
```



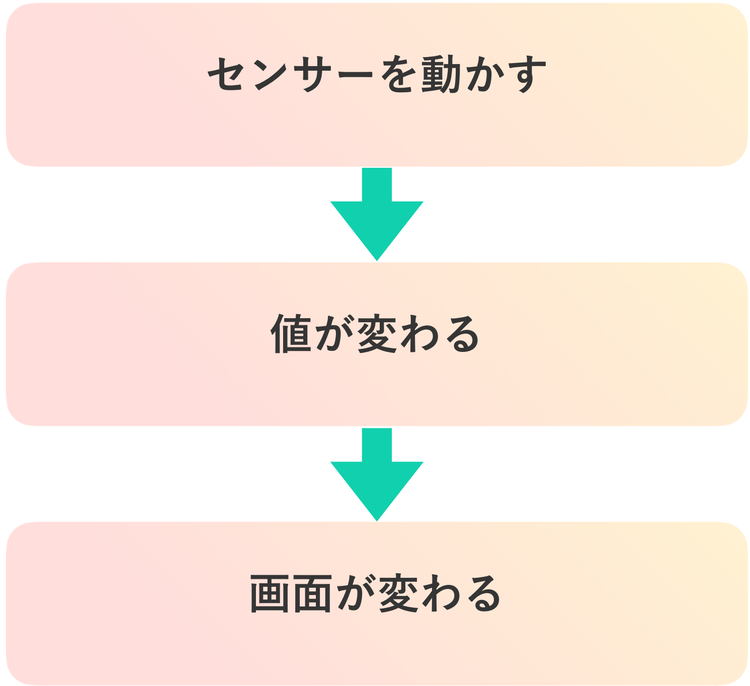
https://ksbmyk.github.io/sketch/events/tangible_code.html

- Webプロトタイプ（冒頭）
  - 値を制御できリアルタイム変化があるが、触れている実感が薄い
- 物理版での学び
  - センサーの身体性、直接触れることの重要性
- 今回のWeb版
  - コードの見せ方やビジュアルを改善したが、マウスの限界は残る
- 新しい仮説
  - タッチパネルなら解決できる？

- 検証への協力依頼
  - スマホ・タブレットで
  - 画面を直接タッチ
- 確認したいこと
  - コードを変えている感覚、コード理解の助けになるか
  - 体験そのものの感想

いつでも歓迎。今すぐでも、後日でも

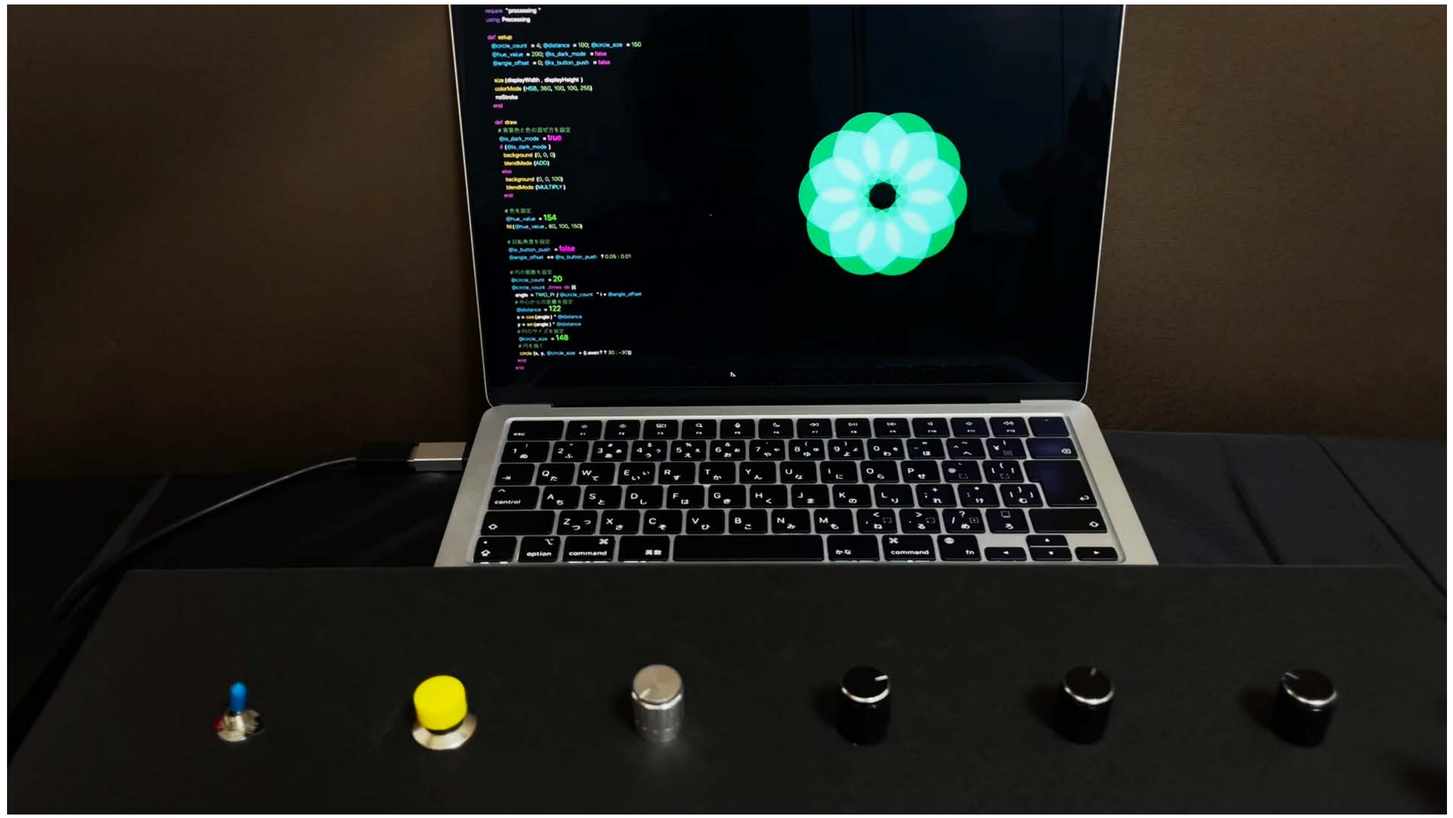
- 身体性
  - 抽象的なコードを身近に感じられる
- 因果関係の理解
  - 変えると何が起きるかが分かる
- 自発的な興味
  - コードの仕組みを知りたくなる
- 探索の楽しさ
  - すぐに結果が見え、何度でも試したくなる



- コードを理解する時
  - 変える（パラメータ・条件・構造）
  - 実行する
  - 確認する
- Tangible Code
  - 変える → センサーを操作
  - 実行する → リアルタイムに反映
  - 確認する → ビジュアルで即座に確認

環境構築やエラーなしに、この体験ができる

これは、一つの方法



あなたならコードをどう見せますか？