Draft 1

While you're working on your copy, articulate (in writing) the key critical questions that emerge through this exploration. (You can use the prompts found on the first page of this brief as a guide.) Discuss how the project that you're copying raises these questions and write a proposal for a studio-based experiment that would allow you to explore them further.

While remaking Enigmatriz's ASCII cow drawings using p5.js, I began to question whether the tool functions primarily as a means of producing images, or as a system that regulates conditions under which images are allowed to appear. Attempting to manually recreate the ASCII image without fully understanding image-to-text conversion revealed how p5.js foregrounds process over outcome. Errors, warnings, and failed executions became as present as the final image itself, shifting my attention from what was made to how it was negotiated.

This raised several critical questions: What does p5.js privilege as valid output, and what does it suppress? How do errors and console messages operate—not simply as technical feedback, but as forms of communication and control? What relationship exists between visible results and invisible labour within creative coding environments? Finally, how does understanding (or misunderstanding) a tool affect authorship and agency within the work?

The copied project suggests that ASCII imagery is not simply drawn, but emerges through a system of translation, constraint, and interpretation. Building on this, I propose a studio-based experiment that treats p5.js as a mediated system rather than a drawing tool. By framing the sketch.js, console, and preview windows as distinct roles within a 'football match'—players, coaches, and broadcast—I will use comments and console outputs as sites of hidden dialogue and conflict, while maintaining a stable visual outcome in the preview.

Through iterative variations of this system, I aim to examine how creative labour, communication, and control are distributed within the tool.

Draft 2

Identify a reference from the reading list that you can use as a lens through which to view and analyse your project. Then create a second draft of your writing that advances your enquiry in response to this new context.

In my project Methods of Iterating, I recontextualise p5.js not as a mere tool for image production, but as a site for experimenting with the distribution of instructions. By shifting the focus from the final visual output to the conditions of its emergence, I position myself not as a user of a tool, but as a designer of systems and relationships.

This enquiry is informed by the Conditional Design Manifesto, which proposes that "the process is the product" (Lust, 2011). Rather than prioritising a fixed artefact, Conditional Design understands design as something unfolding over time. Lust argues that contemporary design is concerned less with static form and more with "time, relationships and change" (Lust, 2011). My iterations follow this logic. Although the visible output — for example a cow or a cake — appears visually similar, the epistemological structure of each iteration changes.

I moved from manually constructing images using ASCII characters (a highly human-centred method) to automated pixel-data translation (a delegated machine process). Each method therefore represents a different mode of knowing and instructing the system. The value of the work lies not in the final static image, but in the systematic exploration of the logic that produces it.

The manifesto also suggests a shift in metaphors. Where systematic thinking was once understood through the metaphor of the machine, it is now closer to that of an organism (Lust, 2011). This corresponds to my observation of p5.js as a living ecosystem. I understand the `sketch.js` window as a field of interaction where code-agents operate, the `console` as a feedback mechanism, and the `preview` window as a performative output. The work therefore functions not as a linear production chain but as an emergent pattern generated by interacting components.

Within Conditional Design, constraints are not considered limitations but generative conditions (Lust, 2011). In my work, I actively adopt the restrictions of p5.js — including error messages and red error highlights — and repurpose them as aesthetic material, for example using error markers to construct visual elements such as strawberries. This act reframes the software's rules as opportunities for improvisation, aligning with the principles of Adhocism (Jencks and Silver, 1972).

My container for holding this research takes the form of an Accordion Recipe Book. A recipe is the perfect embodiment of Conditional Design: it is not the image of the meal, but a set of conditions under which the outcome appears. By documenting "failed recipes" alongside successful ones, I highlight that failure is a moment where the system reveals its limits. As a "Gardener" rather than a "Sculptor," I surrender total control over the form, focusing instead on designing the formation— the system of translation where time and logic converge to create design.

Bibliography:

- Jencks, C. and Silver, N. (1972) Adhocism: The Case for Improvisation. London: MIT Press.
- Lust (ed.) (2011) Conditional Design Workbook. Amsterdam: Valiz. (Contains the Conditional Design Manifesto by Luna Maurer, Edo Paulus, Jonathan Puckey and Roel Wouters)
- Maurer, L., Paulus, E., Puckey, J. and Wouters, R. (2011) 'Conditional Design Manifesto', in Lust (ed.) Conditional Design Workbook. Amsterdam: Valiz.

Draft 3

For the third draft of your written response, render your text using the tool or medium that you've been exploring during this project. This is both a visual and intellectual exercise. How does the text and its meaning change when you translate it in this way?

Draft 3 shifts my writing from explanation to operation. Rather than presenting my statement as a printed paragraph, the text functions inside p5.js as an executable recipe — a set of conditions that produces an outcome. In doing so, the work answers the question of how meaning changes through translation by performing it rather than describing it.

The original argument — that the same image can be reached through manual, canonical, and automated processes, and that authorship shifts according to how instructions are distributed — is no longer stated verbally but demonstrated through form. The writing is reorganised into recipe sections (Ingredients, Equipment, Chef's Notes, Output), so it is not read linearly but followed procedurally. Visually it appears as code, while simultaneously generating a cake image in the output window, allowing instruction (text-as-code) and execution (visual result) to coexist.

Through this transformation, narration becomes process, and the authorial "I" disappears, replaced by system actions such as functions running and outputs appearing. The draft therefore positions me not as the maker of an image but as a designer of conditions, and it also operates as the textual counterpart — and meta-structure — connecting the three final recipe formats in my project.

sketch.js — Saved: 5일 ago

```
/**
 * Draft 3 — Render the written response USING the tool (p5.js)
 * Option 1: Statement translated into a "recipe-code manifesto"
 *
 * Controls:
 *  - Press 1 : Code (manifesto) view
 *  - Press 2 : Output view (cake preview)
 *  - Press S : Save PNG
 */

let viewMode = 1; // 1 = code manifesto, 2 = cake output
let codeLines = [];
let lineMeta = []; // same length; meta tags for simple highlighting

function setup() {
  createCanvas(800, 1500);
  textFont("Courier New");
  textSize(14);
  randomSeed(1);
  noiseSeed(1);
  noLoop();

  buildRecipeCodeFromStatement();
}

function draw() {
  background(255);

  if (viewMode === 1) {
    drawCodeManifesto();
  } else {
    drawCakeOutput();
  }

  // subtle footer
  noStroke();
  fill(0, 80);
  textSize(12);
  textAlign(LEFT, BOTTOM);
  text("Draft 3 (p5.js) — Press 1: Code view  |  Press 2: Output view  |  Press S: Save", 18, height
- 14);
}

function keyPressed() {
```

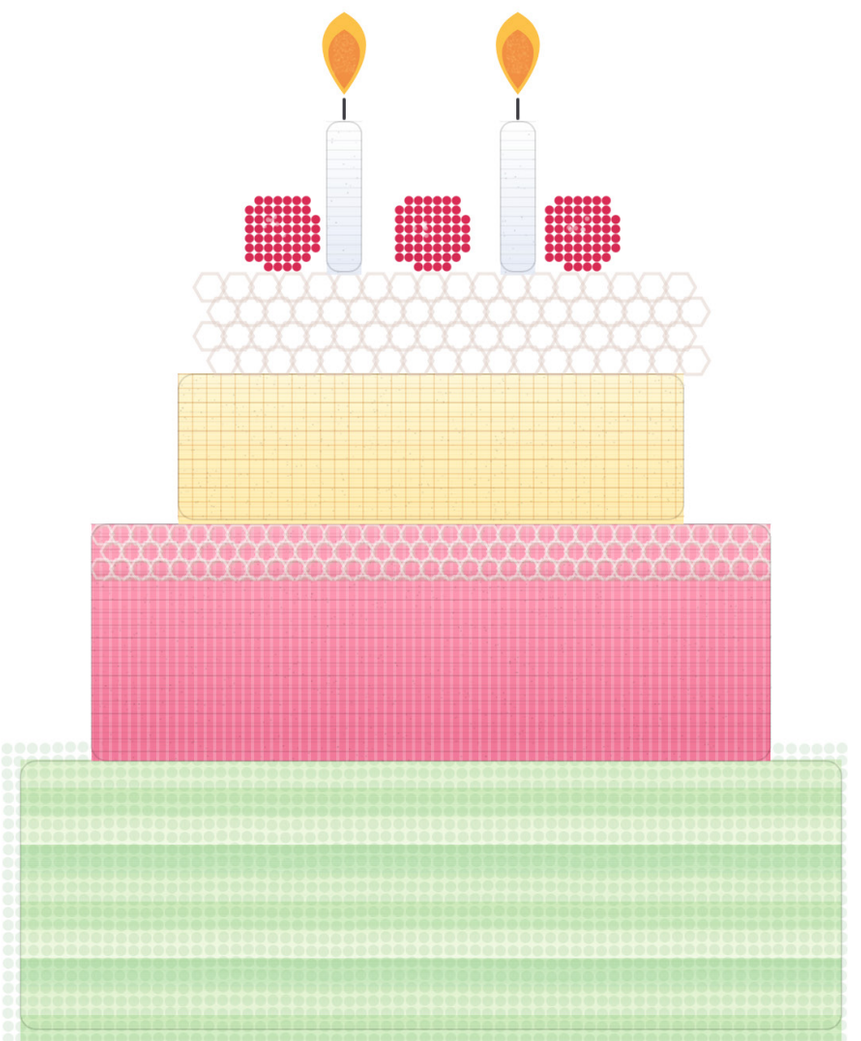
p5.js coding instructions in which the recipe is written as executable code, structuring the conditions required to generate a cake image.

```
                                                    Code-as-recipe: instructions first, outcome la
//=========================================================================//
//                          CODING AS A RECIPE                             //
//                       Draft 3 — Rendered in p5.js                        //
//=========================================================================//

//---------------------------- Ingredients ----------------------------//
let authorship;
let instruction;
let epistemology;
let failure;
let time;
let sharing;

//---------------------------- Equipment -----------------------------//
function followPatterns() {}
function breakPatterns() {}
function reversePatterns() {}
function distributeInstruction() {}
function allowOutcomeToAppear() {}

//------------------------------ Method ------------------------------//
function setup() {
  createCanvas(900, 900);
  noLoop();
}

function draw() {
  // A recipe is not an image of the outcome,
  // but conditions under which the outcome may appear.

  followPatterns();
  breakPatterns();
  reversePatterns();
  distributeInstruction();
  allowOutcomeToAppear();
}

//------------------------ Chef's Notes ----------------------------//
// The statement below is "cooked" into a code-comment:

/*
  Through iteration, I began to treat coding not simply as a means to
  produce images, but as a way to think about authorship. I repeatedly
  arrived at the same visual outcome through different paths—each
  shaped by a different epistemology and a different distribution of
  instruction. Coding environments are built on accumulated patterns:
  reusable solutions that make certain actions easier, faster, and
  more "proper." p5.js sits on top of these inherited conventions. My
  practice moves between following them, breaking them, and using them
  backwards—an adhocist approach of situated reconfiguration. Rather
  than producing a single solution, I assemble programmes of solutions
  from what already exists—borrowing from tutorials, archives, and
  other people's code—then reassembling them into my own working
  method. This is why I frame the project as a recipe book. A recipe
  is not an image of the outcome, but a set of conditions under which
  the outcome may appear. Recipes carry instruction, repeatability,
  variation, failure, sharing, and time. In this book, the process is
  the outcome. ASCII art becomes drawing material inside the code
  editor itself; even error behaviour becomes an ingredient. Functions
  and transforms operate as kitchen equipment, shaping how the recipe
  runs. Like p5.js—where code, preview, and console operate together
  as an emergent system—my role shifts from tool user to designer of
  conditions and translation systems. I do not aim for total control
  over the final form; instead, I design the conditions under which
  forms emerge. Failed recipes are not mistakes, but moments where the
  system reveals its limits, assumptions, and dependencies. If
  traditional design is sculpture—directly carving an outcome—my role
  is closer to gardening: selecting soil, light, water, and seeds,
  then allowing growth to unfold beyond my control.
*/

//---------------------------- Output ------------------------------//
// Press 2 to view the "cake" output.
```

Draft 3 (p5.js) — Press 1: Code view | Press 2: Output view | Press S: Save

The recipe translated into code form, where culinary instructions operate as a procedural script rather than a linear text.

Output View — the same “recipe” allows a cake to appear



Note:
This view is intentionally a “result” panel.
The meaning shift happens when you compare it to View 1 (code-manifesto):
where instruction lives changes what authorship feels like.

Draft 3 (p5.js) — Press 1: Code view | Press 2: Output view | Press S: Save

The resulting cake image produced through the execution of the coded recipe.