A FRAMEWORK FOR DEVELOPERS WHO ARE DONE WATCHING

The Tutorial Hell Escape Plan

A practical, step-by-step system for self-taught developers stuck in an endless loop of courses and tutorials — with a framework, a self-diagnostic, and a 14-day challenge to get you actually building.

haas.dev
Thinking framework over tutorials.

BEFORE YOU START

If you've finished 6 courses and still freeze on a blank file…

…this is for you. Not because you're not smart enough, and not because you haven't worked hard. You've probably worked harder than most people who already have a job. The problem isn't effort. It's that watching someone else build something and being able to build something yourself are two completely different skills — and almost nothing about the way courses are taught trains the second one.

This isn't another list of "100 project ideas." It's a framework for actually building the skill that tutorials quietly let you skip: sitting with a problem you don't know how to solve yet, and solving it anyway.

WHAT'S INSIDE

- What tutorial hell actually is, and why it happens to disciplined, capable people
- A self-diagnostic to find out how deep in it you are
- The Build Loop — a 5-stage framework to replace "follow along" with "figure it out"
- The relapse triggers that pull people back in, and how to catch them
- A 14-day challenge to run the framework for real, twice
- A no-excuses way to pick what to build, without another project idea list

SECTION 1

What Tutorial Hell Actually Is

Tutorial hell is the cycle of learning to code by continuously watching, following along with, or completing structured courses — without a matching amount of time spent building something with no one guiding your hand. You understand the material while it's on screen. You could probably explain it back. But open a blank file with no instructions, and the same knowledge doesn't seem to be there anymore.

That gap isn't a memory problem. It's the difference between two things that feel identical while you're learning but are not the same skill at all:

Following a tutorial	Building on your own
Recognizing the next step when it's shown to you	Generating the next step yourself
Debugging with the answer already visible	Debugging with no idea what's wrong
Deciding between options someone already narrowed	Deciding what options even exist
Comprehension — does this make sense?	Recall & synthesis — can I produce this?

Tutorials are optimized to make the left column feel good — smooth, confidence-building, never stuck for long. That's not a flaw in tutorials. It's what makes them good at teaching concepts. But it also means the right column, the actual job of a developer, never gets practiced unless you deliberately go build it somewhere else.

“I understood it perfectly while watching” is not evidence you've learned it. It's evidence the video was well made.

SECTION 2

How Deep In It Are You?

Answer honestly. This isn't a judgment — it's a starting point. Give yourself 1 point for every statement that's true for you right now.

- [ ] I've completed more than 3 courses/tutorials in the language or framework I'm trying to learn.
- [ ] I can follow along with a tutorial easily but freeze when starting a project from a blank file.
- [ ] I often start a new course before finishing or applying the last one.
- [ ] When I get stuck on my own project, my first instinct is to look for a tutorial on the exact thing I'm stuck on, rather than trying to work it out.
- [ ] I've never deployed or shared something I built entirely without following a guide.
- [ ] I feel like I "understand" far more than I can actually produce without help.
- [ ] I've told myself "I just need one more course before I'm ready to build something real."

YOUR SCORE

0–1	**Mostly clear.**	You're building more than you're consuming. Use this pack to sharpen the habit, not fix a crisis.
2–4	**Moderate tutorial hell.**	The pattern is there but not locked in. The Build Loop on the next pages will break it fast.
5–7	**Deep tutorial hell.**	Completely normal, very common, and very fixable. Start with Stage 1 today — not after “one more course.”

SECTION 3

The Build Loop

A 5-stage framework that replaces “follow along” with “figure it out.” Run it once, start to finish, on one small thing before you touch another course.

01 Pick a Broken Problem

Not a project idea from a list — something you actually want fixed

Skip the “50 beginner project ideas” list. Those projects are already solved a thousand times over, so the moment you get stuck, a full tutorial for that exact project is one search away — and you'll take it.

Instead, find something small and mildly annoying in your own life that code could fix: a spreadsheet you keep updating by hand, a way to track something you care about, a tool that doesn't exist yet for a habit you have. It doesn't need to be original. It needs to be yours, so no exact tutorial for it exists.

> **DO THIS:** Write down one small, mildly annoying problem in your own life or work that you could plausibly fix with code. Keep the scope small enough to finish in a weekend.

02 Set a No-Tutorial Timer

Give yourself permission to search, not permission to follow

Set a timer — 60 to 90 minutes is enough. For that block, you can search documentation, search error messages, and search “how does X work” conceptually. What you can't do is search for or follow a step-by-step build-along for your specific project.

This isn't about struggling alone forever. It's about separating “looking something up” (a real skill you'll use your whole career) from “following someone else's steps” (the habit you're trying to break).

> **DO THIS:** Block 60–90 minutes on your calendar this week, labeled “build, no tutorials.” Treat it like a meeting you can't skip.

03 Get Stuck on Purpose

The struggle isn't a sign you're doing it wrong — it's the actual exercise

You will get stuck during that timer. That's not a problem to route around — it's the entire point. Debugging your own confusion, with no answer key, is the specific muscle tutorials never make you use.

When you feel the urge to search for a full walkthrough, that's the exact moment the skill is being built. Sit in it for at least 10 more minutes before you do anything else.

> **DO THIS:** When you get stuck, set a second, smaller timer — 10 minutes — before you're allowed to look anything up beyond documentation or an error message.

04 Reverse-Engineer One Reference

Look it up after you've struggled, then compare — don't copy

After genuinely trying, it's fine to look at one reference: documentation, a similar (not identical) example, or a forum answer. The rule is what you do with it: read it, close it, and then write your own version from memory. If you can't, you copied instead of learned — and that's useful information, not a failure.

This single habit — look, close, rewrite from memory — is the difference between a reference making you better and a reference making you dependent on it.

> **DO THIS:** Next time you look something up mid-project, close the tab before you type the fix, and write it from what you just read.

05 Ship It Ugly

Unfinished-but-shared beats polished-but-private, every time

Deploy it, push it, or show it to one person — even if it's rough, even if half the features are missing. The goal isn't a portfolio piece. It's crossing the threshold from “code that only exists on my laptop” to “something real in the world,” which is a psychological line that matters far more than the code quality on either side of it.

Most tutorial-hell code never ships, which means it never has to survive contact with reality — real users, a real deploy process, real edge cases. Shipping, even badly, is where the next set of real skills starts.

> **DO THIS:** Push it to GitHub with a real README, or deploy it somewhere live — today, not “once it's done.”

SECTION 4

The Relapse Triggers

Tutorial hell doesn't end in one clean break — it pulls people back in through a few predictable moments. Recognizing them in the moment is most of the fix.

THE THOUGHT	WHAT'S REALLY HAPPENING	WHAT TO DO INSTEAD
“I should learn the fundamentals properly first.”	Often a comfortable way to delay the discomfort of Stage 3.	Ask: would one more course actually teach me something Stage 3 wouldn't? If not, it's avoidance.
“Let me just watch how a pro does this one part.”	One “just this part” tutorial quietly turns into following the whole build.	Set a hard rule: you may watch the concept explained, never watch it being built.
“I don't know enough to build this yet.”	Almost always false — the gap is confidence built by tutorials, not actual missing knowledge.	Start the 10-minute stuck-timer anyway. You'll usually find you knew more than you thought.
“This isn't good enough to show anyone.”	Perfectionism protecting you from the vulnerability of Stage 5.	Ship it privately first, to one trusted person, if showing it publicly feels too exposed.

SECTION 5

The 14-Day Build Loop Challenge

Run the full framework twice, back to back, with a slightly bigger problem the second time. Two full loops is usually enough for the pattern to start feeling normal instead of forced.

DAY	FOCUS
1	Pick your problem (Stage 1). Write it down in one sentence.
2–3	First no-tutorial build blocks (Stages 2–3). Expect to feel stuck — that's correct.
4	Reverse-engineer one reference if you're genuinely blocked (Stage 4).
5–6	Keep building. Resist starting a course “to catch up” — that's a relapse trigger.
7	Ship it, however rough (Stage 5). This is the end of Loop 1.
8	Pick a second problem — slightly harder than the first (Stage 1, Loop 2).
9–12	Build blocks for Loop 2. Notice: does getting stuck feel less scary this time?
13	Reverse-engineer a reference only if truly needed (Stage 4).
14	Ship Loop 2. Compare both projects — not on polish, on how it felt to build them.

If you miss a day, don't restart the whole 14 days — just pick the loop back up. The habit you're building is returning to it, not a perfect streak.

SECTION 6

“But I Don't Know What to Build”

This is usually the real blocker before Stage 1 even starts. Instead of another project list, answer these four prompts — the overlap between them is almost always a project worth building.

What do you track by hand?

A spreadsheet, a notes app, a sticky note — anything you're manually updating is a candidate for a small tool.

What do you look up the same way, repeatedly?

A recurring search, a recurring calculation, a recurring lookup — repetition is a signal something could be automated.

What's mildly annoying about a tool you already use?

Not a redesign of the whole tool — one missing feature you wish it had, built as its own small thing.

What do you wish existed for a hobby or habit of yours?

Personal interest projects tend to hold your attention through the stuck moments better than generic ones.

Whatever comes out of this, keep it small. The right first project is boring enough to finish in a weekend, not exciting enough to still be “in progress” in six months.

YOU'RE READY

Close this PDF. Open a blank file.

That's really the whole plan. Not another course, not another list — one small, real problem, one no-tutorial timer, and permission to be stuck for a while before you're allowed to look anything up.

haas.dev exists for exactly this: building the engineering mindset that tutorials skip over. More frameworks, prompts, and practical resources for developers who are ready to build are published there regularly.

— haas.dev
Thinking framework over tutorials.