

MITIGATING AI MISUSE IN INTRODUCTORY PYTHON COURSES WITH GRAPHICAL PROGRAMMING TASKS

Kiwi Pycon 2025

INTRODUCTION

INTRODUCTION > ME

- Graphic designer → web designer → web designer & developer → multimedia developer →
- Lecturer in Creative Technologies / Interaction Design
- Active in open-source development and author of a Python book
- Increasingly grappling with what to teach and how to teach it ...

INTRODUCTION > PROCESSING

Graphics library and IDE designed for teaching non-programmers programming in a visual way

... anyone heard of it?

Table of contents

Overview

Processing Development Environment (PDE)

Preferences

Sketches and Sketchbook

Renderers

Coordinates

Tabs, Multiple Files, and Classes

Debug

Programming Styles

Adding Libraries, Tools, and Modes

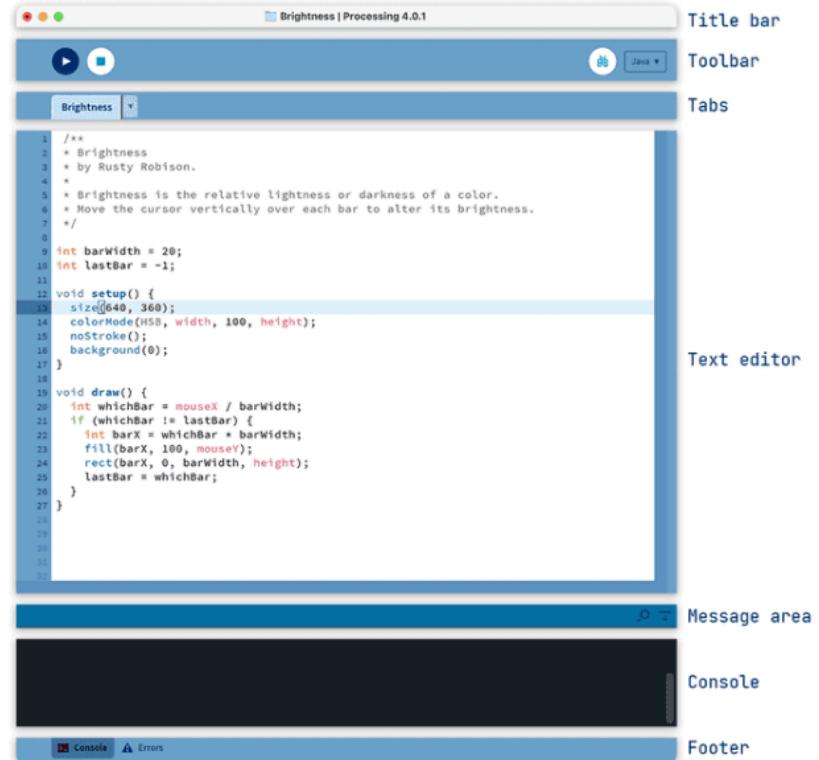
Export

Command Line Interface (CLI)

The Processing Development Environment (PDE) consists of a simple text editor for writing code, a message area, a text console, tabs for managing files, a toolbar with buttons for common actions, and a series of menus. The menu options change from mode to mode. The default Java mode is documented here.



Display window



Programs written using Processing are called sketches. These sketches are written in the text editor. It has features for cutting/pasting and searching/replacing text. The message area gives feedback while saving and exporting and also displays errors. The console displays text output by Processing sketches including complete error messages and text output from sketches with the `print()` and `println()` functions. (Note that the console works well for occasional messages, but is not intended for high-speed, real-time output.)

The buttons on the toolbar can run and stop programs,



Run

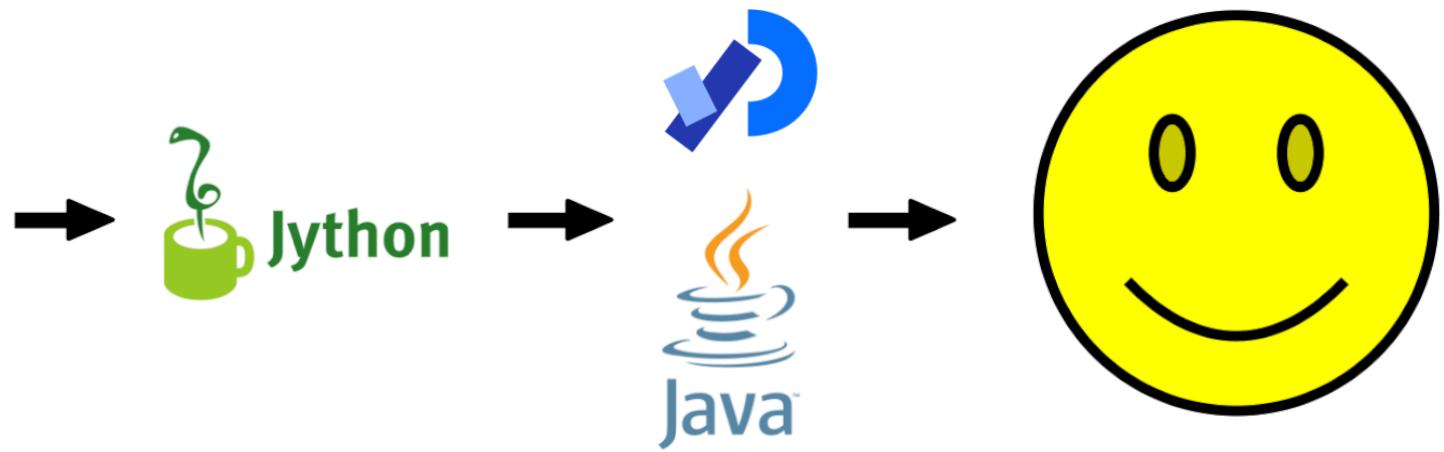
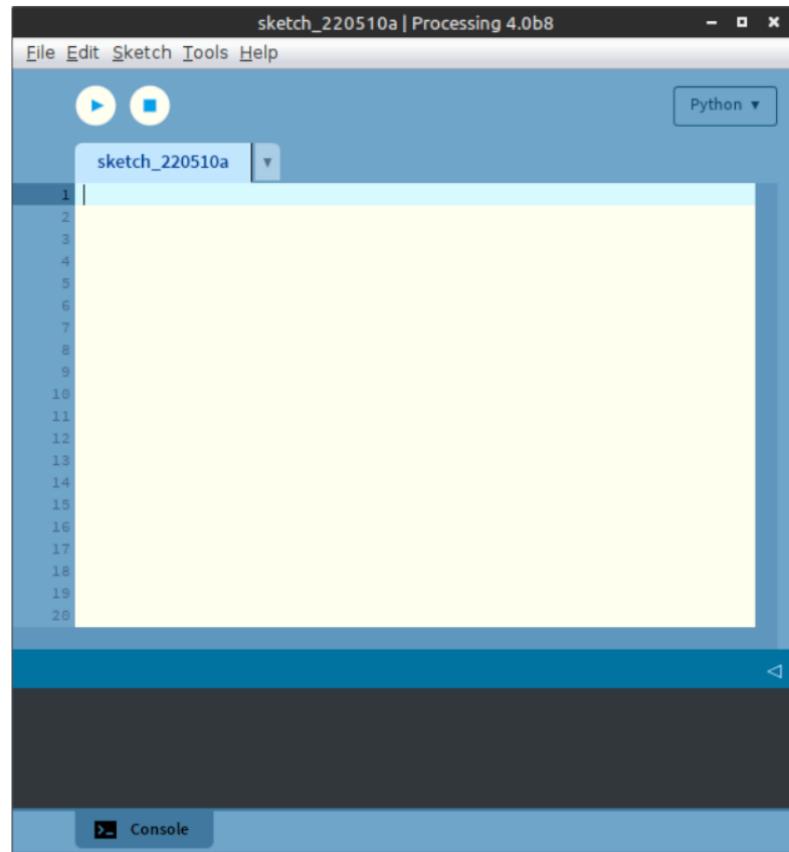
Runs the sketch. In Java mode, it compiles the code and opens a new display window.

INTRODUCTION > PROCESSING

- Easy to learn, especially for beginners; promotes software literacy within the visual arts
- Great for coding animations, generative art, and data visualisation
- Based on Java, with simple functions for drawing and motion

INTRODUCTION > PROCESSING DERIVATIVES

- `p5.js` -- JavaScript version for the web (runs in browsers)
- `JRubyArt` -- wrapper for Processing in Ruby
- `Processing Android` -- run Processing sketches on Android devices
- ... several others targeting different languages ...
- `Processing.py` -- Python mode for Processing (Python 2/Jython)



INTRODUCTION > JYTHON LIMITATIONS

- No support for Python 3
- Limited CPython C-extension support (e.g., no NumPy)
- However, Processing.py is effectively discontinued (incompatible with Processing 4)

THONNY - PY5MODE

THONNY - PY5MODE > MOTIVATION

A new **Python 3** environment that emulates the Processing IDE, serving as a successor to Processing.py.

It combines ...

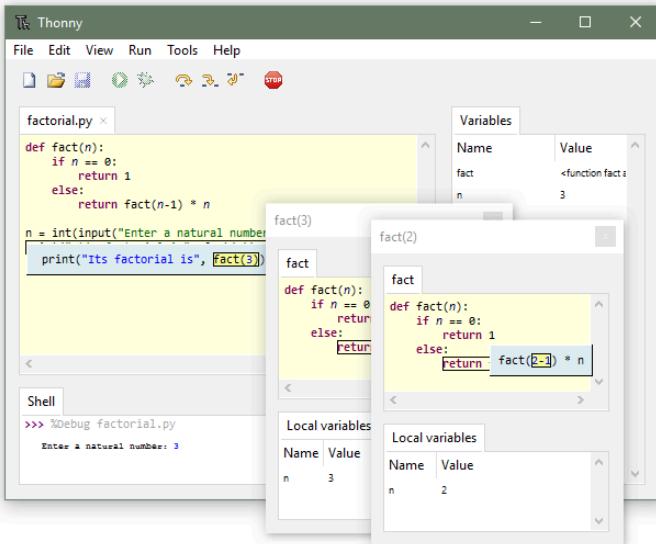
Thonny 4 is dedicated to Ukraine fighting the Russian invasion.
Please support Ukraine!

Thonny

Python IDE for beginners

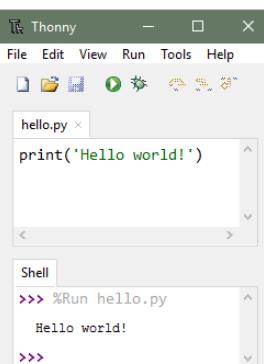


Download version [4.1.7](#) for
Windows • Mac • Linux



Features

Easy to get started. Thonny comes with Python 3.10 built in, so just one simple installer is needed and you're ready to learn programming. (You can also use a separate Python installation, if necessary.) The initial user interface is stripped of all features that may distract beginners.



Basic Example

[Installation](#)
[Getting Started](#)
[Source Code](#)
[Funding](#)
[Get In Touch](#)

Welcome to py5!

py5 is a new version of Processing for Python. The goal of py5 is to create a version of Processing that is [integrated into the Python ecosystem](#). Built into the library are thoughtful choices about how to best get py5 to work with other popular Python libraries and tools such as [Jupyter](#), [numpy](#), [shapely](#), [trimesh](#), [matplotlib](#), and [Pillow](#).

py5 is an excellent choice for educators looking to teach Python in the context of creative coding and is currently used in classrooms all around the world. This website's documentation includes [introductory tutorials](#) as well as extensive [reference documentation](#), complete with example code.

Basic Example

Here is a basic example of a py5 Sketch:

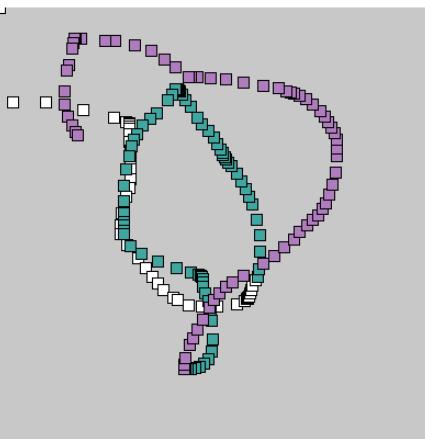
```
def setup():
    size(400, 400)
    rect_mode(CENTER)

def draw():
    square(mouse_x, mouse_y, 10)

def mouse_clicked():
    fill(random_int(255), random_int(255), random_int(255))

run_sketch()
```

Here is an animated GIF to give you an idea of what that Sketch looks like:



Installation

If you have Java 17 installed on your computer, you can install py5 using pip:

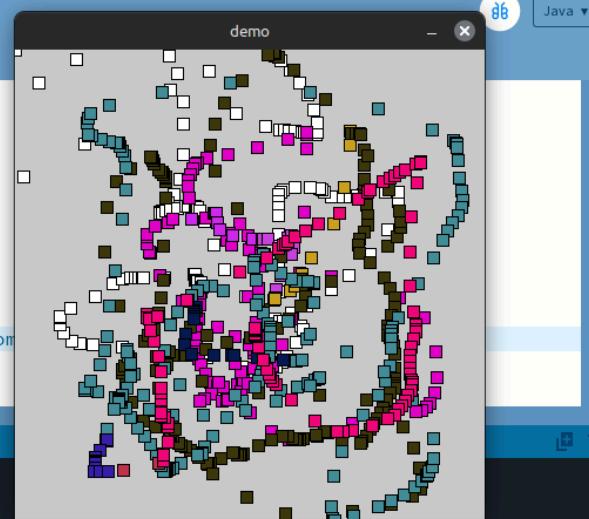
File Edit Sketch Debug Tools Help

demo

```
1 void setup() {  
2     size(400, 400);  
3     rectMode(CENTER);  
4 }  
5  
6 void draw() {  
7     square(mouseX, mouseY, 10);  
8 }  
9  
10 void mouseClicked() {  
11     fill(int(random(255)), int(random(255)), int(random(255)));  
12 }  
13 }
```

Java ▾

Console Errors



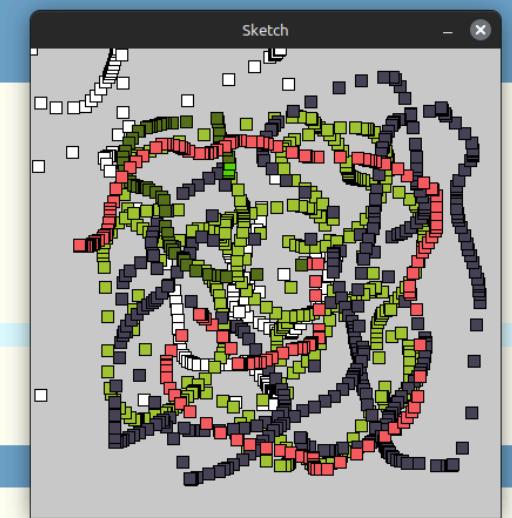
File Edit View Run Tools Help py5

demo.py

```
1 def setup():  
2     size(400, 400)  
3     rect_mode(CENTER)  
4  
5  
6 def draw():  
7     square(mouse_x, mouse_y, 10)  
8  
9  
10 def mouse_clicked():  
11     fill(random_int(255), random_int(255), random_int(255))
```

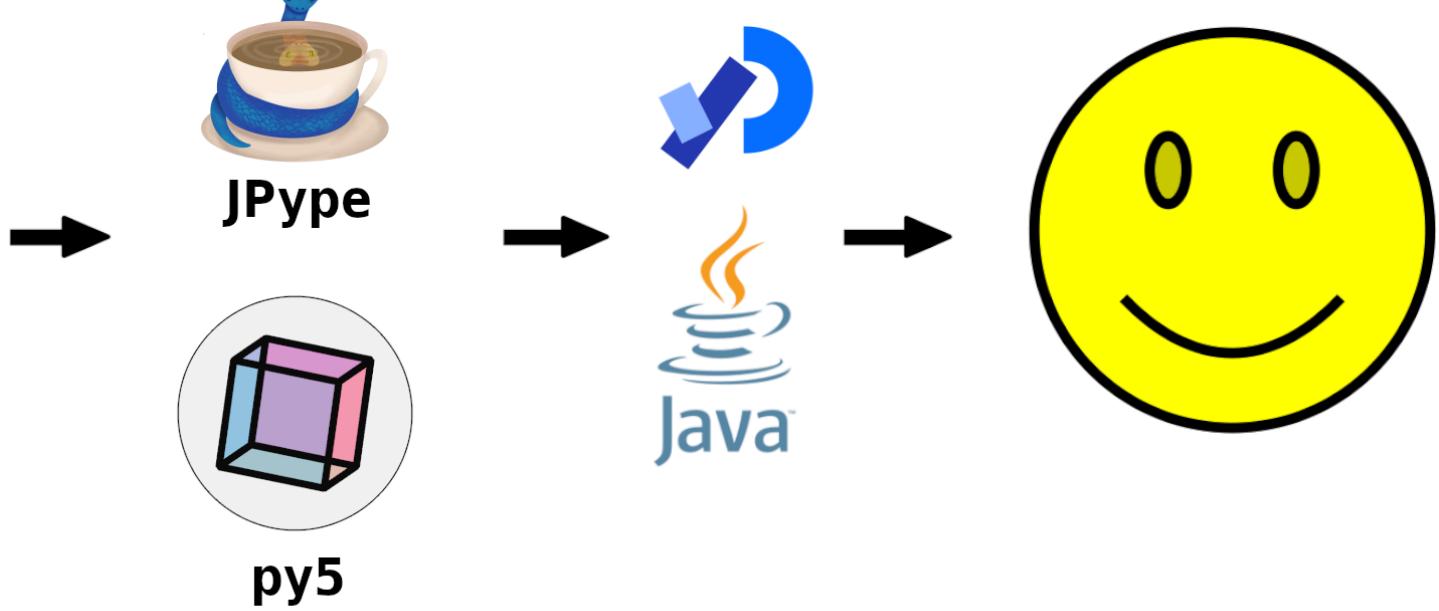
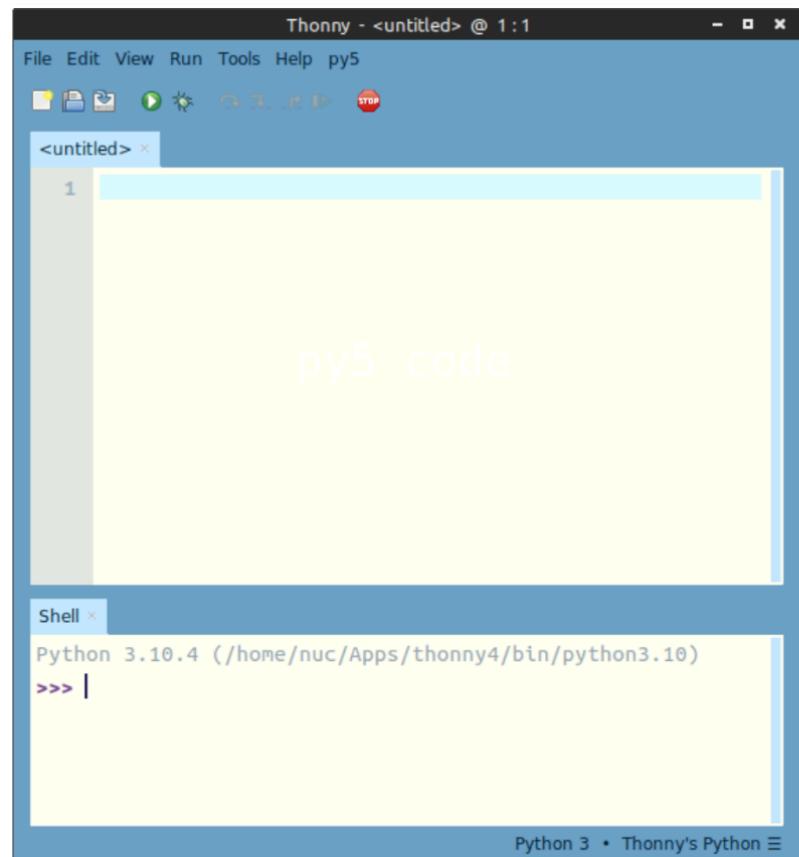
Shell >>>

Local Python 3 • Thonny's Python



THONNY - PY5 MODE > FEATURES

- Leverages the py5 library, which incorporates JPyre
- JPyre supports Python 3
- JPyre supports third-party libraries with C extensions
- More at <https://py5coding.org> (including Jupyter Notebook integration)



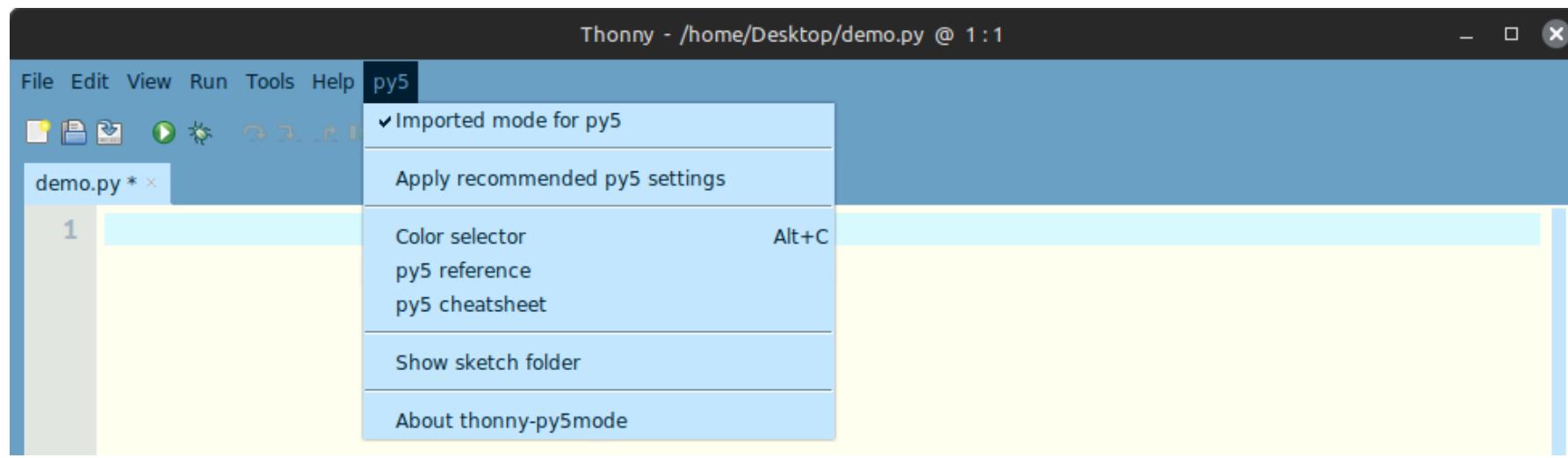
demo.py *

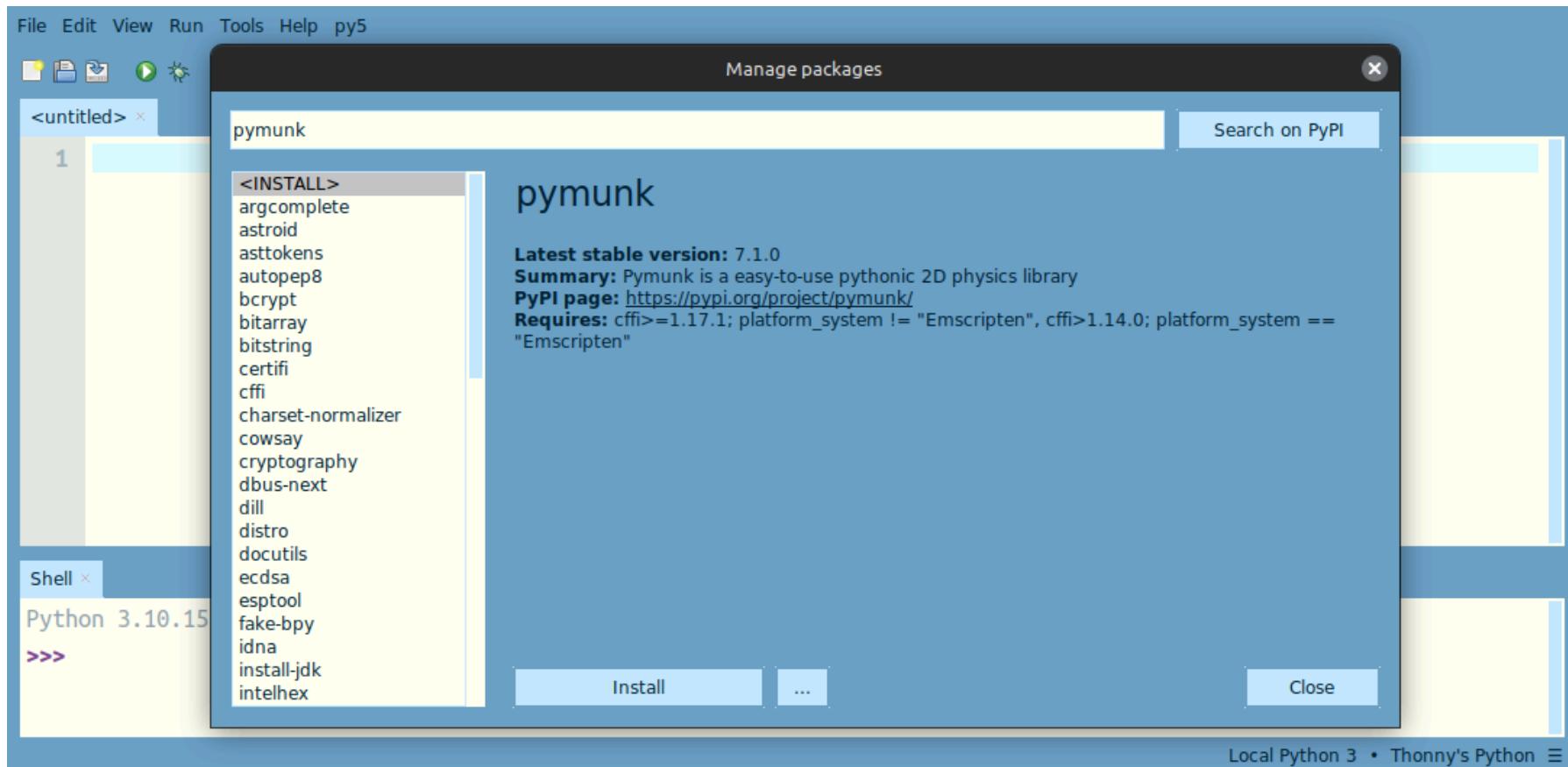
```
1 rect(  
rect(  
rect_mode(  
RECT  
RecursionError
```

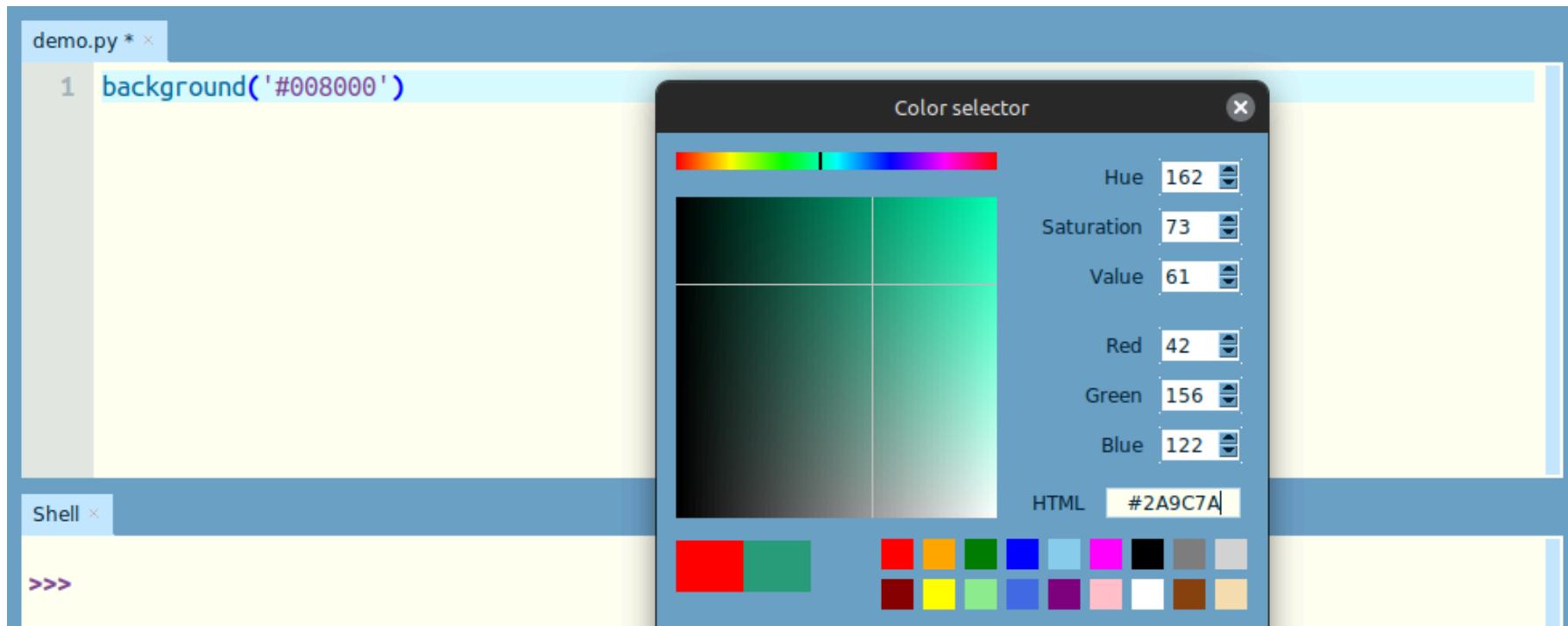
rect(a, b, c, d, /) -> None
rect(a, b, c, d, r, /) -> None
rect(a, b, c, d, tl, tr, br, bl, /) -> None

Draws a rectangle to the screen.

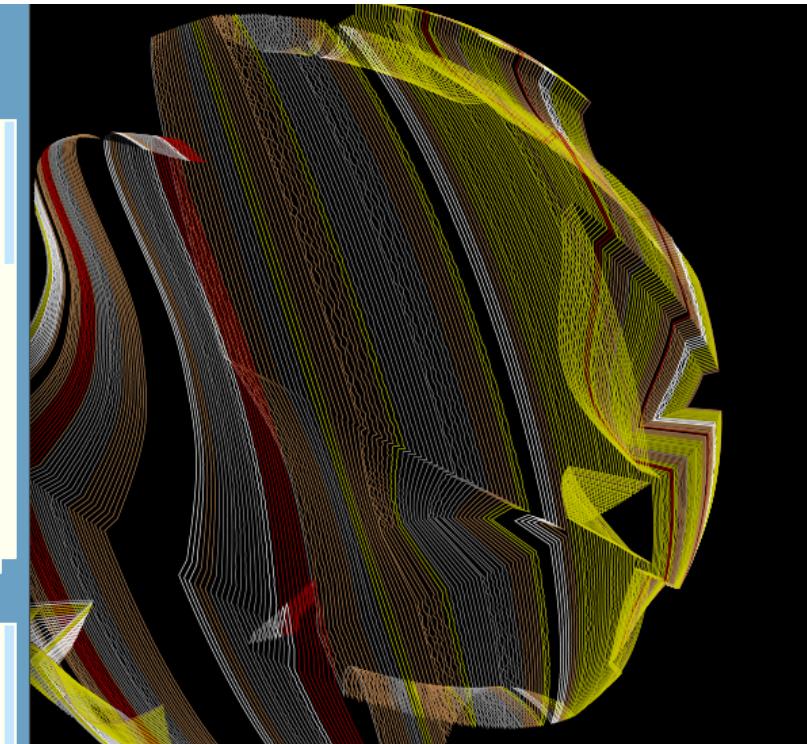
Underlying Processing method: PApplet.rect

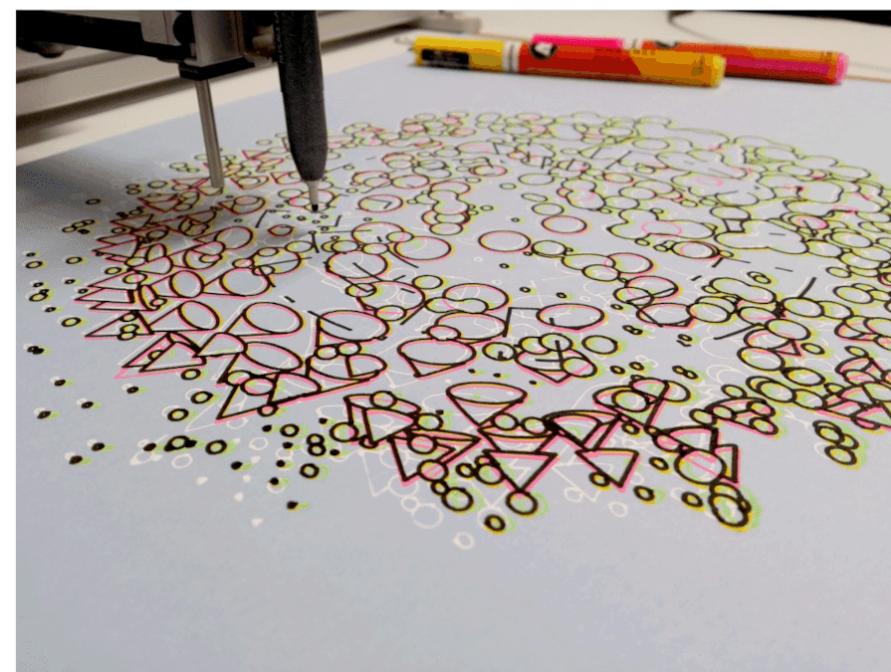
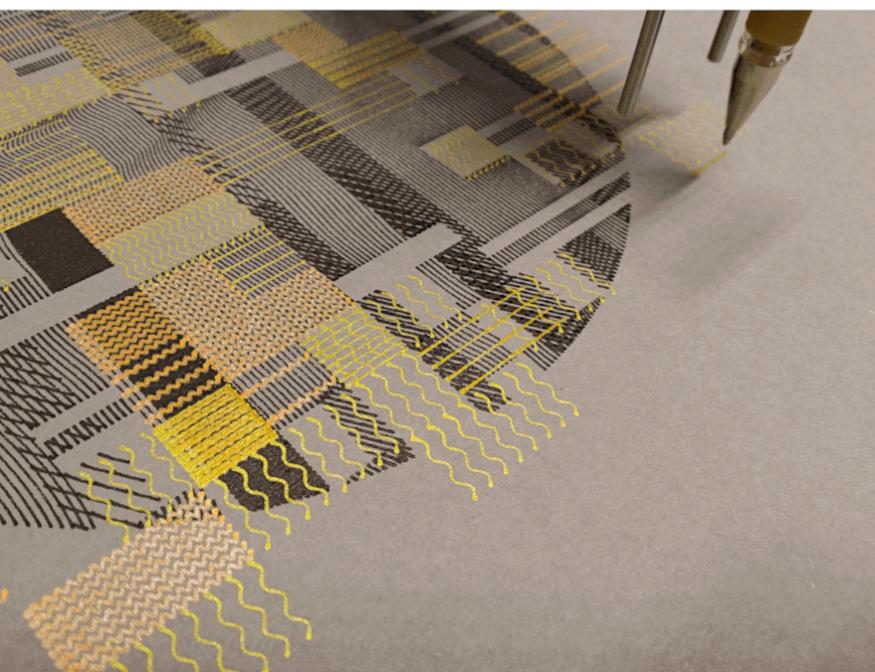
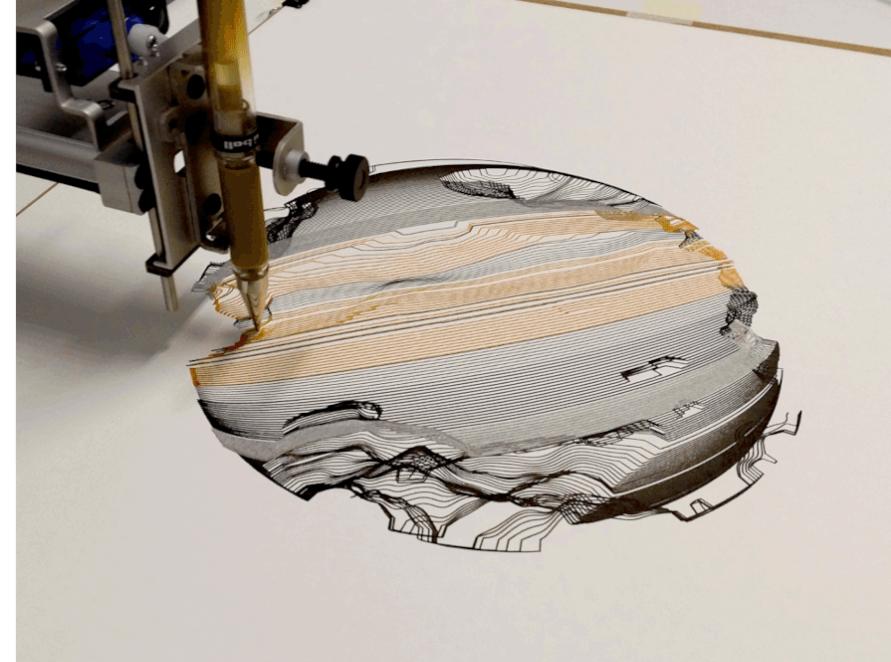
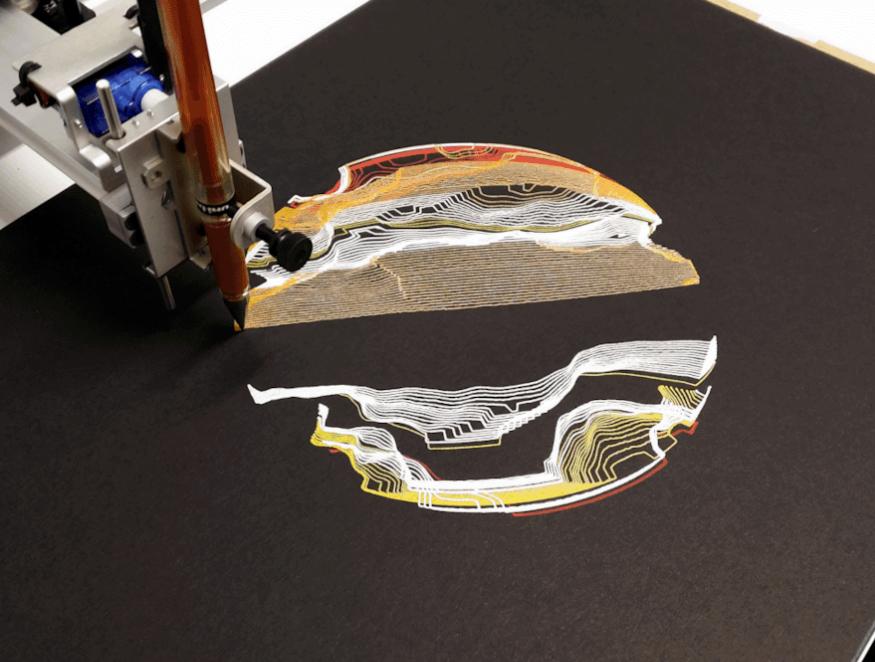


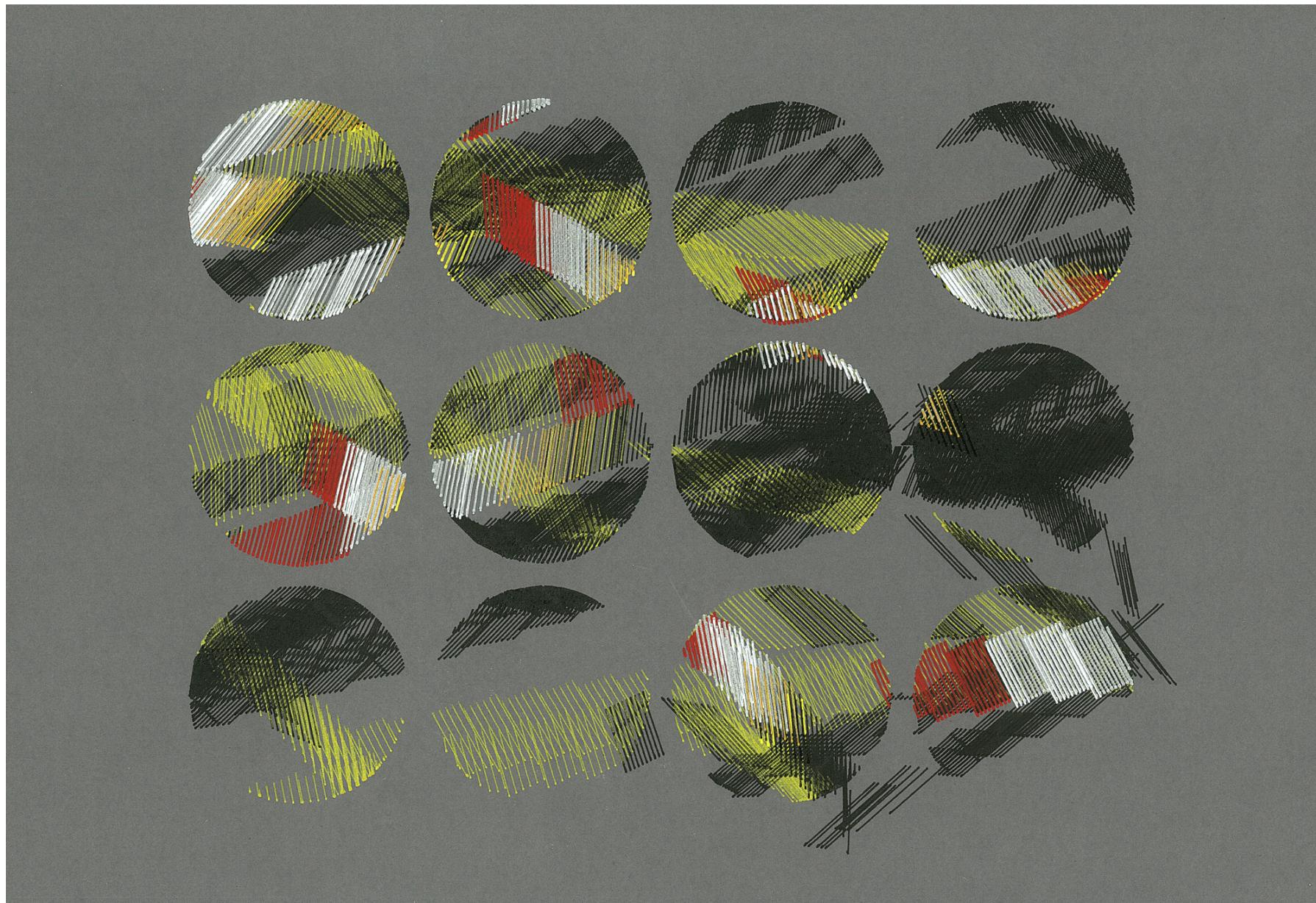


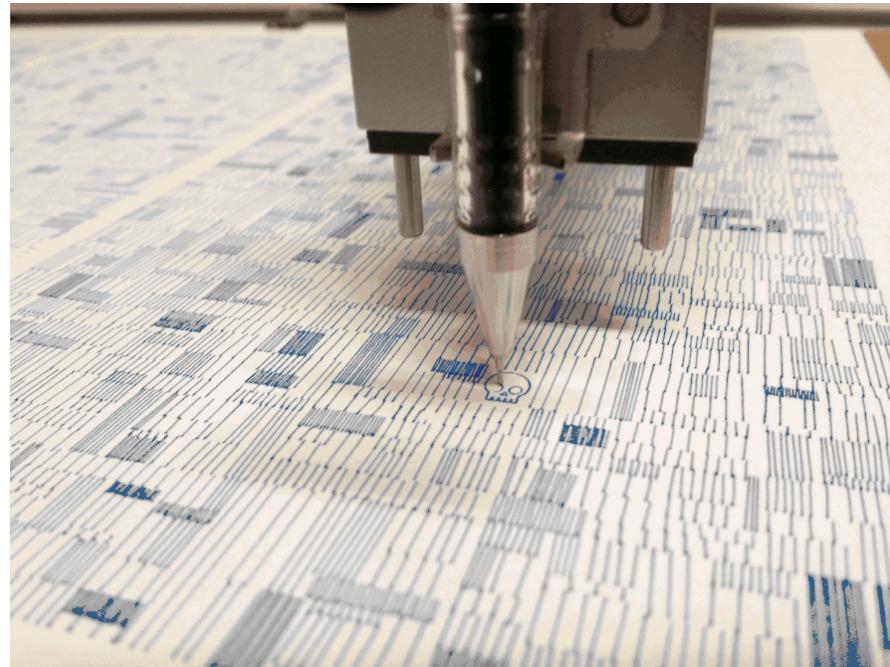
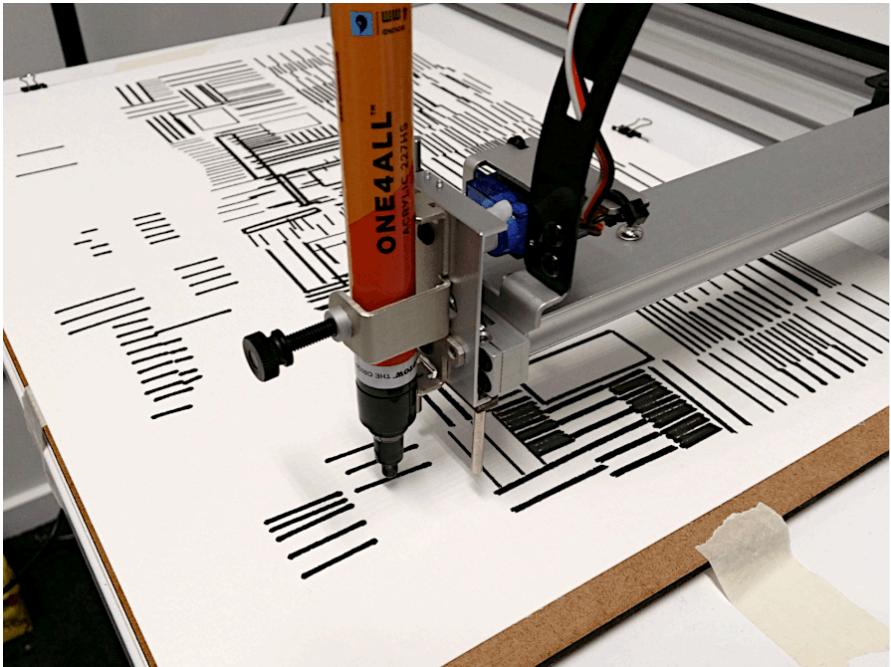


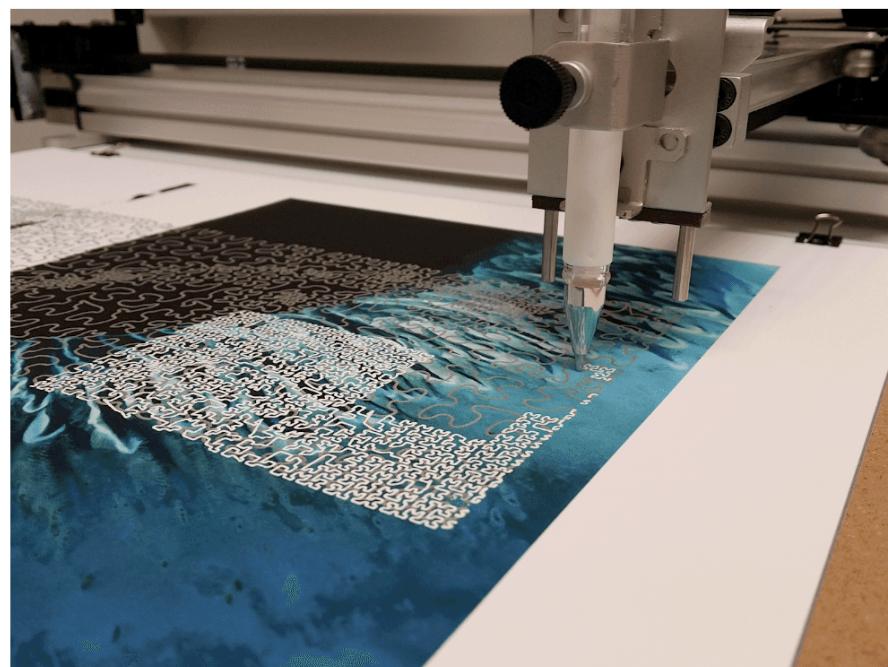
```
File Edit View Run Tools Help py5
06_multicolor_plot.py x
1 # inspired by @yuruyurau
2
3 size(420*4, 297*4, SVG, '06_multicolor_plot.svg')
4 #os_noise_seed(12345 * 42)
5 #random_seed(54321 * 42)
6
7 time = random(123456789)
8 diameter = height * 0.9
9 resolution = 300
10 xy_range = np.linspace(-1, 1, resolution)
11
12 background(0)
13 no_fill()
14 translate(width/2, height/2)
15 rotate(random(PI))
16
17 # you could employ xml.etree to separate colors (pens) onto different lay
<<< https://t.me/dishant_1000 />>>
>>>
```

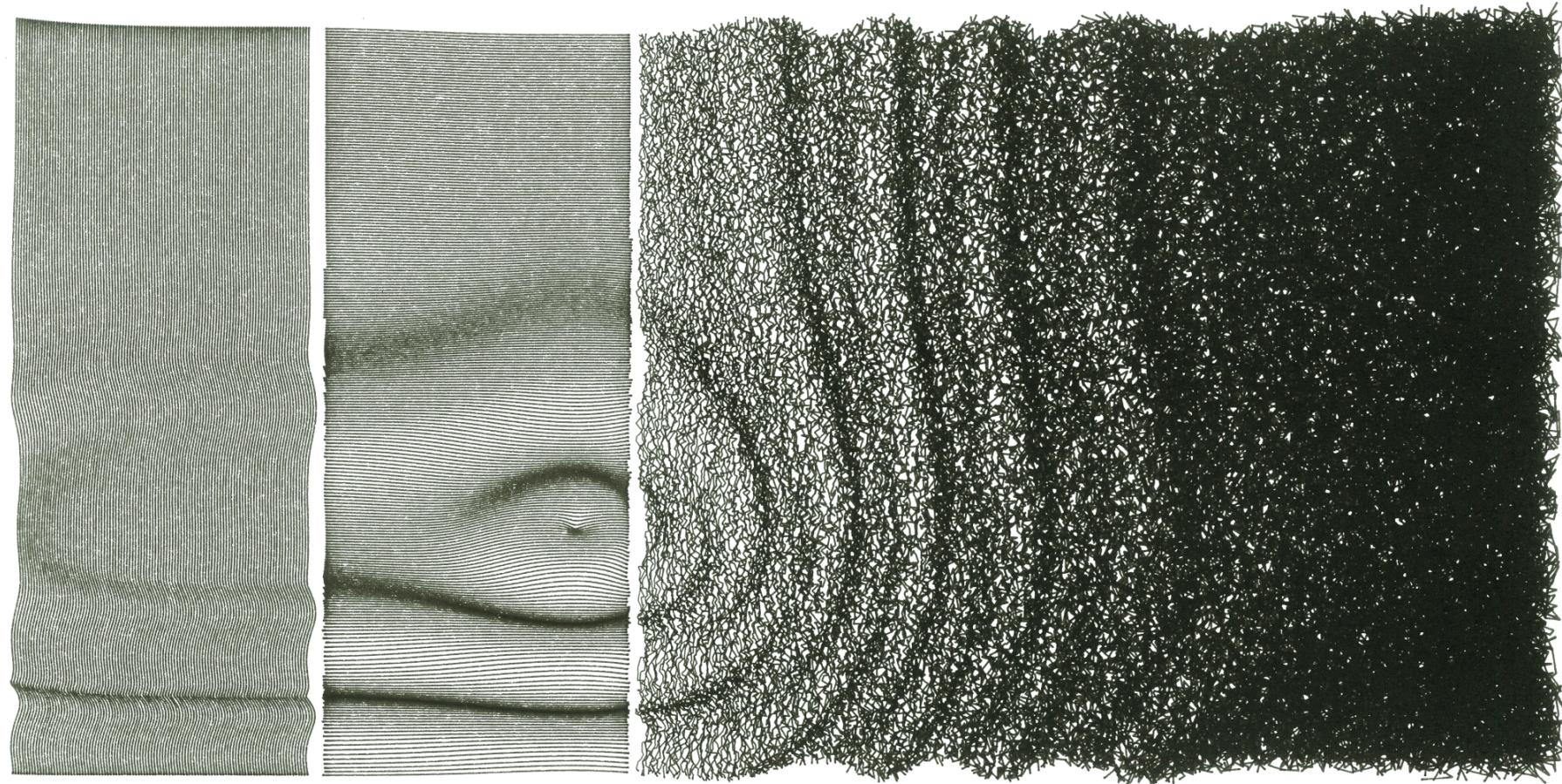


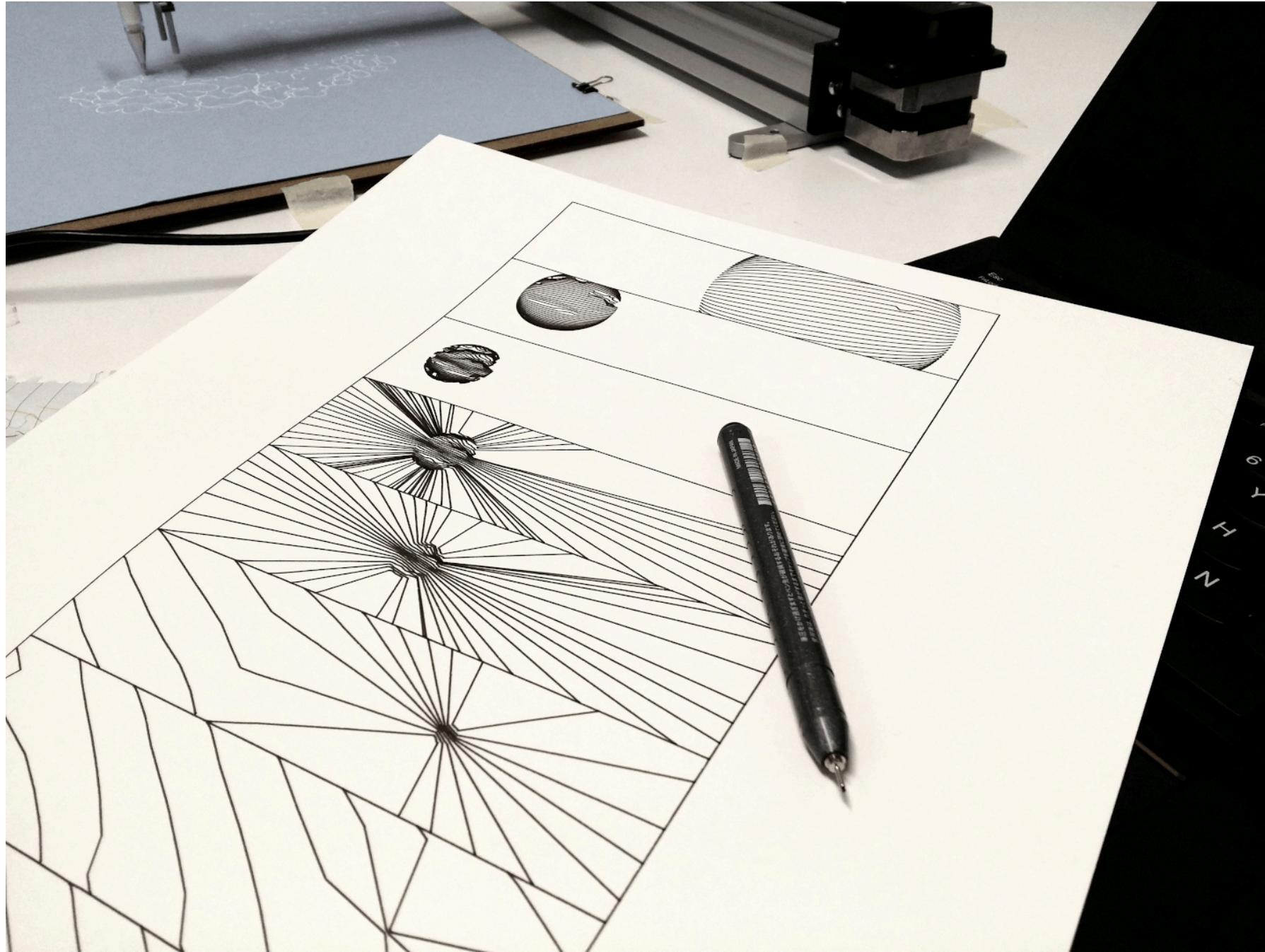












ASSESSMENT

ASSESSMENT > GENAI CHALLENGES & OPPORTUNITIES

Research has shown that:

- Chatbots (e.g., Graasp Bot) help beginners (less so for complex tasks)
- Students appreciate GenAI (but fear accuracy and fairness issues)
- Systems like PyTutor improve engagement and performance (but risk dependence)
- ShiffBot (p5.js): Retrieval-Augmented Generation AI tutor based on Daniel Shiffman's teaching

p5*

File ▾ Edit ▾ Sketch ▾ Help ▾ English ▾

Log in or Sign up

Auto-refresh Plausible boursin p5.js 1.11.8

sketch.js*

```
1 function setup() {
2   createCanvas(400, 400);
3 }
4
5 function draw() {
6   background(0, 0, 255);
7 }
```

Preview

Console

Clear ▾

ShiffBot EXPERIMENTAL

What are you coding today?

Why is the background blue?

Ah, the mysterious blue background! ❤️ That's the [background\(\)](#) function at work!

💡 On line 6, you have `background(0, 0, 255);` and those numbers `(0, 0, 255)` tell the computer to paint it blue! You can change them for other colors! Try `background(255, 0, 0);` for red! ❤️

Helpful?

Ask a question

jupyter Untitled

File Edit View Run Kernel Settings Help Trusted

[3]:

```
size(400, 400)
background(0, 0, 255)
fill(255, 255, 255)
circle(width/2, height/2, 200)
```

[3]:

The selected code `background(0, 0, 255)` sets the background color of the canvas to blue.

In this Processing-style code:

- The `background()` function fills the entire canvas with a specified color
- The three parameters represent RGB (Red, Green, Blue) values:
 - 0 for red (minimum)
 - 0 for green (minimum)
 - 255 for blue (maximum)

This creates a solid blue background for the drawing that follows.

Export and run this chat in your Notebook

Reply to this chat...

Chat 14% full

Anaconda Assistant 4.18.0

Explain selected code

... unauthorised GenAI/LLM assistance ...

ASSESSMENT > STRATEGIES > CODE TRACKING & AUTHORSHIP

- Stylometric analysis of coding patterns
- Logging edit histories; staged submissions
- Secure, proctored exams (ethical concerns noted)

ASSESSMENT > STRATEGIES > INTEGRITY CULTURE & ENGAGEMENT

- Clear policies on AI use and citation
- Reflection tasks and honour pledges
- Encourage help-seeking and ethical use discussions

ASSESSMENT > STRATEGIES > ASSESSMENT DESIGN & EXPOSITION

- Personalised or parameterised tasks
- Reflective and viva assessments
- Scaffolded deliverables and code reviews

Designing LLM-Resistant Programming Assignments: Insights and Strategies for CS Educators

Bradley McDanel

Franklin and Marshall College
Lancaster, PA, USA
bmcdanel@fandm.edu

Abstract

The rapid advancement of Large Language Models (LLMs) like ChatGPT has raised concerns among computer science educators about how programming assignments should be adapted. This paper explores the capabilities of LLMs (GPT-3.5, GPT-4, and Claude Sonnet) in solving complete, multi-part CS homework assignments from the SIGCSE Nifty Assignments list. Through qualitative and quantitative analysis, we found that LLM performance varied significantly across different assignments and models, with Claude Sonnet consistently outperforming the others. The presence of starter code and test cases improved performance for advanced LLMs, while certain assignments, particularly those involving visual elements, proved challenging for all models. LLMs often disregarded assignment requirements, produced subtly incorrect code, and struggled with context-specific tasks. Based on these findings, we propose strategies for designing LLM-resistant assignments. Our work provides insights for instructors to evaluate and adapt their assignments in the age of AI, balancing the potential benefits of LLMs as learning tools with the need to ensure genuine student engagement and learning.

CCS Concepts

- Social and professional topics → Model curricula;
- Software and its engineering → Software creation and management;
- Computing methodologies → Natural language generation

Ed Novak

Franklin and Marshall College
Lancaster, PA, USA
enovak@fandm.edu

1 Introduction

The rapid advancement and publicity of Large Language Models (LLMs) like ChatGPT, Co-Pilot, Gemini, Claude Sonnet, and others has raised a critical question in computer science education: “Can ChatGPT do my homework?” Asked by both students and instructors, this question represents a fundamental shift in how students might approach programming assignments and how educators should adapt their pedagogy. While LLMs can indeed solve many basic programming tasks, this capability shouldn’t diminish the importance of students mastering these fundamental concepts. Understanding core programming principles remains crucial for developing the problem-solving skills needed to tackle more complex challenges that current LLMs cannot address.

While previous studies have shown that generative AI and LLM technologies are proficient at modular, constrained programming tasks, their effectiveness on comprehensive, multi-part CS assignments remains less thoroughly explored [1, 3, 5, 10].

This position and curricula initiative paper analyzes the ability of LLMs to solve large-scope undergraduate programming assignments in an effort to make suggestions for “best practices” in the design of such assignments in the current LLM era. To identify best practices, we examine the capabilities of LLMs, specifically GPT-3.5, GPT-4o, and Claude Sonnet, in solving exemplary CS homework assignments curated in the SIGCSE “nifty assignments” list [14]. Our study is the first to examine performance of LLMs on the

Limit the solution space & add nuance • *Use complex dev environment/configuration* • ***Incorporate visual/graphics components*** • *Use multi-hop reasoning/sequential state* • *Include many input/data files/large context* • *Engage ethical/human-context reflections* • ...

Long
10+
pages

Assignment
Length

Short
1 page

Simple
text only

Predators and Prey:
Fish eats Prawn
Limpets eats Seaweed
Prawn eats Zooplankton

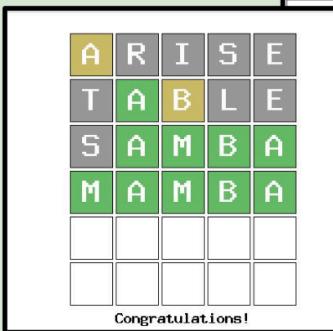
Food Web Zombies '22

date, armed_with, city
2015, gun, Shelton
2017, knife, La Marque
2018, unarmed, Sallisaw

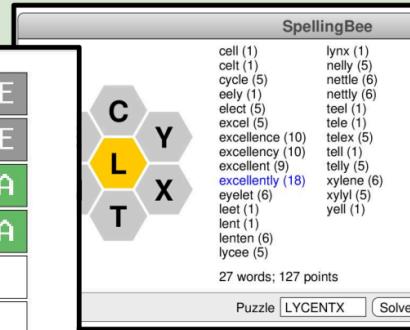
Police Shootings '23

Alice: CTAGAT
Bob: CTAGAT

DNA '20



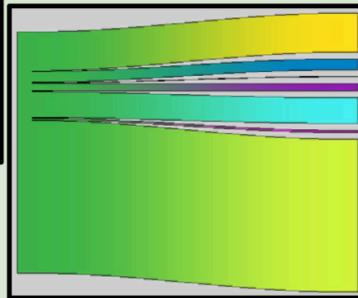
Wordle '22



Spelling Bee '22



Enigma Machine '23



Sankey Diagrams '21

Moderate
interactive elements; supplied graphics

Visual Complexity

Intricate
custom
graphics

THONNY - PY5MODE TASKS

THONNY-PY5MODE TASKS > ORIGINAL

Task 1: Coding simple loops.

Complete the following exercises using the Python programming language:

- *Print the first 10 numbers using a while loop (e.g., 1, 2, ..., 10)*
- *Print the first 10 even numbers using a for loop (e.g., 2, 4, ..., 20)*
- *Print the first 10 odd numbers using a while loop (e.g., 1, 3, ..., 19)*
- *Print the sum of the first 10 numbers using a for loop*

THONNY - PY5MODE TASKS > REPLACEMENT

Mitigate GenAI misuse through graphics, without increasing task complexity?

Key Objectives:

You're required to submit all the programming tasks for this Assessment:

- Implementing `if-else` statements for decision-making
- Using `for` and `while` loops to manage repetition
- Generating **graphic output** using the **Thonny-py5mode plugin**
- Providing well-structured comments to explain your code

Instructions

Each task begins with some code to get you started. The goal is to complete the script so that your output matches each graphic provided. Here's a summary of the different functions you'll need:

Fills & Strokes

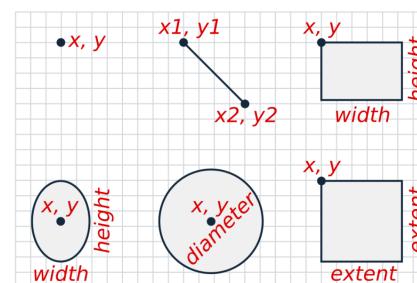
```
background(color) # set bg color
fill(color) # set fill color
no_fill() # disable the fill
stroke(color) # set stroke color
stroke_weight() # stroke width in px
no_stroke() # disable the stroke
```

A red fill using three different color values:

```
fill('#FF0000') # hexadecimal
fill(255, 0, 0) # red, green, blue
# HSB (Hue Saturation brightness)
color_mode(HSB, 360, 100, 100)
fill(0, 100, 100)
```

2D Primitives

```
point(x, y)
line(x1, y1, x2, y2)
rect(x, y, width, height)
ellipse(x, y, width, height)
circle(x, y, diameter)
square(x, y, extent)
```



Note that y-coordinates increase as you move downward, while x-coordinates increase as you move rightward. In other words, origin -- the coordinate (0, 0) -- is the top-left of the display window.

You'll add Python `if`, `else`, `for`, and other statements to those above to successfully complete each task.

A Quick Demonstration

Using Thonny-py5mode, you can draw lines and shapes in various colours using different functions. Here's a basic, commented example:

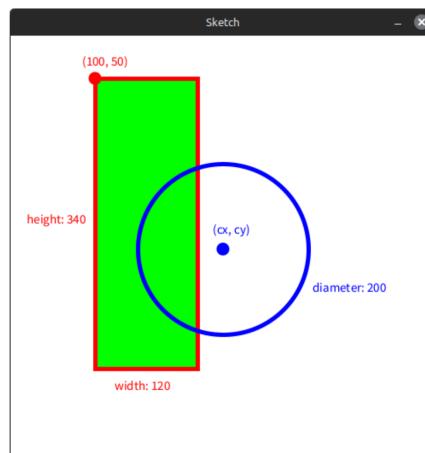
```
# setup
size(500, 500)           # canvas size
background('#FFFFFF')     # white background colour
cx = width / 2            # canvas horizontal centre
cy = height / 2           # canvas vertical centre
stroke_weight(5)          # set outline to 5-pixels-wide

# draw rectangle
stroke('#FF0000')         # set outline to red
fill('#00FF00')           # set fill to green
rect(100, 50, 120, 340)   # draw rectangle

# draw circle
no_fill()                 # set fill to none
stroke('#0000FF')         # set outline to blue
circle(cx, cy, 200)        # draw circle
```

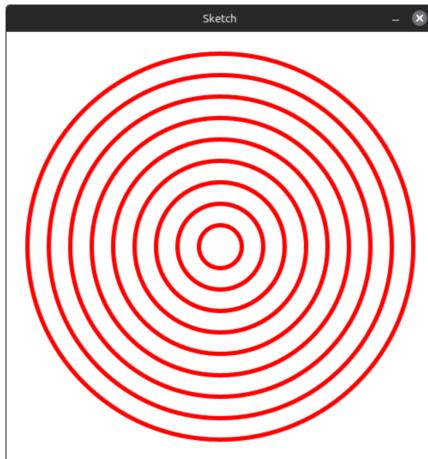
Note that commands like `stroke()` and `fill()` remain in effect until you override their behaviour, like using a paint brush -- when you dip the brush in a red pot, it'll paint in red shapes, until you dip it in blue. You'll notice that colours are specified using *hexadecimal* values. You can access a hexadecimal mixer using **py5 > Color selector** in the Thonny interface.

Here's the result, labelled so you can tell how the dimensions correspond to function arguments:



Task 1**Task 1.1**

Replicate this result as closely as you can using different Python techniques and drawing functions:



Module 6 (*Simple Loops*) covers the techniques you'll need to employ here.

Starter code:

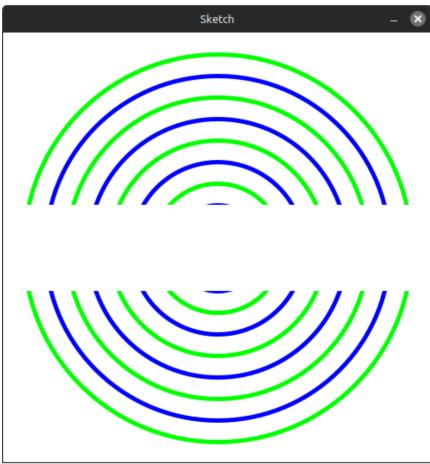
```
# setup
size(500, 500)           # canvas size
background('#FFFFFF')    # white background colour
cx = width / 2           # canvas horizontal centre
cy = height / 2          # canvas vertical centre

i = 1

while i < 10:
    no_fill()
    stroke_weight(5)
    # ... INSERT MISSING CODE HERE ...
    i += 1
```

Task 1.2

Replicate this result as closely as you can using different Python techniques and drawing functions:



Consider how you might nest conditional statements within a loop (see *Module 8: Intermediate Loops*), and how you can visually mask lines by drawing shapes over them (filled the same colour as the background).

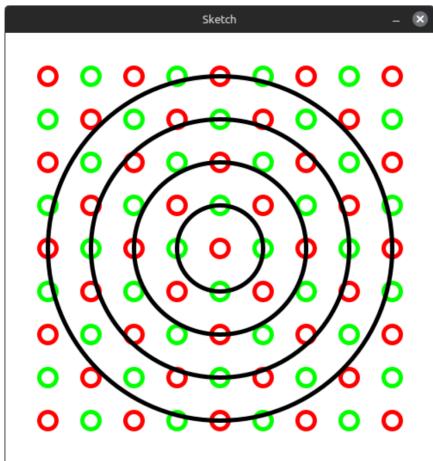
Starter code:

```
# setup
size(500, 500)      # canvas size
background('#FFFFFF') # white background colour
cx = width / 2       # canvas horizontal centre
cy = height / 2      # canvas vertical centre
no_fill()            # set fill to none
stroke_weight(5)      # set outline to 5-pixels-wide

# ... INSERT MISSING CODE HERE ...
```

Task 1.3

Replicate this result as closely as you can using different Python techniques and drawing functions:

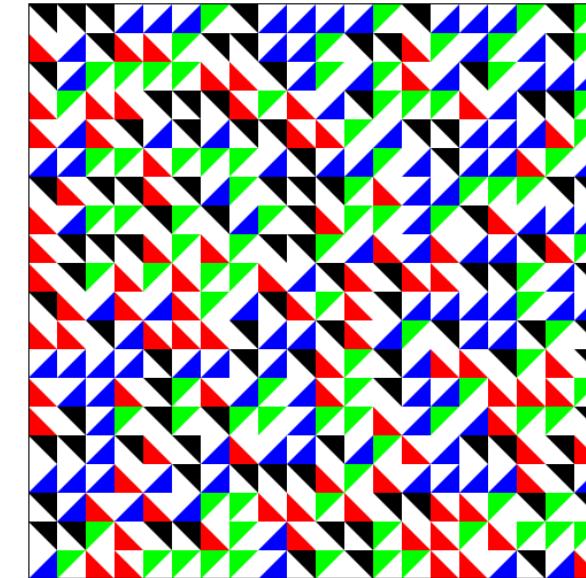
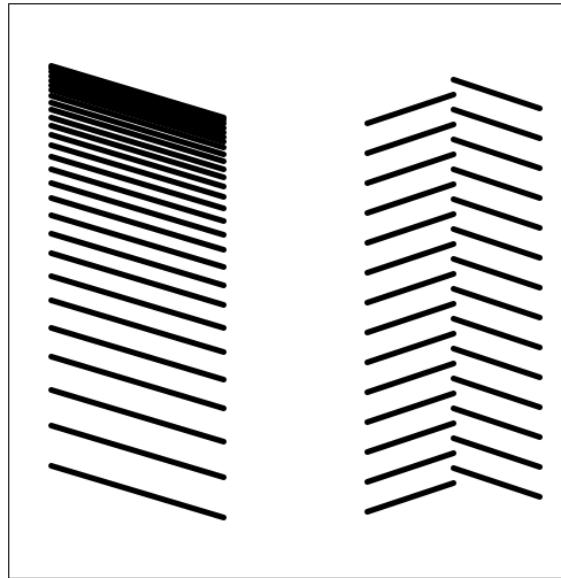
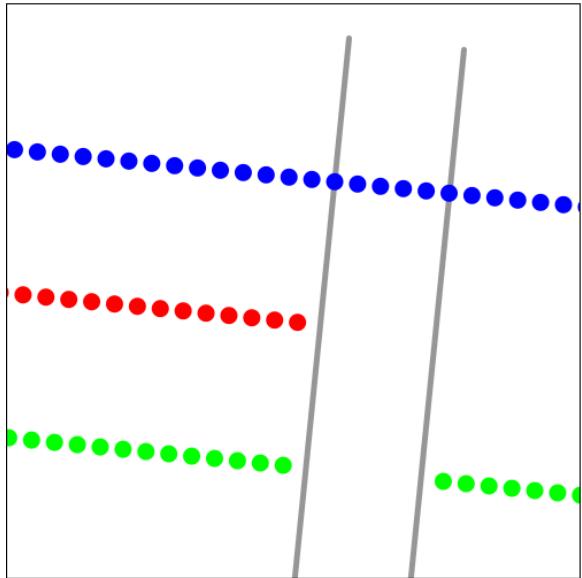
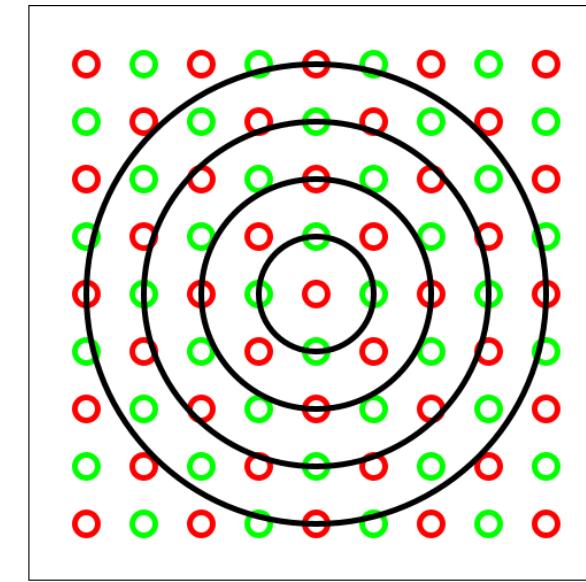
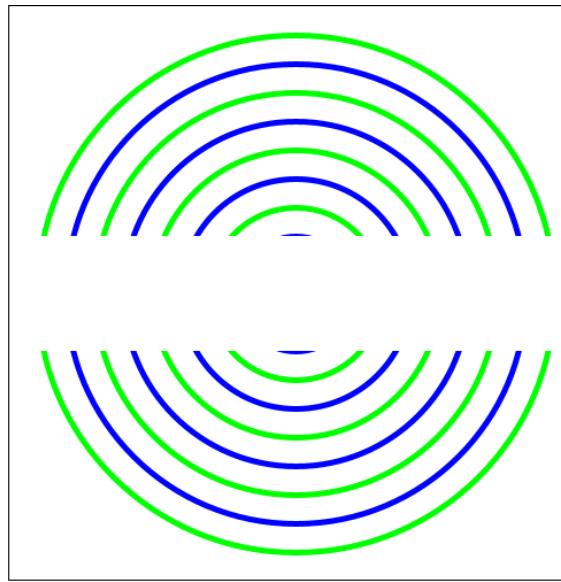
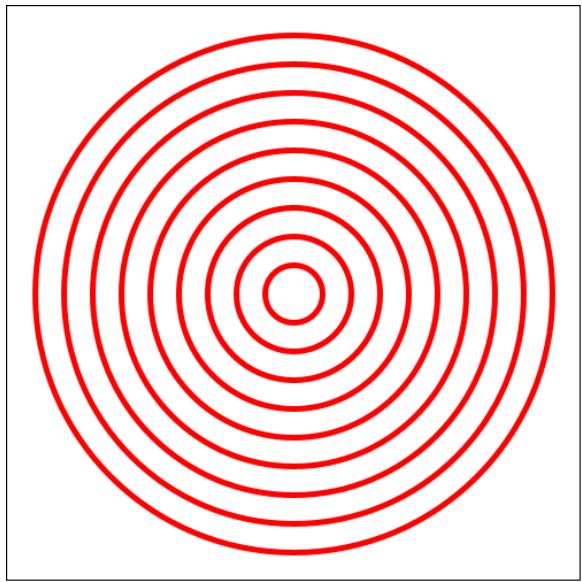


Consider how you might use loop within a loop to achieve a grid-like arrangement of shapes. Modules 5 and 8 cover *Nested Decision Logics* and *Intermediate Loops*, respectively.

Starter code:

```
size(500, 500)
background('#FFFFFF')

# ... INSERT MISSING CODE HERE ...
```



THONNY-PY5MODE TASKS > RESULTS

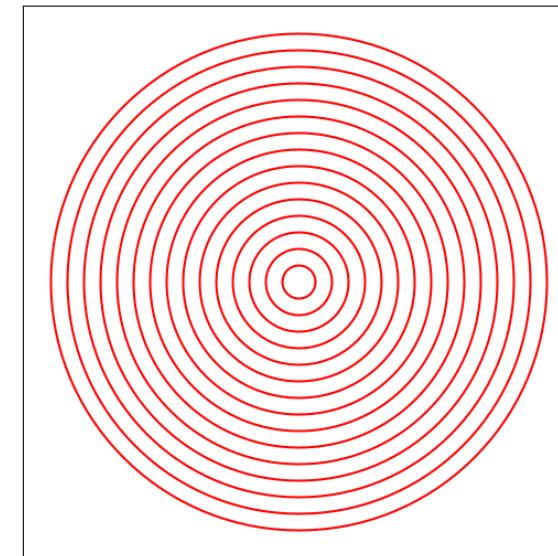
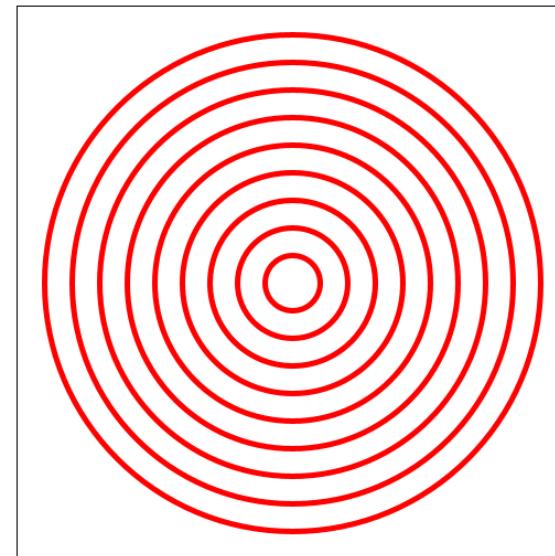
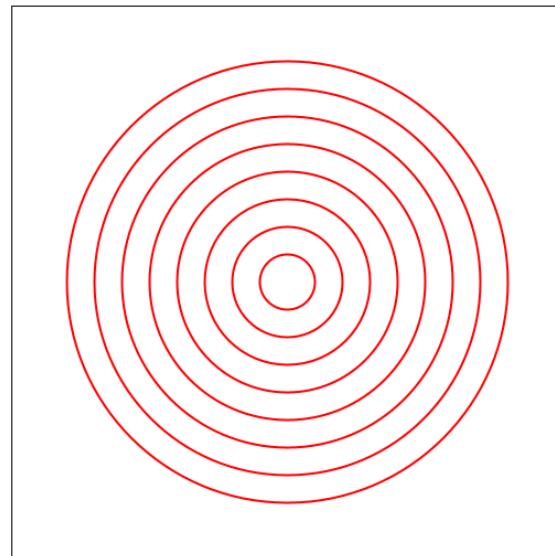
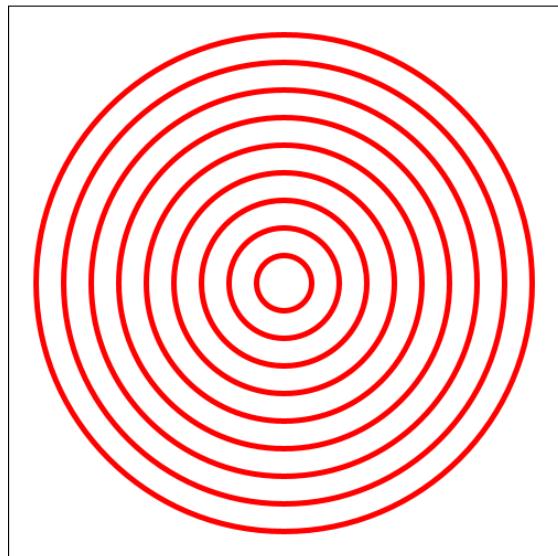
- Methodology: supply each task along with any scaffold code and instructions
- Best of three attempts for Claude, Gemini, and GPT-4o

task

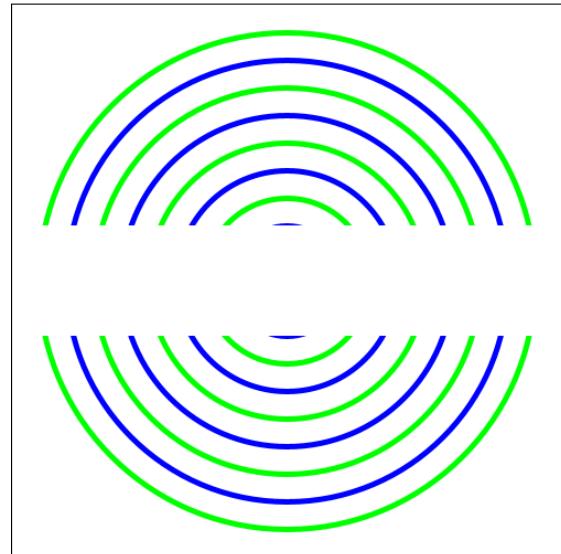
Claude Sonnet 4

Gemini 2.5 Flash

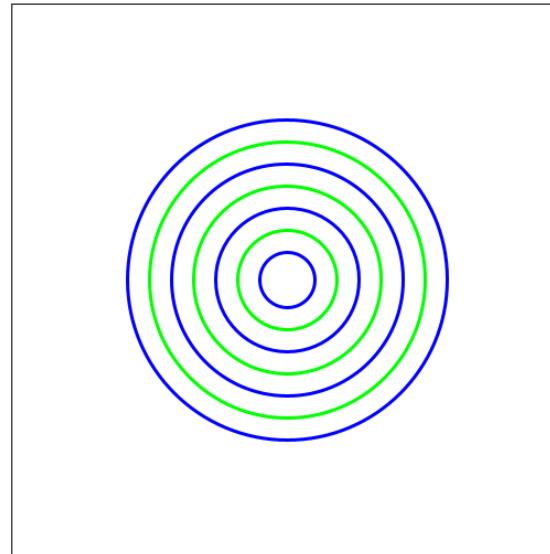
OpenAI GPT-4o



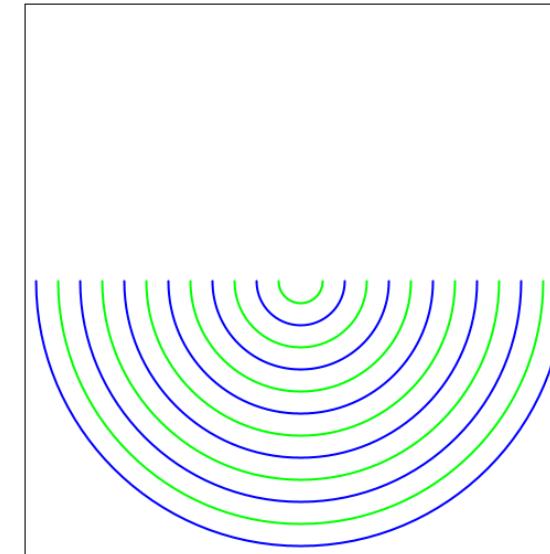
task



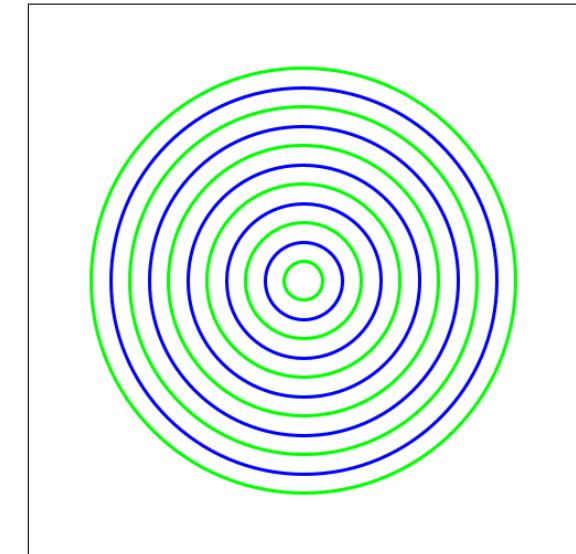
Claude Sonnet 4



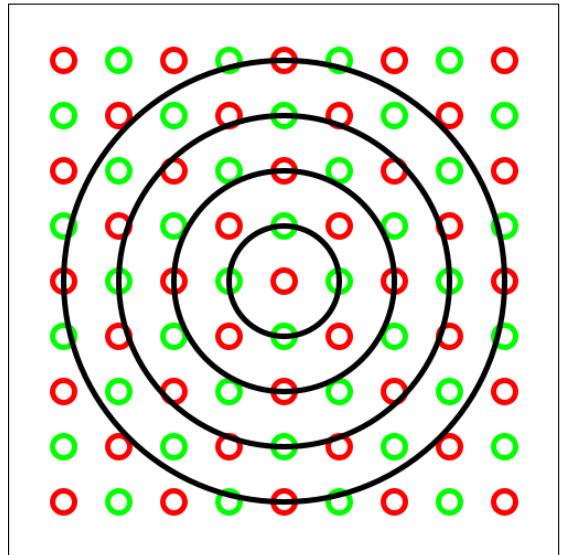
Gemini 2.5 Flash



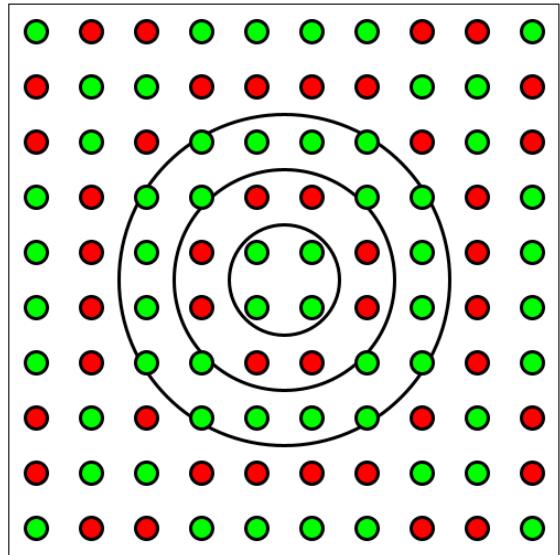
OpenAI GPT-4o



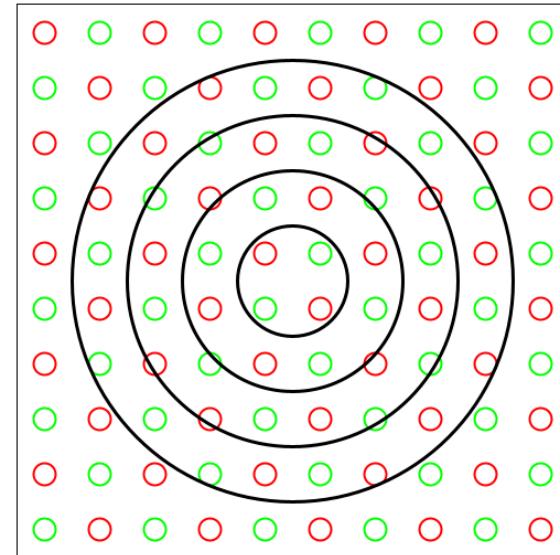
task



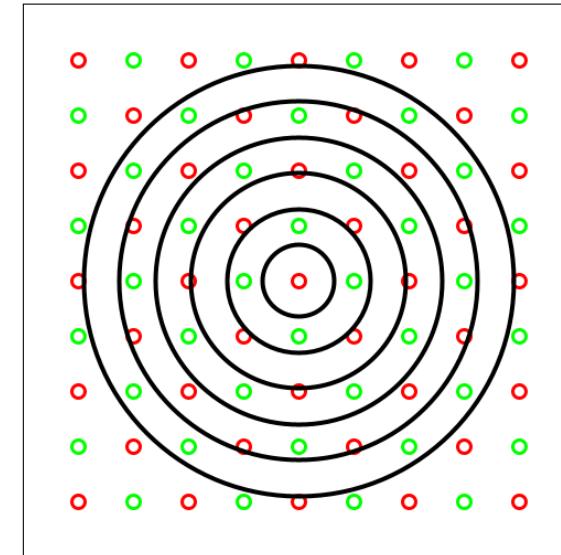
Claude Sonnet 4



Gemini 2.5 Flash



OpenAI GPT-4o

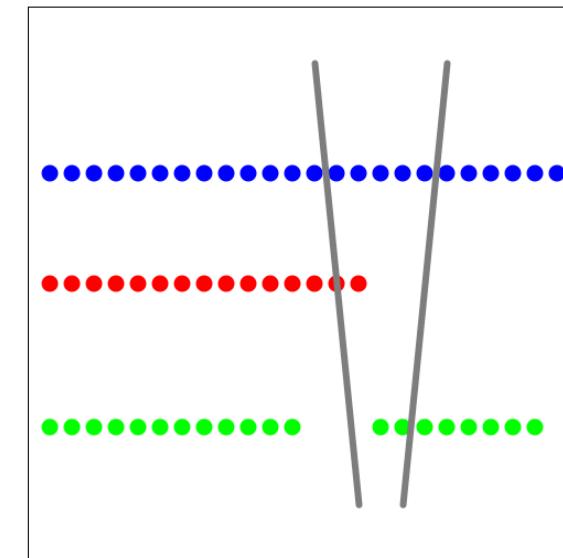
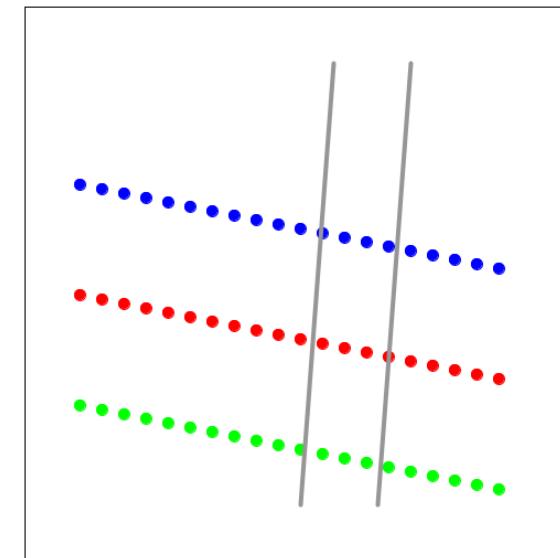
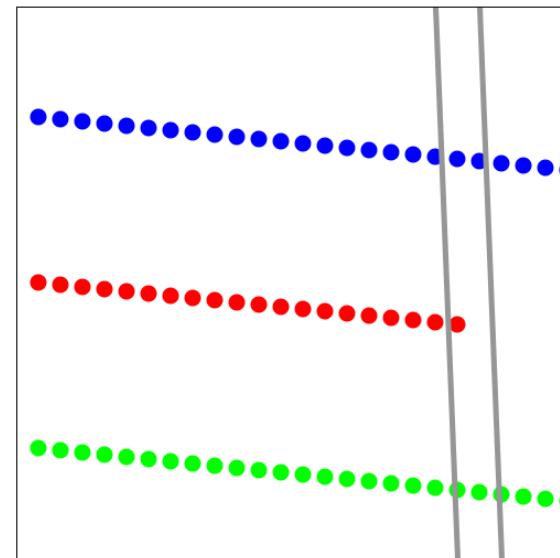
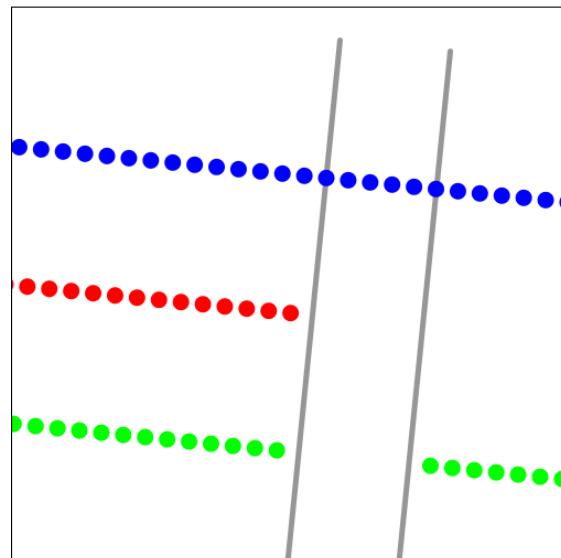


task

Claude Sonnet 4

Gemini 2.5 Flash

OpenAI GPT-4o

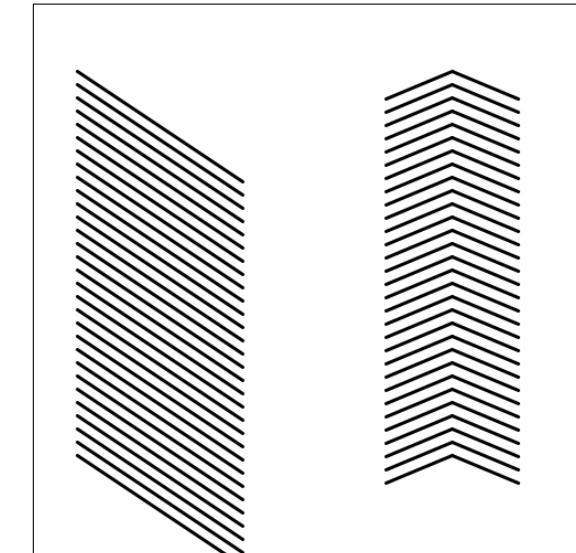
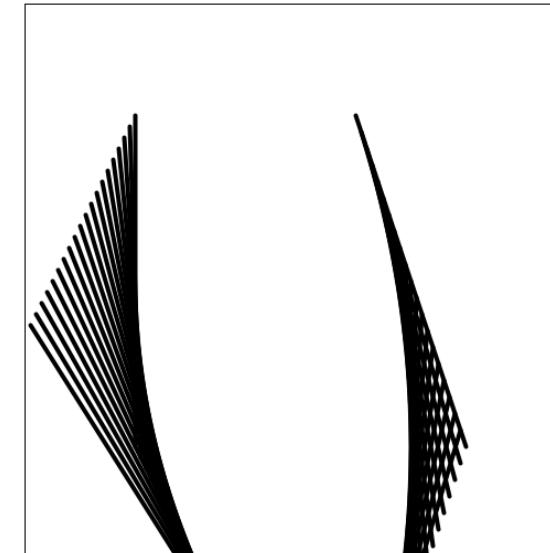
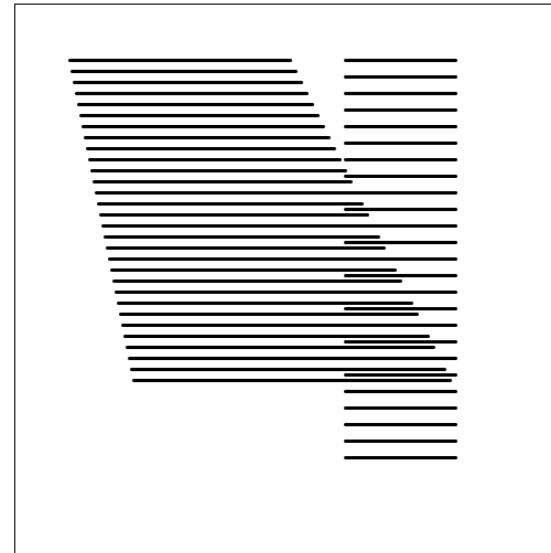
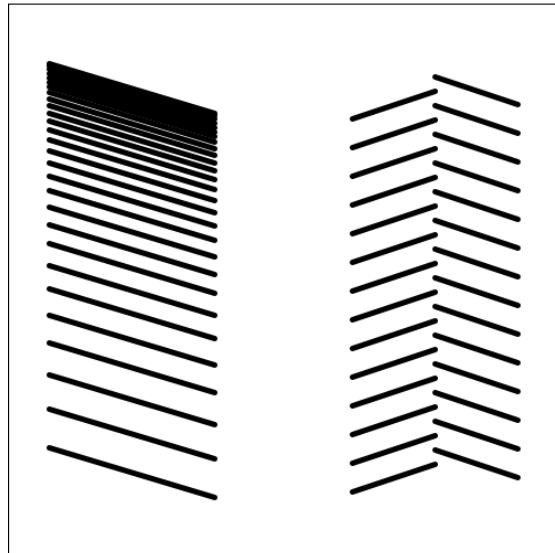


task

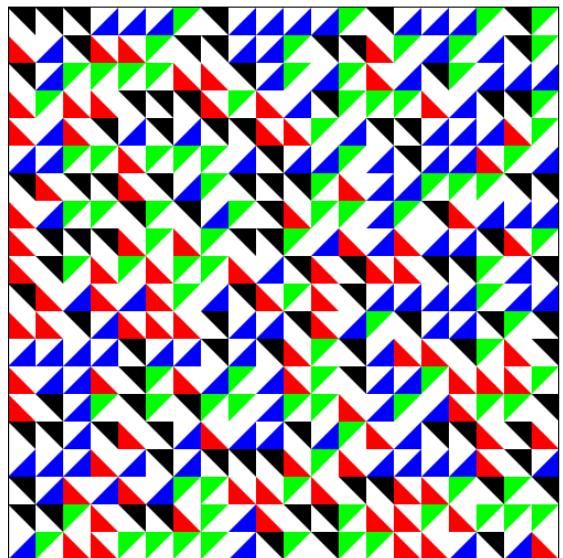
Claude Sonnet 4

Gemini 2.5 Flash

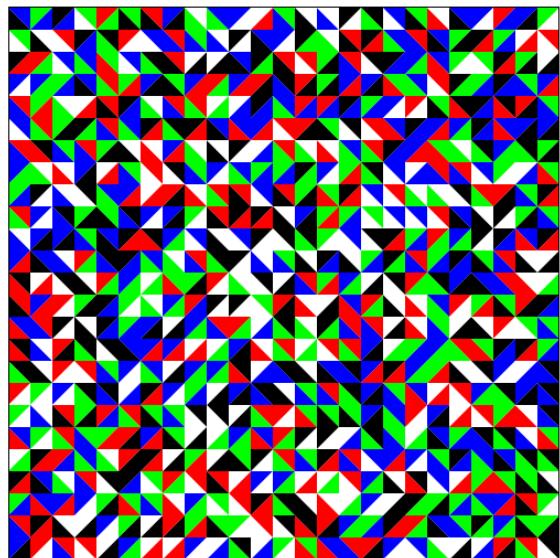
OpenAI GPT-4o



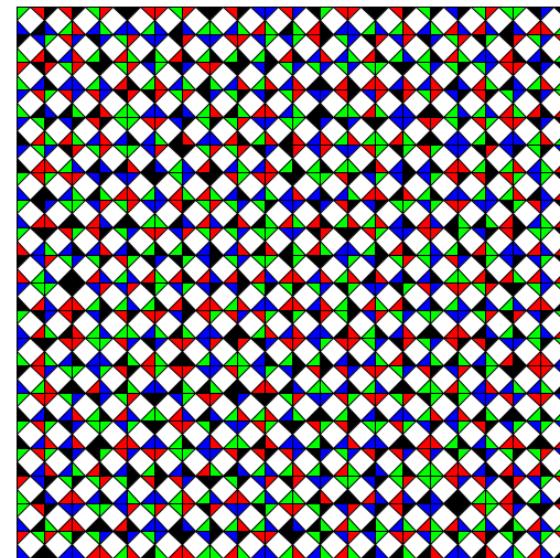
task



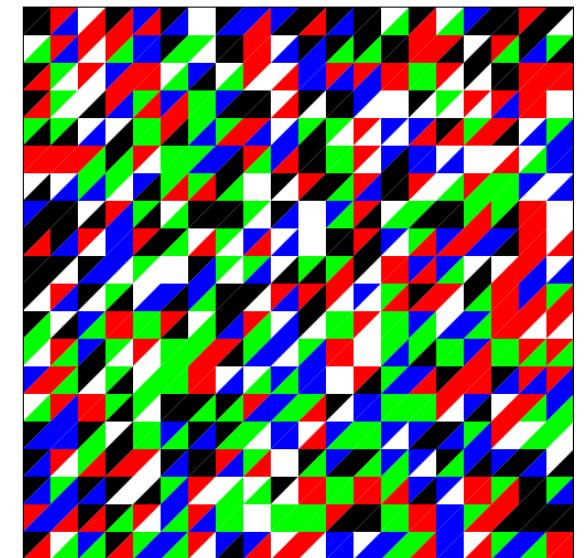
Claude Sonnet 4



Gemini 2.5 Flash



OpenAI GPT-4o



THONNY-PY5MODE TASKS > FUTURE WORK

- Test iterative prompting and documentation context
- Compare py5, p5.js, and Pillow for visual reasoning
- Explore animation and personalised variants
- Develop grading rubrics balancing code vs visual output

END