Fractal Accelerator — Class Overview

Class 1: Agents & the Map of the Computing World

- The central habit: **ask the agent for what you need**
- Mental model of an agent: language model + instructions + environment + tools + autonomy
- Chat assistant vs. agent with tools (advice vs. action)
- The five capabilities the course develops: habit of asking, giving agents tools, teaching agents about you, avoiding harmful outcomes, learning the language of computers
- Prompts operate at different levels (narrow action → high-level outcome); anatomy of a useful prompt (outcome, sources, tools, constraints, output structure)
- Prompting as conversation and design process, not one-shot incantation
- **exercise**: investigate your files and inbox for insight
- Principles for connected tools
- The map of the computing world: your devices, other people's devices, and cloud computers, all connected via the internet — "the cloud" = computers in a data center; "local" = the computer right here
- Separating **process / data** and asking "where does each live?"
- Why use the cloud: availability, persistence, internet access
- When local is better: simpler, faster, cheaper, more private, offline
- Zo as a personal cloud computer (run code, host, store, connect services, reachable by messaging channels, scheduled work)
- Permissions, autonomy, and safety: scoped access, read-only first, confirmation for consequential actions, reversibility, backups
- The **Personal OS**: durable context in an organized folder structure instead of trapped in your chats or locked in an app
- Nested context (open the agent in the project folder, not the top level)
- Syncing local and cloud context
- The agent design process: clarify goal → give context → request & evaluate proposal → build, test, keep going
- Spoken "goal dumps" as a way to surface context faster than composed prompts
- Being the "CEO": evaluating proposals, using "hell yes"/"hell no" reactions as design information
- Being blocked ≠ failure, just change the approach
- Moving context between agents/conversations (summarize to durable files)
- Homework loop: build something that helps you + development log + work report + 3-minute demo

After Class 1 You Should Have:

- ☐ Created your Personal OS and synced it to Zo
- ☐ Built and tested some useful software that solves a problem you have
- ☐ Kept a dev log and can give a three-minute demo about your solution

Class 2: Context Engineering & Memory

- The main idea: **when output feels like gambling, curate the context**
- Two failure modes of context: missing information and too much noise
- Context = everything the model can see this turn (instructions, messages, files, tool results)
- New conversations are a fresh start! Clean context is amazing!
- Don't overload memory and AGENTS.md files.
- The working loop: **Goal → Generate → Inspect → Clarify → Try again**
- The AI output is always good evidence about the context, even if it is not the right answer
- Agents are generative and pattern-recognizing: give positive examples of what you want, not prohibitions
- The four layers of an agent system: **model / context window / agent / harness** ("the model predicts, the context steers, the agent acts, the harness shapes the environment")
- Debugging map by layer: hallucination → model/data problem; ignored preferences → context problem; flailing or bad process → agent-loop problem; access/tool failures → harness problem
- "AI slop" = accepting low-judgment/low-taste output
- Long-term memory live in `AGENTS.md`, layered by folder
- Procedural memory: **skills** (`SKILL.md` + references/assets/scripts) for recurring kinds of work (also, software generally)
- `AGENTS.md` = "remember this"; skill = "when this kind of problem appears, follow this process"
- One canonical data home: single Personal OS folder/repo, deliberate sync, instructions near the data (the monorepo pattern)
- Exercise: retrospective — improve a repeating process in a shared workspace
- Exercise: where do you need better context? (emotions and recurring mistakes as diagnostic signals)
- Exercise: what should your agent remember? (candidate `AGENTS.md` entries)
- Exercise: what should your agent learn how to do? (candidate skills)
- Configuring the harness safely: subscriptions vs. metered credits, spending limits, browser chat vs. computer-based agent

After Class 2

- ☐ Personally read and approve your `AGENTS.md` in your personal-os to guide your agent.
- ☐ Created your first skill for a procedure or tool you keep using/repeating
- ☐ Both local and cloud agents can use your Personal-OS skills/AGENTS.md

Class 3: Engineering as a Way of Thinking

- The central idea: **when building is easy, the bottleneck is good thinking**
- The abstraction ladder (bits → languages → programs → OS/networks → systems → real-world problems); agents help connect to every part of this.
- There is a human abstraction ladder too (biology → cognition → behavior → goals/values/society)
- Asking "at what level is the bottleneck to this problem's solution?"
- **Metacognition**: making thinking inspectable so it can be discussed, challenged, delegated, and improved
- The engineering design process as a metacognitive tool

- The 8-step engineering design process: define the problem → background research → specify requirements → brainstorm → evaluate & pick → prototype → test against requirements → communicate
- Exercise: client-led design process (one real client as source of truth)
- Problem definition as language-building (developing a shared vocabulary before solving)
- Research as collective intelligence (different framings, compare, synthesize)
- Making implicit requirements explicit ("that solves it but I don't want it" is important data, always investigate your strong feelings!!)
- **Design problems are fractal**: decompose into subproblems, each with its own design process, connected by interfaces
- Use the process when it helps, not as ritual (signals to use it are: spinning wheels, disagreement, unclear success criteria, output without progress)
- Not everything valuable in life is problem-shaped, you don't always need to be an engineer (often, you should be jamming/playing instead of engineering)
- AI solutionism: don't bend the problem to fit the tool, do you need AI, or software at all? Remember everything else that exists!
- Homework: one weekly goal + teach yourself one skill from the reference library

After Class 3 you should have:

- ☐ Recorded an ambitious problem/goal in your project folder, to be completed this week (distinct from your week 1 and week 2 goals)
- ☐ Developed a written problem statement and requirements for solving that problem
- ☐ Completed one reference-library guide, and recorded notes for yourself somewhere, which we can check together.

Class 4: Debugging Agents & First Dollar

- Debugging is a diagnostic process: inspect the actual interaction BY READING THE DATA, not a vague impression ("try harder" is not a diagnosis)
- Five places where the failure may have been introduced: **1. in the instructions; 2. in the skills / files / context; 3. in the code or commands; 4. in the connections or integrations; 5. in the platform itself (codex/anthropic/etc...)** — the right fix depends on which
- The agent debugging process: examine the interaction → analyze the agent's reasoning → name the most likely cause ("it most likely failed because ___") → test/investigate your hypothesis → repeat
- Lessons from the debugging examples: agents give plausible-but-wrong explanations constantly; instructions get ignored — use hard boundaries like tests; stale sources of truth break workflows; agents need access to their own results
- **Start a startup today**: turn firsthand frustrations into first-dollar ideas
- Structure the offer around an outcome, not a product ("I am making [product] for [person] who [wants an outcome]. It gives them [result/outcome] for [price].")
- Build barely-good-enough, post publicly, and learn from a real response (payment or rejection > private polishing)
- Long-running work needs a target and a feedback signal (gauntlet prompts: clear target, progress signal, self-inspection, criticism, stopping condition)
- Two ways to work with a powerful agent: teach it your process, or manage it like an independent worker
- Homework: earn $1 online from a real customer

After Class 4 you should have:

- ☐ Debugged one failure and written up your findings: what was the cause, what was the fix, what happened, what did you learn?
- ☐ Publicly posted an offer for a barely-good-enough product you made during this class
- ☐ Collected your first $1 online by selling that offering to one person

Class 5: Case Studies, Gauntlet Loops, Ultracode & Credentialing

- Andrew's AI coding case study — how to produce a complex, polished artifact:
  - Build a rich skeleton first (simplest possible structure for review)
  - Check intent and organization/structural details before fleshing out
  - Share emotions alongside concrete instructions
  - Don't let the model invent extra unnecessary work/content
  - Transform the representation of the work when the work is hard to evaluate
  - Build WYSIWYG interfaces for giving feedback (e.g. an HTML redlining tool)
  - Think about content feedback separately from design/structural feedback
  - Store repeated procedures as skills (the printing skill)
  - Signpost changes of intent when redirecting
  - Break large builds into reviewable sections
  - Keep the source of truth simple (Markdown → site)
  - Design the project to survive compaction (state in files)
- Gauntlet Loops (Claude-of-Duty example): one prompt, ~24 hours, decompose → fan out to subagents → harsh critic vs. an external quality bar → send the biggest gap back through another round
- Close back-and-forth vs. letting the agent set its own direction; meta-agents that review the run history to improve the next iteration
- ultracode walkthrough: starting ultracode in Claude Code / desktop / Codex
- Ultracode vs. manual subagents: the parent designs roles, writes an orchestration workflow, and manages design → implement → review → repair → verify
- `/goal` : attaching an observable outcome so runs continue until the condition is met
- Cost and duration scale with scope (bounded research vs. open-ended builds)
- Introduced glossary of terms and final exam

After Class 5 you should have:

- ☐ Launched an ultracode run and inspected how the parent decomposed the work; finished something you liked and enjoyed using ultracode
- ☐ Taken the practice test