Full AI for the People Book

The complete free path: safe AI foundations, developer setup, coding basics, AI-assisted coding, first AI API app, RAG/tool-calling basics, tests, deployment checks, and portfolio capstone.

Back to AI for the People | Download PDF | Download Markdown

Previous resource | Next resource

Full AI for the People Book

AI for the People — Day One to AI Developer

A free, beginner-safe book for becoming an AI developer from the ground up

Start with one safe AI workflow. Finish with small AI apps, tests, review gates, RAG basics, agent patterns, deployment checklists, and a portfolio capstone.

AI built by people, for people.

This book is the free public side of DWAI. It is not a teaser and it is not just a prompt pack. It is a path: from day one, through developer foundations, into AI-assisted coding, AI app patterns, responsible agents, evaluation, and safe shipping habits.

It does not promise a job, income, instant mastery, productivity, business results, or a guaranteed personal outcome. It gives you a practical curriculum, projects, review gates, and a way to keep learning without handing authority to AI.

Before you paste anything into an AI tool: use placeholders for names and private details. Do not paste passwords, API keys, 2FA codes, recovery codes, private tokens, card details, medical records, legal documents requiring advice, regulated financial records, customer data, or

sensitive account access.

How to use this book

Do not try to become advanced by skipping the basics. The path is layered:

1. **Use AI safely** — learn drafts, review, owner gates, and no-secrets habits.
2. **Think like a developer** — folders, files, Terminal, editor, Git, debugging.
3. **Write small programs** — HTML, CSS, JavaScript/TypeScript, Python, JSON, HTTP.
4. **Use AI as a coding partner** — specs, diffs, tests, explanations, reviews.
5. **Build AI features** — model calls, structured outputs, retrieval, tools, evals.
6. **Ship safely** — privacy, cost controls, logs, rollback, and owner approval.
7. **Prove it with projects** — a portfolio that shows what you can build and verify.

If a chapter feels too hard, mark HOLD, reduce the task, and repeat the smallest working version. That is what developers do.

What “AI developer” means here

An AI developer is not someone who blindly asks a model to “make an app.” An AI developer can:

- define a user problem and acceptance criteria;
- set up a private project workspace;
- read and edit code with AI help without losing ownership;
- use Git to track changes and recover mistakes;
- call APIs safely with environment variables;
- build simple web, script, or automation projects;
- add AI features such as prompts, structured outputs, retrieval, and tools;
- test outputs instead of trusting vibes;
- protect secrets, users, data, costs, and public actions;
- document what the system does, what it does not do, and how to maintain it.

The goal is not to memorize everything. The goal is to become a builder who can learn, test, debug, and ship small useful AI systems responsibly.

Part I — Day one: safe AI foundations

Chapter 1 — The promise and the boundary

AI can help you learn faster, draft faster, compare options, explain code, write small programs, review mistakes, and design workflows. It cannot take ownership for you.

Your first developer habit is not coding. It is **owner control**.

Use this rule everywhere:

> AI drafts and explains. I decide, test, approve, and own the result.

Your first artifact:

```
# My AI Developer Boundary Note
I am using AI to learn and build.
I will not paste secrets, passwords, API keys, 2FA codes, customer data, private
records, or sensitive account access into chat.
I will not let AI spend money, publish, send, delete, deploy, change accounts, or
make commitments without explicit owner approval.
I will test important outputs before trusting them.
```

Chapter 2 — Your first 20-minute AI workflow

Before code, learn the loop. Open ChatGPT, Claude, Gemini, or another AI chat tool and choose one non-sensitive task: organize notes, explain a concept, plan a tiny project, research a tool, or draft a README.

Paste this:

```
I am new to using AI as a system. Help me build one tiny routine from a real task.

My task:
[turn messy notes into a plan / draft an email / research a decision / learn a topic]

My messy input:
[paste non-sensitive notes, or replace private details with placeholders]

My goal:
[what I want back]

My red lines:
Do not send messages, post publicly, spend money, change accounts, delete anything,
make commitments, make final decisions for me, ask for secrets, or continue if my
input contains sensitive/private material that should be replaced with placeholders.
Do not make therapy, counselling, diagnosis, treatment, medical, legal, financial,
tax, crisis, regulated-advice, or guaranteed-outcome claims.

Please give me:
1. a plain-English name for this workflow;
2. the inputs you need next time;
3. a useful first draft or organized output;
4. a checklist I can use to review it;
5. anything risky that needs owner approval;
6. one sentence I should add to my context card for next time.
```

Review the output with three labels:

- **PASS** — safe enough to use after your review.
- **HOLD** — missing context, risky action, or owner decision required.
- **FIX** — useful direction, but needs revision before you use it.

The developer habit is the loop: ask, inspect, test, update.

Chapter 3 — Learn with AI without outsourcing understanding

Do not ask AI to “teach me everything.” Ask for a loop:

```
Teach me this like a practical system.

Topic/workflow: [what I want to learn]
Outcome I want: [what I should be able to do]
My current level: [beginner/intermediate/what I already know]
Constraints: [time, tools, budget, platform, risks]

Please:
1. explain the mental model;
2. show the workflow step by step;
3. give 3 examples;
4. quiz me to find gaps;
5. correct my understanding;
6. give me one small practice task;
7. define how I verify whether I did it correctly;
8. suggest how to update my notes/prompts after the attempt;
9. keep risky actions draft-only and flag anything that needs owner approval.

Do not make therapy, counselling, diagnosis, treatment, medical, legal,
financial, tax, crisis, regulated-advice, or guaranteed-outcome claims.
```

For every topic in this book, use the same pattern:

1. Ask for the mental model.
2. Ask for the smallest working example.
3. Predict what will happen before running it.
4. Run or inspect it.
5. Explain the result back in your own words.
6. Ask AI to quiz you.
7. Update your notes.

You are not learning if you cannot explain the output without the AI.

Chapter 4 — Owner gates for future developers

Developer work touches risky surfaces quickly. These require owner approval:

- installing software;
- creating accounts or paid plans;
- using API keys or environment variables;
- changing Git remotes or pushing code;
- making a repo public;

- deploying to Vercel, Netlify, Cloudflare, Firebase, Supabase, Render, Railway, AWS, Google Cloud, Azure, or any provider;
- changing DNS/domains;
- sending emails, DMs, comments, or public posts;
- deleting files, databases, buckets, repos, or deployments;
- using real customer/private data;
- running commands you do not understand, especially `sudo`, `rm -rf`, recursive permission changes, or install scripts piped into a shell.

If unsure, HOLD.

Chapter 5 — The developer learning log

Create one notes file. Call it `LEARNING-LOG.md`.

Use this structure:

```
# Learning Log

## Today's topic

## What I tried

## What worked

## What failed

## Error messages I saw

## What I think the error means

## One thing I can now explain

## One thing still unclear

## Owner gates / risks noticed
```

A full AI developer is built by repeated logs, not by one perfect tutorial.

Part II — Developer foundations from zero

Chapter 6 — Set up your developer workspace

A beginner-safe workspace has four layers:

1. **A project folder** — one place for files.
2. **An editor** — usually VS Code or another code editor.
3. **A terminal** — the command line for running tools.
4. **Version history** — Git, so you can see and recover changes.

Suggested local folder structure:

```
ai-learning/
  00-notes/
    LEARNING-LOG.md
    CONTEXT.md
  01-html-css-js/
  02-python/
  03-api-basics/
  04-ai-api-app/
  05-rag-demo/
  06-agent-demo/
  07-capstone/
```

Do not put API keys, passwords, private exports, client data, or personal records inside practice folders.

Chapter 7 — Terminal basics without fear

The Terminal lets you move through folders and run tools. Learn these ideas first:

- `pwd` means “where am I?”
- `ls` means “what is here?”
- `cd folder-name` means “move into this folder.”
- `mkdir folder-name` means “make a folder.”

- `touch file-name` means “make a blank file.”
- `cp source destination` copies.
- `mv source destination` moves or renames.

Beginner safety rules:

- Do not run a command you cannot explain in plain English.
- Ask AI to explain commands before you run them.
- Avoid `sudo` until you know why elevated permission is required.
- Avoid `rm -rf` in tutorials unless you fully understand the path and have a backup.
- Avoid copy-pasting install scripts from random websites.

Practice task:

1. Create a folder called `terminal-practice`.
2. Create `notes.md` inside it.
3. Write one sentence in the file using your editor.
4. Use Terminal to show where the file lives.
5. Log what each command did.

PASS means you can explain the folder path and undo the test safely.

Chapter 8 — Editor basics: VS Code or equivalent

Your editor is where code becomes visible. Learn these moves:

- open a folder, not just one file;
- use the file tree;
- search across files;
- open the integrated terminal;
- format a file;
- read errors in the Problems panel;
- compare changes before accepting AI edits.

Ask AI:

```
I am new to VS Code. Explain the file tree, integrated terminal, search, source
control, and Problems panel like I am learning to build AI apps. Give me one
10-minute practice task and a PASS/HOLD checklist.
```

Practice task: open your `ai-learning` folder, create `CONTEXT.md`, and write:

```
# Project Context
I am learning to become an AI developer.
I prefer small steps, explanations, tests, and owner gates before risky actions.
```

Chapter 9 — Git and GitHub without making things public

Git tracks changes. GitHub can store a remote copy. You do not need a public repo to learn. Private first is the default.

Learn these Git concepts:

- **repository** — a folder with history;
- **commit** — a saved checkpoint;
- **diff** — what changed;
- **branch** — a separate line of work;
- **remote** — a copy somewhere else, such as GitHub;
- **`.gitignore`** — files Git should not track, especially `.env`.

First Git practice:

```
Create a private practice repo. Add README.md, CONTEXT.md, and LEARNING-LOG.md.
Commit only safe files. Do not push public. Do not include API keys or .env files.
```

Before every commit, ask:

- Did I inspect the diff?
- Did I accidentally include secrets?
- Does the commit message explain the change?
- Is the repo private if pushed?
- Would I be okay if this file became public by mistake?

Chapter 10 — HTML, CSS, and JavaScript for your first web surface

You can build an AI app without being a designer, but you need the basics of the web.

Mental model:

- **HTML** gives structure.
- **CSS** gives style and layout.
- **JavaScript** gives behavior.

Project 1: build a local “AI Learning Dashboard” page.

Required files:

```
01-html-css-js/
  index.html
  styles.css
  app.js
  README.md
```

Features:

- a heading;
- a textarea for a learning note;
- a button that adds the note to a list;
- a PASS/HOLD selector;
- no external accounts, no API keys, no deployment.

Acceptance criteria:

- page opens locally;
- notes can be added;
- no secrets are requested;
- README explains how to run it;
- you can explain what each file does.

Chapter 11 — JavaScript and TypeScript foundations

JavaScript makes web pages interactive. TypeScript adds types so mistakes are caught earlier.

Core ideas:

- values: strings, numbers, booleans, arrays, objects;
- functions: reusable actions;
- conditions: if this, then that;
- loops: repeat work;
- async/await: wait for network or file operations;
- types/interfaces: define the shape of data;
- errors: what can go wrong and how to handle it.

Minimal runnable JavaScript example:

```
const entries = [
  { note: "AI suggested a safer wording for my bio" },
  { note: "AI suggested changing billing settings" },
  { note: "AI helped explain a JavaScript error" },
];

function classifyEntry(entry) {
  const riskWords = ["billing", "password", "secret", "deploy", "delete", "publish"];
  const text = entry.note.toLowerCase();
  const risky = riskWords.some((word) => text.includes(word));

  return {
    note: entry.note,
    verdict: risky ? "HOLD" : "PASS",
    reason: risky ? "Owner approval required before action." : "Safe as draft/review work.",
  };
}

console.log(entries.map(classifyEntry));
```

Run it with Node:

```
node app.js
```

Tiny TypeScript upgrade:

```ts
type Verdict = "PASS" | "HOLD";

type Review = {
  note: string;
  verdict: Verdict;
  reason: string;
};
```

AI-assisted practice prompt:

```
Teach me JavaScript objects and functions using an AI learning log example.
Give me a tiny example, then ask me to predict the output before showing the
answer. Then give me one bug to fix and a PASS/HOLD checklist.
```

You are ready to move on when you can read a small function, run it, change one line, and explain inputs, outputs, and possible failures.

Chapter 12 — Python foundations

Python is useful for scripts, data processing, automation, AI prototypes, and API experiments.

Core ideas:

- variables and types;
- lists and dictionaries;
- functions;
- reading and writing files;
- JSON;
- command-line scripts;
- virtual environments;
- installing packages carefully;
- handling errors.

Project 2: build a local learning-log summarizer without AI API calls.

Create `summarize_log.py`:

```
from pathlib import Path
import re
import sys


def summarize(path: Path) -> dict:
    if not path.exists():
        raise FileNotFoundError(f"Missing file: {path}")

    text = path.read_text(encoding="utf-8")
    words = re.findall(r"\w+", text)
    headings = [line.strip("# ") for line in text.splitlines() if line.startswith("#")]

    return {
        "file": str(path),
        "lines": len(text.splitlines()),
        "words": len(words),
        "headings": headings,
    }


def write_summary(result: dict) -> Path:
    output = Path("summary.md")
    heading_lines = "
".join(f"- {heading}" for heading in result["headings"]) or "- None"
    output.write_text(
        f"# Summary

"
        f"File: {result['file']}

"
        f"Lines: {result['lines']}

"
        f"Words: {result['words']}

"
        f"## Headings
{heading_lines}
",
        encoding="utf-8",
    )
    return output


def main() -> None:
    if len(sys.argv) != 2:
        raise SystemExit("Usage: python3 summarize_log.py learning_log.md")
    result = summarize(Path(sys.argv[1]))
```

```
    output = write_summary(result)
    print(f"Wrote {output}")


if __name__ == "__main__":
    main()
```

Create `learning_log.md`:

```
# Day 1
I learned what PASS and HOLD mean.

## Mistake
I almost pasted private context. I used placeholders instead.
```

Run:

```
python3 summarize_log.py learning_log.md
```

Acceptance criteria:

- script runs locally;
- no API keys;
- `summary.md` is created;
- missing files show a clear error;
- one test input and one expected output are documented;
- you can explain every function.

Chapter 13 — JSON, HTTP, and APIs

AI apps are mostly data moving between systems.

Learn these terms:

- **JSON** — common data format for apps and APIs.
- **HTTP** — the request/response protocol of the web.
- **GET** — ask for data.
- **POST** — send data to create/process something.
- **headers** — metadata such as content type or authorization.
- **status codes** — 200 means success, 400/401/403/404/500 mean different failures.

- **API key** — a secret token that must not be pasted into public code or chat.
- **rate limit** — the provider says “slow down.”

Practice with a public no-auth API or a local mock first. Do not start with paid AI keys.

Your API checklist:

- What URL am I calling?
- What method is used?
- What data is sent?
- What data returns?
- What errors can happen?
- Does this require a key?
- Where will logs store data?
- What is the cost or rate limit?

Chapter 14 — Debugging and tests

A developer is not someone who never sees errors. A developer is someone who can investigate errors calmly.

Debugging loop:

1. Read the full error.
2. Reproduce it.
3. Identify what changed.
4. Reduce to the smallest example.
5. Form one hypothesis.
6. Test one change.
7. Log the result.

Testing loop:

1. Define expected behavior.
2. Write a small test case.
3. Run it.
4. Fix until it passes.
5. Keep the test so the bug does not return.

Ask AI for help like this:

```
Here is the full error message and the smallest code example. Do not guess a fix
yet. First explain what the error means, what line matters, what assumptions you
need to check, and the smallest test I should run next.
```

Never accept “try this” if neither you nor the AI can explain why.

Part III — AI-assisted coding workflow

Chapter 15 — Specs before code

AI writes better code when the target is clear. Before asking for code, write a small spec.

Spec template:

```
# Project Spec

## User problem

## Inputs

## Outputs

## Constraints

## Non-goals

## Safety / owner gates

## Acceptance criteria

## Test cases

## Files expected
```

A weak request says: “Build me an AI app.”

A better request says: “Create a local-only note classifier that reads sample notes, labels them PASS/HOLD/FIX, writes JSON output, includes tests, and does not call external APIs.”

Chapter 16 — How to ask AI for code responsibly

Use AI as a pair programmer, not an owner.

Good coding prompt:

```
You are helping me learn. First explain the plan and file changes. Do not write
code until the plan is clear. Keep the project local-only. Avoid secrets, public
pushes, deploys, account changes, and destructive commands. Include tests or a
manual verification checklist. After code, explain the diff in beginner language.
```

Code review checklist:

- Do I understand the purpose?
- Are file changes limited to the project folder?
- Are secrets avoided?
- Are errors handled?
- Are tests or verification steps included?
- Does the README explain how to run it?
- Did AI add packages I did not approve?
- Did AI introduce network calls, file deletion, account changes, or deployment?

Chapter 17 — Read diffs before accepting changes

A diff shows what changed. Learn to read it before trusting AI-generated code.

Look for:

- new files;
- deleted files;
- package/dependency changes;
- network calls;
- hard-coded secrets;
- permissions changes;
- external sends;
- deploy commands;
- hidden complexity;
- TODOs that hide missing work.

Practice prompt:

```
Review this diff like a cautious AI developer. Separate: expected changes,
unexpected changes, security risks, tests missing, docs missing, and PASS/HOLD/FIX
verdict. Do not approve deployment or public push.
```

Chapter 18 — Build a tiny command-line tool

Project 3: `ai-task-sorter`.

Goal: a local script that reads a text file of tasks and sorts them into:

- `do_now`;
- `schedule`;
- `ask_owner`;
- `hold_risky`.

No AI API yet. Use rules first so you understand the workflow.

Acceptance criteria:

- sample input file exists;
- output JSON exists;
- risky words like “delete,” “pay,” “publish,” “deploy,” “password,” and “API key” are classified as `hold_risky`;
- README explains how to run;
- at least three test examples pass.

This teaches the pattern before adding a model.

Chapter 19 — Build a tiny web/API app

Project 4: local “review queue.”

Goal: a small web page or local API that lets you paste a draft and mark it PASS, HOLD, or FIX.

Acceptance criteria:

- works locally;

- stores only sample data or local demo data;
- no login, no public deploy, no real customer information;
- user can add a draft;
- user can choose PASS/HOLD/FIX;
- README explains how to run and what is intentionally missing.

This is the bridge from “I can write code” to “I can build a small tool.”

Chapter 20 — Documentation is part of development

Every project needs a README.

README template:

```
# Project Name

## What it does

## What it does not do

## Why it exists

## How to run locally

## Test / verification steps

## Safety notes

## Known limits

## Next improvements
```

If you cannot explain the project in a README, you do not understand it yet.

Part IV — Building AI features

Chapter 21 — LLM basics for developers

You do not need to know every model detail to start, but you need the mental model.

Key terms:

- **model** — the AI system generating output;
- **prompt** — instructions and context;
- **context window** — how much text the model can consider;
- **tokens** — chunks of text that affect cost and limits;
- **temperature** — randomness setting in many APIs;
- **system instruction** — higher-level behavior guidance;
- **structured output** — asking for JSON or schema-shaped data;
- **tool call** — the model asks software to do a bounded action;
- **eval** — a repeatable check of output quality.

Developer habit: separate prompt, input data, model settings, output parser, and review criteria. Do not hide everything inside one magic prompt.

Chapter 22 — First AI API call safely

Before using a real paid key, read the provider docs and understand billing, privacy, and rate limits. If you use a real key:

- store it in an environment variable;
- keep it out of Git;
- add `.env` to `.gitignore`;
- never paste it into chat;
- set usage limits where possible;
- rotate it if leaked;
- test with tiny inputs first.

Practice path:

1. Build a mock function that pretends to call AI.

2. Make your app work with the mock.
3. Add the real provider call only after the rest works.
4. Log cost-sensitive usage carefully.

Runnable mock-first adapter in Node:

```
async function callAi({ prompt }) {
  if (process.env.USE_REAL_AI !== "1") {
    return {
      text: `MOCK AI RESPONSE: ${prompt.slice(0, 80)}`,
      source: "mock",
      estimated_cost: 0,
    };
  }

  const apiKey = process.env.DWAI_DEMO_AI_KEY;
  if (!apiKey) throw new Error("Missing DWAI_DEMO_AI_KEY environment variable.");

  // Replace this URL/body with the provider docs you choose.
  const response = await fetch("https://api.example.com/v1/responses", {
    method: "POST",
    headers: {
      "content-type": "application/json",
      "authorization": `Bearer ${apiKey}`,
    },
    body: JSON.stringify({ input: prompt }),
  });

  if (!response.ok) throw new Error(`AI API failed with ${response.status}`);
  return response.json();
}

callAi({ prompt: "Explain PASS/HOLD in one paragraph." })
  .then(console.log)
  .catch((error) => {
    console.error(error.message);
    process.exit(1);
  });
```

Run safely in mock mode:

```
node ai_adapter.js
```

Only switch to real mode after the owner approves account creation, billing, provider choice, environment variable setup, and test budget.

Owner gate: creating provider accounts, adding billing, and using real keys are owner-approved actions.

Chapter 23 — Structured outputs and schemas

AI text is flexible. Apps need predictable shapes. Structured outputs ask the model for data like:

```
{
  "verdict": "HOLD",
  "reason": "This touches deployment and needs owner approval.",
  "next_action": "Review the deploy checklist before running commands."
}
```

A schema defines what fields are allowed. Your app should validate AI output before trusting it.

Project 5: build a local draft reviewer.

Input: a draft note. Output: JSON with `verdict`, `risks`, `missing_context`, and `next_action`. Requirement: if output is invalid JSON, mark HOLD and ask for repair.

Acceptance criteria:

- valid sample passes;
- invalid sample is caught;
- risky sample is HOLD;
- no external actions happen.

Chapter 24 — Retrieval and RAG basics

RAG means retrieval-augmented generation. Plain English: the app finds relevant source material first, then asks the model to answer using that material.

RAG parts:

1. documents;
2. chunking;
3. embeddings;
4. retrieval;

5. prompt with retrieved context;
6. answer with citations or source references;
7. evals to check whether answers are grounded.

Safe beginner RAG project:

- use public docs or your own non-sensitive notes;
- keep the dataset tiny;
- show which source chunks were used;
- require “I do not know” when sources are insufficient;
- do not ingest private/customer/medical/legal/financial records.

Project 6: “Ask my public notes.”

Runnable toy retrieval before embeddings:

```
NOTES = [
    {"title": "Owner Gates", "text": "Deploys, billing, secrets, and public posts require owner approval."},
    {"title": "Learning Log", "text": "Write what you tried, what failed, and what you can explain."},
    {"title": "RAG Boundary", "text": "If sources do not support the answer, say HOLD: insufficient evidence."},
]


def score(query: str, text: str) -> int:
    query_words = set(query.lower().split())
    text_words = set(text.lower().replace(",", "").replace(".", "").split())
    return len(query_words & text_words)


def retrieve(query: str, k: int = 2) -> list[dict]:
    ranked = sorted(NOTES, key=lambda note: score(query, note["text"]), reverse=True)
    return [note for note in ranked[:k] if score(query, note["text"]) > 0]


def answer(query: str) -> str:
    sources = retrieve(query)
    if not sources:
        return "HOLD: insufficient evidence in the notes."
    titles = ", ".join(note["title"] for note in sources)
    combined = " ".join(note["text"] for note in sources)
    return f"Based on {titles}: {combined}"


print(answer("Do deploys need approval?"))
```

This is not production RAG, but it teaches the real shape: documents, retrieval, cited sources, insufficient-evidence behavior, and eval questions. Later you can replace the `score` function with embeddings.

Acceptance criteria:

- at least five public/sample notes;
- retrieval returns relevant notes;
- answer cites note titles;
- if no source supports the answer, output says HOLD / insufficient evidence;
- eval set includes at least ten questions.

Chapter 25 — Tool calling and function calling

Tool calling lets a model ask software to do a defined task, such as search a file, calculate a value, or classify a record. This is powerful and risky.

Safe rules:

- tools should be narrow;
- tools should have clear inputs and outputs;
- tools should log what happened;
- tools should not get broad filesystem, account, payment, email, social, or deployment authority by default;
- external side effects require owner approval.

Beginner tool demo:

- tool: `count_words(text)`;
- tool: `classify_owner_gate(text)`;
- tool: `format_markdown_checklist(items)`.

Do not start with tools that send emails, delete files, deploy apps, move money, or change accounts.

Chapter 26 — Agents with stop rules

An agent is a loop: observe, decide, act, review, continue or stop. Most beginner failures come from giving a loop too much authority.

Safe agent design:

- narrow goal;
- explicit tools;
- maximum steps;
- cost limit;
- log every action;
- stop conditions;
- PASS/HOLD/FIX verdict;
- human approval before risky actions;
- no secrets in prompts or logs.

Project 7: build a local review agent.

It can:

- read a sample task description;
- classify risks;
- produce a checklist;
- suggest next action.

It cannot:

- send, spend, publish, delete, deploy, create accounts, change settings, or use real credentials.

Runnable local agent loop:

```python
RISK_WORDS = ["send", "publish", "delete", "deploy", "billing", "password", "api key",
        "medical", "legal"]


def classify(task: str) -> dict:
    lowered = task.lower()
    risks = [word for word in RISK_WORDS if word in lowered]
    if risks:
        return {"verdict": "HOLD", "risks": risks, "next_action": "Ask owner before action."}
    if len(task.strip()) < 20:
        return {"verdict": "HOLD", "risks": ["missing context"], "next_action": "Ask for goal,
        input, output, and red lines."}
    return {"verdict": "PASS", "risks": [], "next_action": "Create a draft/checklist only."}


def run_agent(task: str, max_steps: int = 3) -> list[dict]:
    log = []
    for step in range(1, max_steps + 1):
        result = classify(task)
        log.append({"step": step, "task": task, **result})
        break  # This beginner agent reviews once, then stops.
    return log


for event in run_agent("Draft a private launch checklist, do not deploy anything."):
    print(event)
```

The point is not autonomy. The point is a controlled loop with a maximum step count, readable logs, and HOLD behavior before risk.

Acceptance criteria:

- safe examples PASS;
- risky examples HOLD;
- unclear examples ask for clarification;
- loop stops within the step limit;
- logs are readable.

Chapter 27 — Evals: testing AI output

“Looks good” is not enough. Evals are repeatable checks.

Types of evals:

- **golden examples** — known inputs and expected outputs;

- **rubric scoring** — judge against criteria;
- **safety tests** — risky requests should HOLD;
- **regression tests** — old bugs should stay fixed;
- **RAG grounding tests** — answers must cite sources;
- **cost/latency checks** — output must be useful within budget and time.

Create an eval sheet:

```
# Eval Set

## Test ID

## Input

## Expected behavior

## Required verdict

## Why

## PASS/HOLD/FIX result
```

Your AI app is not serious until it has examples that can fail.

Part V — Shipping safely

Chapter 28 — Security and privacy for AI developers

Developer safety expands the no-secrets rule.

Checklist:

- `.env` is ignored by Git;
- no API key appears in commits, screenshots, docs, logs, or prompts;
- sample data is synthetic or public;
- dependencies are reviewed before adding;
- AI-generated code is reviewed before running;
- prompt injection risks are considered for RAG and tools;

- user uploads are treated as untrusted;
- logs do not store sensitive data by accident;
- public demos do not reveal private files, tokens, or account IDs.

If a key leaks, rotate it. Do not argue with the leak.

Chapter 29 — Costs, limits, and failure modes

AI systems cost money and fail in normal ways.

Plan for:

- token costs;
- rate limits;
- timeouts;
- provider outages;
- model changes;
- hallucinations;
- invalid JSON;
- slow retrieval;
- user misuse;
- runaway loops.

Every AI feature should have:

- max input size;
- max retries;
- timeout;
- fallback message;
- cost-aware logging;
- clear “try again later” behavior;
- owner gate for paid usage changes.

Chapter 30 — Deployment without pretending it is harmless

Deployment means making software reachable outside your machine. That can be public and risky.

Before deployment:

- app runs locally;
- tests pass;
- secrets are in environment variables, not code;
- sample data is safe;
- logs are safe;
- README explains limits;
- owner approves provider, billing, repo visibility, domain, and DNS;
- rollback plan exists.

Deployment is optional in this book. You can become a credible beginner AI developer with local demos, private repos, screenshots, walkthrough videos, and clear README files before touching public infrastructure.

Owner gate: public deploy, GitHub push, provider setup, billing, domain, DNS, and production data are not automatic learning steps.

Chapter 31 — Monitoring, rollback, and maintenance

Shipping is not the end. You need to know if the system still works.

Maintenance checklist:

- version and date of last verification;
- provider/model used;
- known limits;
- tests/evals run;
- errors seen;
- costs checked;
- dependencies updated deliberately;
- rollback path known;

- stale prompts removed;
- owner gates reviewed.

Use a weekly self-update loop:

```
Review the last 7 days using only what I provide. Ask up to 5 clarifying questions
if needed. Then produce:
1. what changed;
2. what should update in my profile/context;
3. which goals, assumptions, prompts, or routines are stale and should be pruned;
4. which AI instructions, prompts, or review gates should change;
5. one simple next-week operating rhythm;
6. any risky actions that need owner approval.

Do not make clinical, medical, legal, financial, tax, crisis, or guaranteed-outcome
claims. Keep risky actions draft-only.
```

Accept only updates you understand.

Chapter 32 — Portfolio without fake claims

A beginner portfolio should show proof, not exaggeration.

For each project, include:

- what problem it solves;
- what you built;
- what AI helped with;
- what you personally reviewed and understood;
- how to run it locally;
- tests/evals;
- screenshots or demo video if safe;
- known limits;
- what you would improve next;
- what is private or intentionally not deployed.

Do not claim client results, job-level mastery, medical/legal/financial expertise, or production reliability unless you have real evidence and permission.

Part VI — Capstone path: from learner to builder

Chapter 33 — Project ladder

Build in this order:

1. **Learning dashboard** — HTML/CSS/JS local page.
2. **Learning-log summarizer** — Python local script.
3. **Task sorter** — rules-based CLI with tests.
4. **Review queue** — local web/API app with PASS/HOLD/FIX.
5. **AI draft reviewer** — structured JSON output with validation.
6. **Public-notes RAG demo** — retrieval over non-sensitive documents.
7. **Owner-gated review agent** — limited tools, logs, stop rules, no external actions.
8. **Capstone** — one real workflow turned into a small AI app with README, tests, evals, safety notes, and optional owner-approved deployment plan.

Each project should have:

- spec;
- README;
- local run instructions;
- test or manual verification;
- owner-gate checklist;
- known limits;
- learning log entry.

Chapter 34 — Capstone spec

Choose one real but safe workflow. Examples:

- organize public research notes;
- classify content ideas;
- summarize non-sensitive meeting notes you own;
- review drafts for risky claims;
- ask questions over public documentation;

- create a weekly learning report from your own notes.

Capstone requirements:

```
# Capstone Requirements

## Problem
A clear user problem.

## User
Who this is for.

## Workflow
Input → AI/process → review → output → update.

## Features
Small and specific.

## AI feature
Prompt, structured output, retrieval, tool call, or agent loop.

## Safety
No secrets, no private data, no external side effects without owner approval.

## Tests/evals
At least 10 examples, including risky HOLD cases.

## README
How to run, verify, and understand limits.

## Demo
Screenshot, video, or local walkthrough notes.
```

Chapter 35 — 90-day path

This is a realistic beginner route. Adjust pace, but keep the order.

Days 1–7: safe AI + learning log

- run the 20-minute workflow;
- create context card;
- learn owner gates;
- start learning log;
- complete the first HTML/CSS/JS local page.

Days 8–21: developer tools

- Terminal basics;
- editor basics;
- Git basics;
- private repo;
- README habit;
- JavaScript and Python small scripts.

Days 22–45: app foundations

- JSON and HTTP;
- local API or web app;
- tests/debugging;
- AI-assisted coding workflow;
- diff review;
- documentation.

Days 46–70: AI features

- mock AI call;
- real AI API only after owner-approved key setup;
- structured outputs;
- eval examples;
- RAG over public/sample docs;
- tool calling demo;
- agent loop with stop rules.

Days 71–90: capstone and portfolio

- choose capstone;
- write spec;
- build local version;
- run tests/evals;
- write README and safety notes;
- record demo or screenshots;
- decide whether deployment is necessary or whether local proof is enough.

Chapter 36 — When to ask for help

Ask for help when:

- setup fails after two careful attempts;
- you are about to paste secrets or private data;
- a command asks for admin access and you do not understand why;
- billing, provider accounts, DNS, deployment, or repo visibility is involved;
- the AI is confidently making changes you cannot explain;
- tests fail and you are guessing;
- the workflow touches customers, money, health, legal, financial, or public claims.

DW AI Studio, mentors, friends, forums, docs, and other courses can all fit here. The free essentials are not hidden. Paid help is for speed, judgement, troubleshooting, adaptation, and accountability.

Appendices

Appendix A — Starter prompt

```
I am new to using AI as a system. Help me build one tiny routine from a real task.

My task:
[turn messy notes into a plan / draft an email / research a decision / learn a topic]

My messy input:
[paste non-sensitive notes, or replace private details with placeholders]

My goal:
[what I want back]

My red lines:
Do not send messages, post publicly, spend money, change accounts, delete anything,
make commitments, make final decisions for me, ask for secrets, or continue if my
input contains sensitive/private material that should be replaced with placeholders.
Do not make therapy, counselling, diagnosis, treatment, medical, legal, financial,
tax, crisis, regulated-advice, or guaranteed-outcome claims.

Please give me:
1. a plain-English name for this workflow;
2. the inputs you need next time;
3. a useful first draft or organized output;
4. a checklist I can use to review it;
5. anything risky that needs owner approval;
6. one sentence I should add to my context card for next time.
```

Appendix B — Systems-thinking prompt

```
Teach me this like a practical system.

Topic/workflow: [what I want to learn]
Outcome I want: [what I should be able to do]
My current level: [beginner/intermediate/what I already know]
Constraints: [time, tools, budget, platform, risks]

Please:
1. explain the mental model;
2. show the workflow step by step;
3. give 3 examples;
4. quiz me to find gaps;
5. correct my understanding;
6. give me one small practice task;
7. define how I verify whether I did it correctly;
8. suggest how to update my notes/prompts after the attempt;
9. keep risky actions draft-only and flag anything that needs owner approval.

Do not make therapy, counselling, diagnosis, treatment, medical, legal,
financial, tax, crisis, regulated-advice, or guaranteed-outcome claims.
```

Appendix C — Developer build prompt

```
I am learning to become an AI developer. Help me build this project in small,
safe steps.

Project idea:
[describe it]

Current level:
[beginner / some coding / comfortable with basics]

Constraints:
[Mac/Windows, tools, time, budget]

Safety rules:
- Do not ask for secrets, passwords, API keys, 2FA codes, private tokens, card
  details, customer data, or sensitive records.
- Do not deploy, push public, create accounts, change billing, delete files, send
  messages, or change settings without owner approval.
- Start with a plan and file list before code.
- Keep changes small.
- Include tests or a verification checklist.
- Explain every command before asking me to run it.

Please give me:
1. the smallest useful version;
2. files needed;
3. acceptance criteria;
4. owner gates;
5. step-by-step build plan;
6. how to verify PASS/HOLD/FIX.
```

Appendix D — Code review prompt

```
Review this code/diff like a cautious AI developer.

Check:
1. Does it match the spec?
2. Are there hidden account, network, file, deletion, deploy, or public-send actions?
3. Are secrets avoided?
4. Are errors handled?
5. Are tests or evals present?
6. Is the README accurate?
7. What should be PASS, HOLD, or FIX?

Do not approve deployment, public push, spending, account changes, or destructive
commands.
```

Appendix E — Weekly self-update prompt

```
Review the last 7 days using only what I provide. Ask up to 5 clarifying questions
if needed. Then produce:
1. what changed;
2. what should update in my profile/context;
3. which goals, assumptions, prompts, or routines are stale and should be pruned;
4. which AI instructions, prompts, or review gates should change;
5. one simple next-week operating rhythm;
6. any risky actions that need owner approval.

Do not make clinical, medical, legal, financial, tax, crisis, or guaranteed-outcome
claims. Keep risky actions draft-only.
```

Appendix F — Owner checklist

Before any action, ask:

- Does this touch secrets, accounts, money, provider setup, public repo visibility, deployment, DNS, external messages, customer data, deletion, or regulated claims?
- Do I understand what will happen?
- Have I inspected the diff or command?

- Have I verified facts outside the AI where needed?
- Am I comfortable owning the outcome?

If no, HOLD.

Important boundary

DWAI shares practical AI and AI-developer-path resources for thinking, drafting, organizing, researching, reviewing, learning, coding, debugging, testing, building small AI apps, and shipping owner-controlled projects. This is not therapy, counselling, diagnosis, ADHD or addiction treatment, medical advice, legal advice, financial advice, tax advice, crisis support, regulated professional advice, or a guarantee of clarity, productivity, income, saved time, business results, jobs, clients, or any personal outcome. AI outputs are drafts. The owner approves risky action.

How to use this download

Use this resource for: The complete free path: safe AI foundations, developer setup, coding basics, AI-assisted coding, first AI API app, RAG/tool-calling basics, tests, deployment checks, and portfolio capstone.

- If you are new, start with the 20-Minute AI Starter Workflow, then use the full book and workbook as your main path.
- Keep the Owner Approval Checklist nearby before acting on anything risky.
- Use PDF for reading/printing, HTML for browser reading, and Markdown/TXT for AI study sources where available.
- Treat AI outputs as drafts. Use PASS / HOLD before money, accounts, public posts, deletes, deploys, customer/private data, code changes, or regulated claims.

Optional: learn it with NotebookLM

NotebookLM is a third-party Google tool. If you use it, upload only public DWAI downloads or copied public resource URLs. Do not upload private notes, secrets, customer data, account screenshots, or completed workbook pages.

1. Create or open a NotebookLM notebook.
2. Add the public DWAI PDFs, Markdown files, or public resource page URLs as sources.
3. Ask NotebookLM: "Using only these DWAI sources, explain the path in plain English, make me a 7-day study plan, quiz me, and flag anything involving secrets, accounts, money, public posting, deletion, deployment, customer or private data, code changes, or regulated claims as HOLD."
4. If your NotebookLM account has **Video Overview**, generate one for a video-style walk-through. If Video Overview is not available, use Audio Overview, briefing docs, study guides, or source-grounded Q&A instead.
5. Check NotebookLM's answer against the source citations before acting. AI study aids are drafts, not owner approval.

Do **not** upload completed workbook pages, context cards, customer/private data, private business records, passwords, API keys, 2FA or recovery codes, medical, legal, financial, tax, crisis, or sensitive personal details unless you have intentionally replaced them with placeholders and accept the tool's data terms.

Full AI for the People Book

The complete free path: safe AI foundations, developer setup, coding basics, AI-assisted coding, first AI API app, RAG/tool-calling basics, tests, deployment checks, and portfolio capstone.

Back to AI for the People | Download PDF | Download Markdown

Previous resource | Next resource