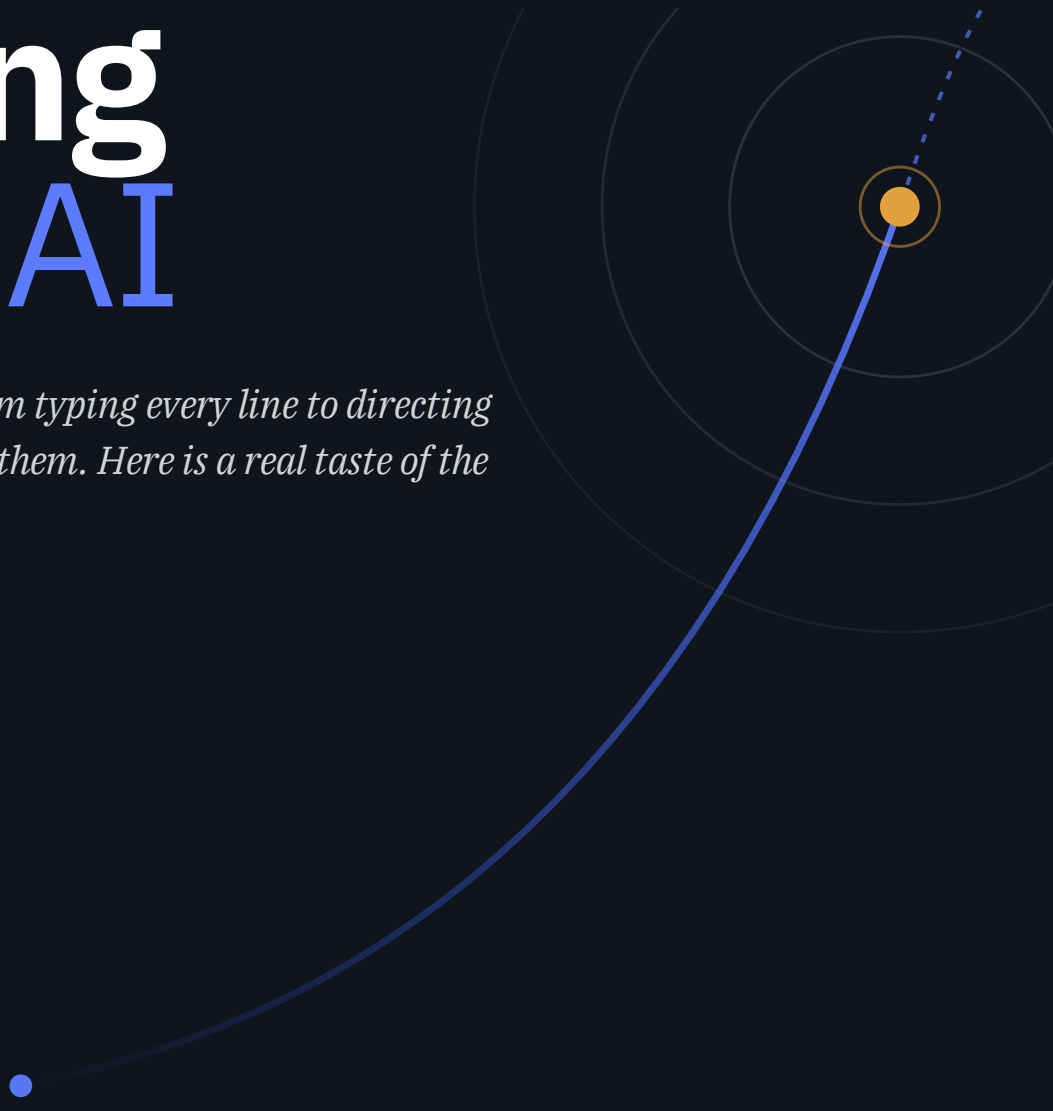
MASON PRESS · TECHNICAL SERIES

SNEAK PEEK EDITION
10 PAGES · V0.9
2026

A FIELD GUIDE FOR THE AGENT ERA

Coding with AI

Four tools. One shift: from typing every line to directing the machines that write them. Here is a real taste of the course.



WRITTEN BY

Max Mason

CH·01	CX·02	LV·03	AG·04
CLAUDE	CODEX	LOVABLE	ANTIGRAVITY

BEFORE YOU BEGIN

The job just changed.

For thirty years the developer's core motion was the same: think, then type. You held the whole system in your head and translated it into syntax, one character at a time. That motion is now optional. The tools in this course don't autocomplete your typing — they take a goal, plan it, write the code, run it, and hand you something to review.

Your new job is **direction**: describe intent clearly, give the agent the context it needs, and verify what comes back. Do that well and you move at a different speed. Do it badly and you ship confident nonsense. This teaser teaches the difference — with one genuine lesson from each of the four tools the full course covers.

00 / LOOP	**The agentic loop**	*the one skill that carries across every tool*	p.03
CH · 01	**Coding with Claude**	*context is the interface — writing memory that steers an agent*	p.04
CX · 02	**Coding with Codex**	*one agent, three surfaces — and when to send work away*	p.05
LV · 03	**Coding with Lovable**	*prompt to product — plan before you build*	p.06
AG · 04	**Coding with Antigravity**	*an IDE built for delegation — trust through artifacts*	p.07
MAP	**Which tool, when**	*a decision guide and the combined workflow*	p.08
FULL	**Inside the full course**	*every module, and what you'll build*	p.09

HOW TO READ THIS

Each tool section teaches one concrete technique you can use today, shows it running, and ends with a **field note** — the thing most people learn the hard way. The full course goes deeper on each.

LESSON 00 · FOUNDATIONS

The loop that runs under everything

Learn this once and every tool in the course becomes a variation on a theme.

Whether you're in a terminal, a browser app builder, or an agent-first IDE, an AI coding tool runs the same five-beat loop. The tools differ in how much of it they show you — but the loop is always there, and your leverage comes from steering each beat rather than watching it happen.

A. Spec beats prompt

A prompt asks for something. A spec defines *done*: the behaviour you want, the constraints that bound it, and how you'll know it's finished. "Add search" is a wish. "Add case-insensitive search over the title and tags fields, debounced at 300ms, with an empty-state message, and a test for the no-results case" is a spec. Agents are astonishingly literal — they reward specificity and punish vagueness with plausible guesses.

B. Context is the real interface

The chat box is not where the work happens. The *context* is — the repository, the examples, the house rules, the memory file the agent reads before it acts. Most "the AI is bad at this" moments are really missing-context moments. Teach the environment once and every later request inherits it.

C. Verify like you don't trust it

Because you shouldn't — not yet. Never merge a diff you haven't read. Make the agent *prove* its work with something you can check in seconds: a passing test, a rendered screen, a before/after. The goal isn't suspicion for its own sake; it's building justified trust, task by task.

D. Small steps, tight loop

One giant instruction produces one giant thing to untangle. A short loop — plan, act, verify, correct, repeat — keeps every change reviewable and every mistake cheap. Speed in this craft comes from small confident steps, not from one heroic prompt.

> FIELD NOTE
>
> The best operators sound almost boring: they narrate constraints, ask for a plan, and read the diff. The magic isn't a clever prompt — it's a disciplined loop. Keep it and any of the four tools ahead will feel like a superpower.

01

CH·01

Coding with Claude

Anthropic · terminal & desktop agent · reads your repo, plans, edits, tests

Claude Code lives where your code lives. You give it a task; it explores the repository, proposes a plan, makes changes across files, runs your tests, and iterates — with you approving as it goes.

The lesson: write the memory file first

Before you ask Claude to do anything, give it a `CLAUDE.md` at the root of your project. It's a plain-text briefing the agent reads automatically on every task — your stack, your conventions, the commands to run, and the things it must never do. This single file is the highest-leverage move in the whole module: it turns a capable stranger into someone who already knows how your team works.

CLAUDE.MD — PROJECT MEMORY

```
# Stack
Next.js 15 · TypeScript strict · Tailwind · Prisma/Postgres

# Commands
Test: npm run test Lint: npm run lint Types: npm run typecheck
```

Now the request stays short, because the standing context does the heavy lifting. Watch the loop from the last page play out — plan first, then act, then prove it:

CLAUDE — SESSION

```
› Move the date-formatting logic out of the invoice
  component into a tested util. Plan first.

Plan — 1. create lib/format-date.ts 2. add test
3. refactor Invoice.tsx to import it 4. typecheck
› go
```

FIELD NOTE

Use **plan mode** for anything non-trivial. Reviewing a five-line plan costs seconds and catches the wrong-direction task before it writes fifty files. Approve the plan, not just the result.

IN THE FULL MODULE →

Sub-agents for parallel work · reusable Skills · wiring MCP tools to your own services · a test-driven refactor of a real legacy module, start to finish.

02

CX·02

Coding with Codex

OpenAI · one agent across CLI, editor & cloud · open-source, terminal-first

Codex is OpenAI's coding agent. The same agent and account follow you across three surfaces — the terminal CLI, an editor extension, and a cloud workspace — so you pick the surface that fits the moment, not the tool.

The lesson: know which surface fits the moment

The instinct is to do everything in one place. Codex's strength is that you don't have to. The loop is identical everywhere; only the *venue* changes — and choosing the right venue is a real skill.

Like Claude, Codex reads a repo-level briefing — here it's `AGENTS.md` — plus reusable skills and MCP tools. Teach it your project once; every surface inherits it.

THREE SURFACES · ONE AGENT

CLI precise work inside a real codebase; lives in your terminal and CI.

EDITOR inline in VS Code / JetBrains while you read the code.

CLOUD fire off a long task; it runs on remote infra while you move on.

The move most people miss: **delegate long jobs to the cloud**. A wide, mechanical task — "update this deprecated API call across the whole repo and open a PR" — doesn't need to tie up your laptop or your attention. Send it away; review the result when it's done.

CODEX — APPROVAL WORKFLOW

```
$ codex "replace all uses of the old
  fetchUser() with getUser(); update tests"

proposed — 11 files · 34 insertions · 34 deletions
apply these changes? [y] / n / show diff
✓ applied · npm test → 128 passing
```

FIELD NOTE

Keep Codex in an **approval-gated sandbox** for anything that touches the shell. Watching it ask before it runs a command — and saying no once in a while — is how you learn exactly what it's about to do. Autonomy is earned per task, not switched on globally.

IN THE FULL MODULE →

Writing an `AGENTS.md` that survives real projects · sandbox & permission tuning · running several cloud tasks in parallel · reviewing agent PRs without rubber-stamping.

LV·03

03 Coding with Lovable

Full-stack app builder · prompt → live web app · frontend, database, auth, deploy

Describe an app in plain English and Lovable generates a working full-stack web application — React frontend, database and auth on the back, deployed to a live URL — then lets you refine it by chat, by visual editing, or in the code itself.

The lesson: plan the app before you build it

The fastest way to waste an afternoon in Lovable is to type "build me a marketplace" and start accepting whatever appears. The discipline that separates a demo from a shippable MVP is the same one from Lesson 00: **plan first**. Use Plan mode to agree the data model and the screens *before* a line of code is generated — then build in small, nameable slices.

LOVABLE — PLAN MODE

```
you — A booking app for a yoga studio. Classes,
  members, bookings. Members sign in and book a spot.

plan — Tables: classes, members, bookings
Screens: schedule · class detail · my bookings · admin
Auth: email sign-in · Roles: member, admin
you — approve. Build the schedule + booking flow first.
```

Because Lovable writes real, exportable code and syncs to GitHub, a plan-first project doesn't trap you. When you outgrow the builder, you hand the repository to a terminal agent from Module 01 or 02 and keep going.

WHERE IT SHINES

0→1 apps, MVPs, internal tools, landing pages, standard CRUD-and-auth patterns — built and shared in an afternoon, no local setup.

KNOW THE CEILING

Deeply custom logic and large apps strain any builder. Treat generated code as a strong start to review and secure — not an untouched finished system.

IN THE FULL MODULE →

Prompting for data models that hold up · connecting auth, payments and storage · the visual Design view and reusable themes · shipping to a custom domain, then exporting to GitHub to keep building.

AG·04

04 Coding with Antigravity

Google · agent-first IDE · orchestrate agents across editor, terminal & browser

Antigravity is an IDE built the other way round: agents are the primary actors and you are mission control. From a Manager surface you dispatch agents on whole tasks, and they report back not with raw tool output but with reviewable artifacts.

The lesson: review the artifact, not the transcript

Delegating real work requires trust, and scrolling a thousand lines of tool calls doesn't build it. Antigravity's answer is to have each agent produce **artifacts** — a task list, an implementation plan, screenshots, a short browser walkthrough of the feature actually working. You review the deliverable a human would review, which lets you supervise several agents at once instead of babysitting one.

```
MANAGER — AGENTS IN FLIGHT
● agent-1 checkout bug · reproduced → fix → test [artifact: recording]
● agent-2 dark-mode pass · 8 screens [artifact: screenshots]
○ agent-3 awaiting your review of plan.md

You: architect & reviewer · agents: execute & verify
```

The other habit worth building: **set autonomy to match risk**. Boilerplate scaffolding can run agent-driven and unattended; a payment or auth change should stay review-driven, approved step by step. Antigravity is designed to mix these per task — the tool bends to your risk tolerance, not the reverse.

> **FIELD NOTE**
>
> Because it's model-flexible, you can run different models per task inside one project — Gemini for one agent, another model for the next. Don't over-tune early: pick a capable default, and switch only when a task has a clear reason to.

IN THE FULL MODULE →

Structuring a task so an agent can verify it · reading and trusting artifacts · orchestrating parallel agents from the Manager · the async workflow — dispatch, review, redirect — without losing the thread.

THE MAP

Four tools, one workflow

These aren't rivals to pick between once. They're stations on a single flight — most real projects touch several.

REACH FOR	WHEN THE TASK IS	ITS NATURAL STRENGTH
CH·01 **Claude**	precise work inside an existing codebase — refactors, tests, features	deep repo reasoning and a disciplined plan → act → verify loop
CX·02 **Codex**	terminal-native work, or long jobs you want to send away	one agent across CLI, editor and cloud; strong delegation
LV·03 **Lovable**	0→1 — a full-stack web app or MVP from nothing	prompt-to-product speed; frontend, database, auth, deploy
AG·04 **Antigravity**	larger efforts where you orchestrate and supervise agents	agent-first IDE; review-by-artifact; parallel agents

A workflow that uses all four

Here is how a single product might move through them — the point of the course is not four tools in isolation, but the judgment to route each task to the right one.

LV·03
Prototype
Stand up the MVP in Lovable in an afternoon. Validate the idea, export to GitHub.

›

CH·01
Harden
Pull the repo into Claude. Add tests, tighten the data model, refactor the rushed parts.

›

CX·02
Scale work
Send wide, mechanical changes to Codex cloud while you keep building locally.

›

AG·04
Operate
Run ongoing fixes and features as supervised agents from Antigravity's Manager.

FIELD NOTE

Don't marry a tool. The skill the market actually pays for is **routing**: reading a task and knowing which surface will finish it fastest with the least risk. That judgment is what the full course drills.

WHAT YOU JUST TASTED IS ONE PAGE PER MODULE

Inside the full course

Five modules and a capstone. Every technique in this teaser, taught properly — and put to work on real software.

00	**Foundations**	*the agentic loop · specs · context engineering · verification & trust*	4 lessons
CH·01	**Coding with Claude**	*CLAUDE.md · plan mode · sub-agents · Skills · MCP · a real TDD refactor*	7 lessons
CX·02	**Coding with Codex**	*CLI · editor · cloud · AGENTS.md · sandboxing · parallel delegation*	7 lessons
LV·03	**Coding with Lovable**	*plan-first apps · data models · auth & payments · design view · ship & export*	6 lessons
AG·04	**Coding with Antigravity**	*agent-first IDE · artifacts · autonomy levels · orchestrating parallel agents*	6 lessons
CAP	**Capstone**	*build and ship one real product — routing each task across all four tools*	project

By the end, you'll be able to

— Turn a vague idea into a working, deployed app in a single sitting.

— Direct an agent through a real refactor of code you didn't write.

— Write context files that make any agent behave like a teammate.

— Delegate long-running work and review it without rubber-stamping.

— Supervise several agents at once through artifacts, not transcripts.

— Route any task to the tool that finishes it fastest, at least risk.

> WHO THIS IS FOR
>
> Developers modernising how they work, and capable non-developers who want to build real software. No tool is assumed; every one is taught from first principles — but the pace respects that you're smart.

A NOTE FROM THE AUTHOR

The keyboard isn't going anywhere. Your job is.

I wrote this course because the tools got good faster than our habits did. Most people using AI to code are still typing at a rocket. The developers pulling ahead aren't the ones with the cleverest prompts — they're the ones who learned to *direct*: to specify, to hand over context, and to verify with a cool head. That's a learnable skill, and it's the same skill whether you're in a terminal, a browser, or an agent-first IDE.

This teaser was ten pages. The full course is where it becomes muscle memory — hands on the four tools, building something real. If the loop on page three made something click, you're ready for it.

Get the full course

Five modules, a hands-on capstone, and the four tools taught in depth.
Reserve your seat for the next cohort.

JOIN THE WAITLIST →

maxmason.dev/coding-with-ai

Max Mason
AUTHOR · CODING WITH AI · MASON PRESS TECHNICAL SERIES