

Codex and Claude Code: A Practical Guide for Everyone

**What They Can Do, Which One Fits Your Work,
and How to Move Beyond Chat-Based AI**

Choose by Task, Workflow, and Risk—not by Hype

Daisuke Yasuda

Before reading this book

This book does not attempt to declare Codex or Claude Code universally better. It is a practical guide to choosing an entry point based on the work, environment, review method, and risk.

The case studies are not head-to-head performance tests conducted under identical conditions. They use official product information and workflow analysis to show how each task can be approached and what a person should verify.

AI service screens, features, prices, plans, and usage limits can change. Check current official information and your own account before subscribing, paying, sending confidential information, deleting files, exposing a server, or publishing anything.

When using the examples and prompts in this book, back up the source data, confirm the scope and permissions, and manually verify important results. Do not make legal, accounting, medical, security, or other professional decisions based solely on this book.

Codex, ChatGPT, and OpenAI are trademarks or names of OpenAI. Claude, Claude Code, and Anthropic are trademarks or names of Anthropic. Other company, product, and service names belong to their respective owners. This is an independent publication and is not an official publication of OpenAI or Anthropic.

Product information reviewed: August 31, 2026.

Introduction

Choose Between Two Similar Tools by the Work —not by Declaring a Winner

“What is the difference between Codex and Claude Code?”

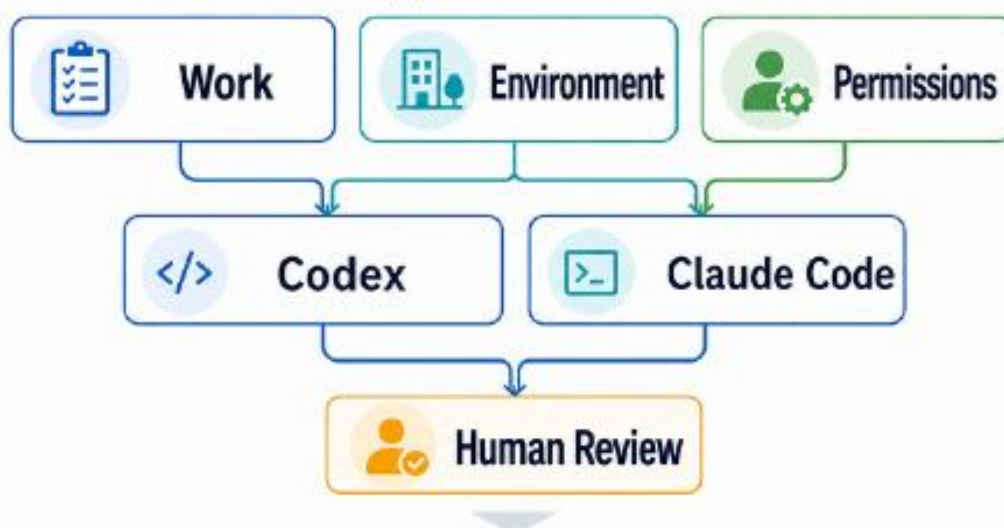
There is no honest one-sentence answer. Both are more than chatbots that return text. Within the environments and permissions you give them, they can read files, edit content, run commands, and check the results while moving a task forward. If you look only at product descriptions or a few screenshots, the two may appear almost identical.

But when you actually have to choose, the important question is not which product deserves the higher score.

The important questions are these:

1. What do you need to complete?
2. Which interface and working environment do you want to use?
3. How much authority will you give the AI?

Choose the Entry Point Based on the Work



Consider the Work Conditions Before the Product Name

Figure 1. Choose from the task, environment, and permissions, then have a person verify the result.

Look at Figure 1. It is easier to understand these tools by the way you enter and carry out a job than by placing them on opposite sides of a scorecard.

For example, “Help me write the text for our company profile” is not the same kind of task as “Inspect our entire website, fix the broken layout, and run the relevant tests.” The first may be completed through conversation. The second requires the AI to read several files, understand how they relate, make changes, run the site, and check the result.

Even when two people are fixing the same website, they may prefer different entry points. One person may want to work from a familiar desktop interface. Another may prefer a terminal or an integrated development environment, usually called an IDE. If company data is involved, convenience is not enough. You also need to know which files the AI can read, which commands it can run, and what information may be sent to an outside service.

Comparing the tools alone will not produce the right answer for you. You can choose intelligently only after considering the task, the environment, and the permissions together.

Why This Book Does Not Run a Head-to-Head Benchmark

This book does not give Codex and Claude Code the same challenge and declare a winner based on completion time or number of errors.

AI services change quickly. Available models, features, prices, and usage limits all change. Results also depend on when you run the task, which plan you have, which interface you use, the state of your computer, and the project itself. A number from one test can become misleading when it is presented as a permanent measure of a product’s ability.

Many development tasks also cannot be judged by asking which tool produced output first.

Did it leave protected files untouched?

Did it preserve existing data?

Could a person review the work before publication?

Could the change be corrected or rolled back later?

Elapsed time alone cannot answer those questions.

This book therefore separates capabilities that can be confirmed in official product information from practical methods for organizing the work. The ten cases in the second half are not the results of a competition. They are hypothetical workflow designs that show what to inspect, how to sequence the work, where to pause, and what a person should verify.

This is not a league table. It is a map for making your own decision.

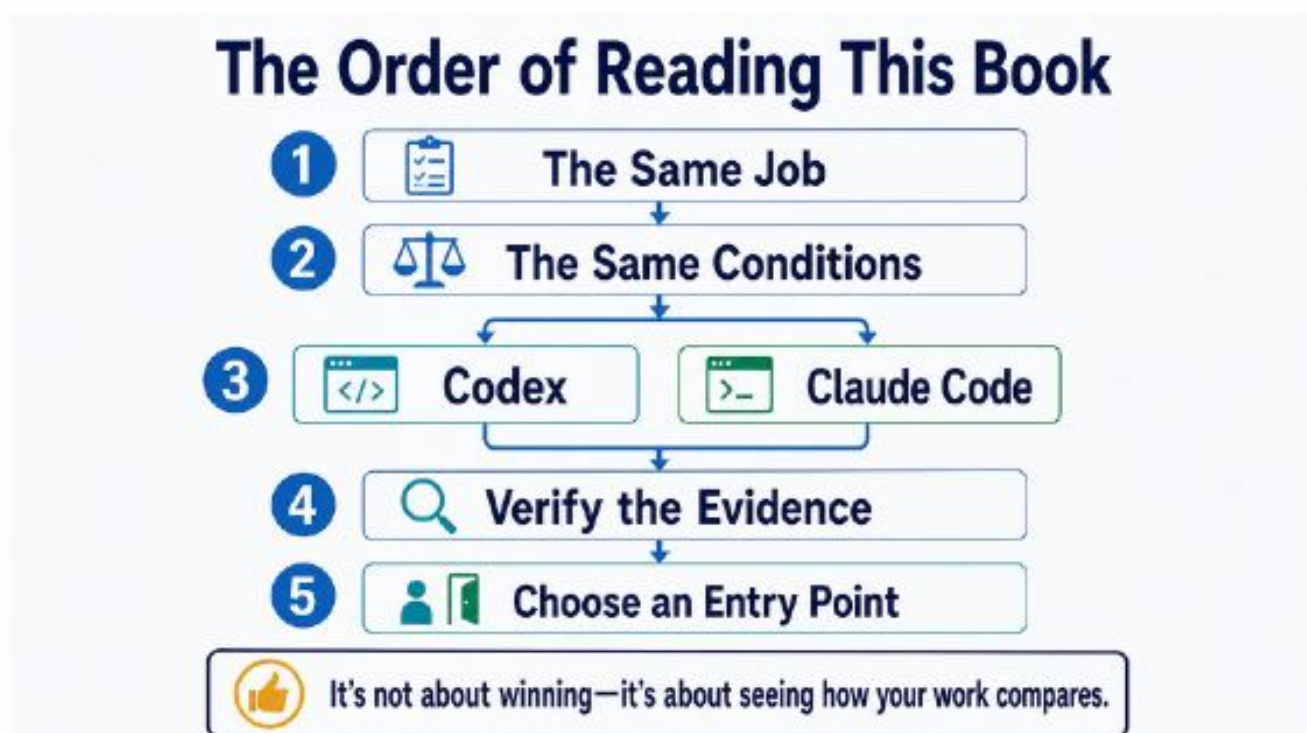


Figure 2. Begin with the same task and conditions, inspect the evidence, and then choose one tool or a deliberate combination.

Figure 2 shows the reading method used throughout the book. Each case begins with a task and a shared set of conditions. It then describes plausible routes through Codex and Claude Code using the same definition of done. Finally, you examine the deliverable and the verification record before deciding which entry point fits the job.

“Use both” does not mean giving both products permission to make the same changes at the same time. A safer combination assigns distinct roles and makes the handoff explicit. One tool might inspect an existing project, for example,

while the findings, evidence, and unresolved questions are packaged for the other tool to continue.

Why This Matters to Non-Programmers

The word *development* may bring to mind a programmer typing unfamiliar symbols into a black terminal window. AI coding agents, however, can work with more than application code.

They can help organize website copy, clean up CSV columns, turn a recurring report into a repeatable process, inspect the source data behind an internal document, find information across many files, or check a page before it is published. These tasks share a common structure: files, folders, steps, and verification.

A non-programmer does not need to write every line of code. The more important starting point is learning to distinguish between these states:

- Did the AI only explain a change, or did it actually edit a file?
- Did it create one sample, or did it build a repeatable workflow?
- Did the AI say “complete,” or did a person open and inspect the deliverable?
- Did the result work on one computer, or was it also verified in the published environment?

Understanding those four distinctions can dramatically improve both the instructions you give and the checks you perform.

What You Need to Know Before Comparing Codex and Claude Code

Both Codex and Claude Code have more than one way to use them. Depending on the product and plan, you may work through a desktop app, browser, terminal, IDE extension, or cloud environment. You cannot define either product completely from one screenshot.

The AI *model*—the system that reasons about text and code—is also not the same thing as the product and working environment. A model may be capable of proposing an excellent solution, but actual work cannot proceed unless the

product can read the relevant files and has permission to run the necessary tools. More authority can make a tool more useful, but it also increases what you must review.

This book will not ask you to memorize every screen name or feature. Those details change quickly. Instead, it returns to a set of questions that remain useful even when the interface changes:

What can the AI see right now? What can it change? Where will the work run? Who will perform the final verification?

How This Book Is Organized

Part I explains the difference between chat-based AI and AI coding agents, then introduces the overall shape of Codex and Claude Code. You will learn where their capabilities overlap and where their working environments need to be distinguished.

Part II presents seven ways to analyze a job before choosing a product. Is the request a one-time task or recurring work? How much risk does it pose to existing files? Does it involve confidential information? Where should human approval be required? The work itself is broken down before a product name is chosen.

Part III uses familiar cases: Excel, PowerPoint, a new website, changes to an existing website, a small web system, bug investigation, bulk file organization, CSV automation, server deployment, and research in a large project. The question is not which product won. Each case compares plausible ways to proceed, the information to provide at the beginning, where to stop for review, and what to check at the end.

Part IV explains how to write useful requests. The goal is not a single brilliant prompt. Instead, you learn to separate investigation, proposal, change, and verification so that dangerous assumptions are easier to catch.

Part V focuses on safety: permissions, confidential information, external transmission, deletion, publication, and cost. These issues are explained in language that does not require a programming background.

The final section includes a decision worksheet based on your work and environment. It does not force every reader toward the same conclusion. You may choose one tool for one task, divide roles between both, or clarify the work in a conversational AI before assigning it to a coding agent.

A Person Still Owns the Final Decision

An AI coding agent can become a capable worker, but it does not know every condition behind your request.

Is this an internal prototype or a final version for customers?

Is an old file safe to delete, or must it be retained for an audit?

Technology alone cannot answer questions like these.

Assigning work to AI does not eliminate human review. In fact, the more actions an AI can perform, the more important it becomes to decide in advance what it may do, where it must stop, and what evidence a person will inspect.

You do not need to remember every feature name when you finish this book. It is enough if you can say:

“For this job, I will ask for an investigation first.”

“I will keep this change in a state that can be rolled back.”

“I will inspect the actual file and screen instead of relying only on the AI’s completion message.”

“I will choose from the way the work needs to proceed, not from the product name.”

Let us begin with the difference between the chat AI you already know and an AI coding agent that can carry work forward.

Chapter 1: What Is the Difference Between Chat-Based AI and an AI Coding Agent?

From an AI That Answers to an AI That Carries Work Forward

The central difference is simple: does the AI stop after returning an answer, or can it carry out real work within the environment and permissions you provide?

If you tell a chat-based AI, “The button on my website does not work,” it may describe possible causes and show an example fix. You read the explanation, locate the relevant file, replace the code, open the page, and check the result yourself.

If you give the same problem to an AI coding agent, it may be able to inspect the project files, find the relevant area, propose a change, edit the file, run tests, and check the display—all within the tools, environment, and authority you have given it. It then reports what changed and what it was actually able to verify.

From Answers to Deliverables



Figure 2. With chat-based AI, a person applies the answer. An AI coding agent can connect investigation, editing, and testing in one workflow.

Figure 2 shows the shift from a person taking back every step of the work to assigning part of the working process to AI.

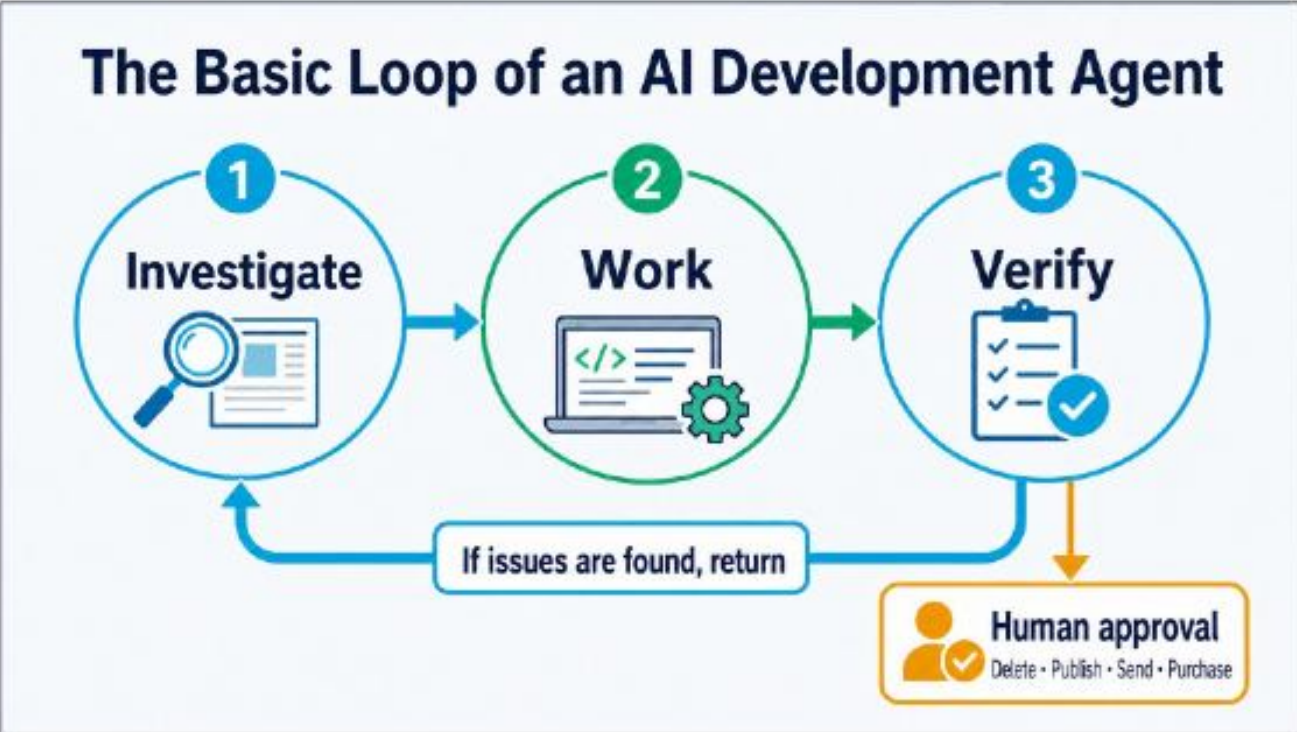


Figure 2-2. The basic cycle of an AI coding agent: investigate, act, and verify while preserving evidence.

An agent makes more sense when you think of it as a cycle of investigation, action, and verification rather than a system that stops after producing an answer.

This does not mean that an agent can automatically complete anything. Its actual capabilities depend on the tools that are connected, the folder or project that is open, the permissions it has, and the features included in the product or plan. It does not simply gain access to an entire computer on its own.

This chapter explains the difference through the flow of work rather than through technical terminology.

What Chat-Based AI Does Well

Chat-based AI is particularly useful for organizing ideas through conversation. For example, you can ask it to:

- expand an idea;
- rewrite difficult text in plain language;
- draft an email or proposal;
- show an example spreadsheet formula or program;
- explain an error message; or
- suggest a sequence of steps.

Conversation is the center of this experience. You provide text, an image, or a file, and the AI responds to what it sees. After receiving the answer, you usually decide what to use and where to apply it.

That arrangement has an important advantage. Because the AI does not immediately change a file, it is easy to use for discussion and drafting. When you want to clarify your thinking or understand your options before changing anything, stopping at conversation can be the safer choice.

The handoff becomes burdensome when the work involves ten or a hundred items. You copy code from the answer, find the correct file, paste it, save it, run it, and return to the chat when an error appears. That repeated transfer between advice and action becomes work of its own.

What an AI Coding Agent Adds

An AI coding agent adds the ability to use tools.

Common tools include reading and editing files, searching a project, running commands, executing tests, and reviewing changes. Depending on the product and environment, an agent may also work with a browser, external service, or cloud workspace.

The important point is not access to any one tool. It is the ability to connect investigation, judgment, action, and verification in support of one goal.

For a broken button, the workflow might be:

1. Inspect the project structure.
2. Find the file that displays the button.
3. Trace the action triggered by the button.

4. Narrow down the likely causes.
5. Change the necessary area.
6. Run tests or inspect the page.
7. Report the change and anything that remains unverified.

A chat-based AI can describe this process. An AI coding agent can perform some or many of these steps within the scope you have authorized.

The Basic Cycle: Investigate, Act, Verify

It is helpful to understand an AI coding agent as a three-stage cycle.

1. Investigate

The agent first establishes the goal and current state. It checks what is in the folder, which files matter, whether there are already uncommitted changes, and which tools are available.

If this step is incomplete, the agent is more likely to modify an unrelated file or overwrite work that already belongs to the user. For a complex task, it can therefore be wise to begin with, “Investigate first. Do not change anything yet.”

2. Act

The agent then follows the plan. It may revise text, edit code, add a file, or run a command.

Not every action has the same impact. Creating a new draft file is usually easy to undo. Deleting existing data, publishing to an external service, sending something to a customer, or triggering a charge can have much larger consequences. High-impact actions should have a human approval gate before execution.

3. Verify

After making a change, the agent checks the result. The existence of a new file does not prove that the goal has been met.

For software, do the tests pass? For a website, does the page actually display correctly? For a spreadsheet, are the columns and totals correct? For a

presentation, is any text clipped? For a deployment, is the new version visible at the public URL?

Each deliverable requires an appropriate check.

If the result is wrong, the cycle begins again: inspect the state, revise the work, and verify it. That loop is the foundation of agent-based work.

“The AI Said It Was Done” Is Not the Same as “The Job Is Complete”

An AI coding agent may finish with the message, “Completed.” That message alone is not enough to establish that the job is complete.

The agent may have checked a narrower condition than the one the person actually cares about.

For example, an agent may report, “I fixed the website,” while the real state is one of the following:

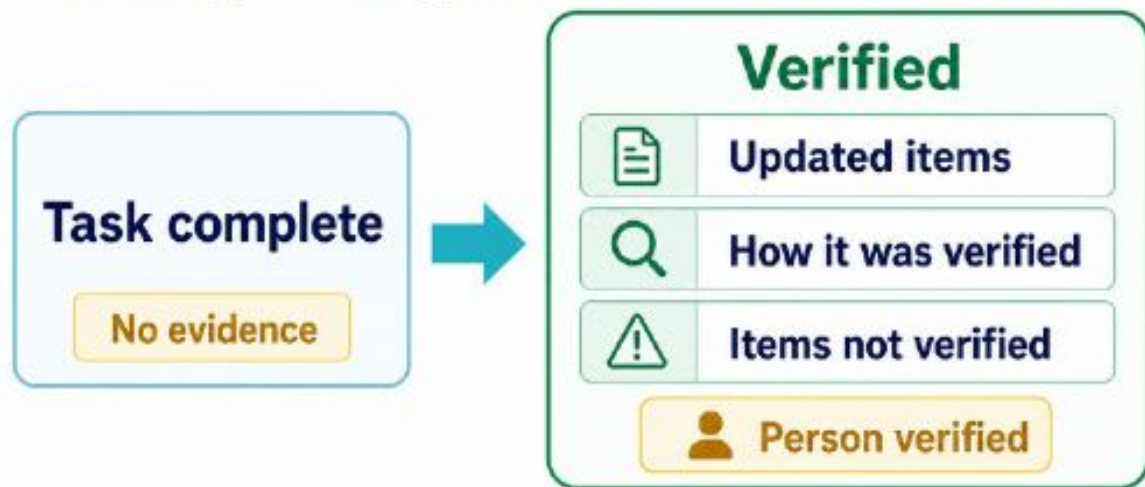
- The file was changed, but the page was not opened in a browser.
- It worked on one computer, but the public site was not updated.
- The website was checked, but the contact page was not.
- The desktop layout was fixed, but the mobile layout was not inspected.
- The new behavior works, but possible effects on older features were not tested.

The goal is not to distrust every completion report. The goal is to ask what evidence supports it.

A useful report contains at least three things:

1. What changed?
2. How was it checked?
3. What remains unverified?

Verify completion with evidence



Reports and verifications are separate

Figure 2-4. Separate the change, the method of verification, and the remaining unknowns, then compare them with the human definition of done.

The left side of Figure 2-4 contains only an unsupported “Done.” The right side identifies the changed files, the tests or browser checks, and the items that have not yet been verified. Only with that information can the requester compare the agent's work with the actual acceptance criteria.

An unresolved item is not automatically a failure. Treating an unchecked item as verified is more dangerous. If desktop display is verified and mobile display is explicitly unverified, you know exactly what to inspect next.

Even a non-programmer can improve the quality of a report with one request:

Separate what you verified from what you have not yet verified.

The AI's Scope Is Not Unlimited by Default

Some people worry that an AI coding agent will be able to see everything on their computer. Others expect it to find every necessary file automatically. Both assumptions skip the need to inspect the real setup.

What the AI can see depends on the product, settings, open workspace, connected services, and permissions. One task may be limited to a single folder. Another may add access to an external service. Read access may be allowed while writing or command execution still requires separate approval.

Before the work begins, check:

- Which folder is in scope?
- Is the agent allowed only to read, or also to edit?
- May it run commands?
- May it use the internet or connect to an outside service?
- Must it stop before deleting, publishing, sending, or purchasing?

More permission is not always better. Giving only the access required for the task limits the possible damage from a mistake.

Work Scope and Approval Boundaries

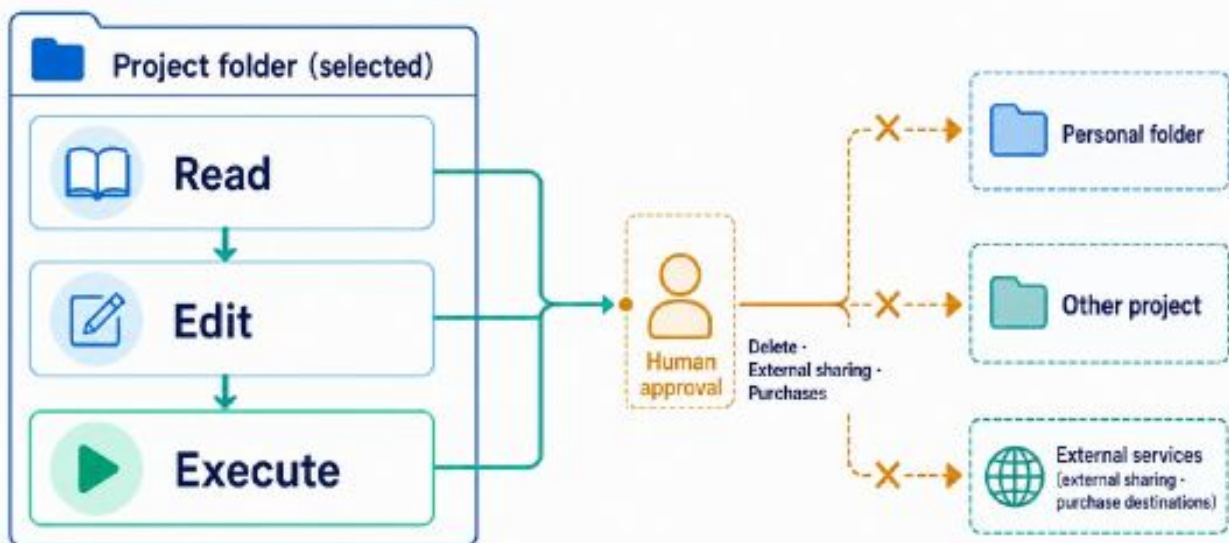


Figure 2-3. Reading, editing, and execution are separate permissions. Out-of-scope and external actions require human judgment.

In Figure 2-3, only the approved project is shown as the active workspace. Using an AI coding agent does not mean handing over the entire computer. You can limit the target folder, permit only the necessary actions, and place a human decision before high-impact operations.

Do not assume that an agent that can read a file can also change it, or that an agent that can edit locally can send data outside the system. Reading, writing, command execution, internet access, and external transmission are distinct actions even when they appear in one interface.

Decide which of them this task actually requires.

■ Separate Local, Cloud, and Remote Work

When choosing an AI coding agent, distinguish the interface you are looking at from the place where the work actually runs.

Local work uses files and software on your own computer. It is convenient for an existing local project, but the required tools and settings must be present on that computer.

Cloud work runs in an environment provided by the service. Your own computer may not need to remain occupied for the entire task, but you need to understand which data is uploaded and how the result returns to you.

Remote work means starting or monitoring work from another device or browser. A browser interface does not prove that every operation runs in the cloud. Conversely, a desktop interface may still communicate with external services.

The names matter less than three practical questions:

1. Where are the source files?
2. Where does the processing take place?
3. Where is the result saved?

■ Seven Terms You Need to Know

The following words appear throughout the rest of the book.

File

A named unit of stored information, such as a document, spreadsheet, image, or program.

Folder

A container that organizes files and other folders. When you assign work to AI, the folder placed in scope matters.

Command

A short instruction to the computer. Commands can list files, run a program, or execute tests.

Terminal

An interface for entering commands and reading their text output. It is not always a black screen, and an AI coding agent may use it behind the scenes.

Codebase

The connected collection of code, configuration, documentation, images, and other material used to build one application or website. It refers to the related whole, not a single file.

Test

A procedure for checking whether the result still behaves as expected. Some tests run automatically; others require a person to inspect a screen. Passing one test does not prove that every requirement is satisfied.

Git

A system for tracking changes to files. It helps you compare what changed and, in some cases, restore an earlier state. It does not automatically protect unsaved work or data that is outside version control.

You do not need to explain these terms perfectly. If you can ask, “Which file is that?” “What will this command change?” and “Which test verified the result?” you are already better prepared to work safely with an agent.

Begin with an Investigation That Makes No Changes

A non-programmer does not need to prescribe the exact fix at the beginning. When you do not yet understand the target, separating investigation from change is often safer.

For example:

Inspect the structure of this folder. Do not change anything yet. Explain the important files and the flow of data without specialist terminology. Also list anything you do not understand and anything I should confirm before making changes.

This request does four useful things:

- limits the target to “this folder”;
- stops modification with “do not change anything yet”;
- specifies how technical the explanation should be; and
- makes uncertainty and approval points visible.

After reading the investigation, you can move to the next stage:

“Change only this one area.”

“Show me the proposed change first.”

“Confirm that the work can be rolled back before you continue.”

Dividing a task and adding checkpoints is not evidence that you are bad at using AI. It is a core skill for working with an AI coding agent.

Use Chat-Based AI and Coding Agents at Different Stages

These tools are better understood as useful at different stages of work than as competitors in every situation.

If the goal is still unclear, begin with a conversation. Clarify who the website is for, what it needs to communicate, and which pages it requires. Then assign the implementation to an AI coding agent.

If the files already exist and the work requires repeated investigation, change, and verification, the agent's working environment may be the better starting point.

Some products offer both conversation and action in the same application. Even then, remain aware of the current state: are you discussing an idea, or is the AI operating on actual files? Similar-looking interfaces do not necessarily have the same permissions.

Chapter Summary

- Chat-based AI is primarily useful for explanations, discussion, drafting, and organizing ideas.
- An AI coding agent can connect file inspection, editing, command execution, and verification within the scope it has been given.
- The basic cycle is investigate, act, and verify.
- An AI completion message is different from completion verified by a person.
- The AI's visible scope and available actions depend on the environment and permissions.
- Begin with a no-change investigation and advance through explicit checkpoints.

Chapter 2 builds on these principles by examining the working environments Codex provides and the different ways you can enter and use them.

Chapter 2: What Is Codex?

Codex Is More Than an AI That Answers Coding Questions

In one sentence, Codex is OpenAI's agentic environment for carrying out work.

The name may suggest an AI dedicated only to writing program code. Its work, however, is not limited to code generation. Codex can help understand an existing project, build a feature or application, fix a problem, and review the resulting changes. Depending on the environment and connected tools, the work can extend to documents, spreadsheets, images, websites, and external services.

The important point is that Codex is not one screen or one model name.

As of August 31, 2026, official OpenAI documentation describes several ways to enter Codex: Codex experience in the ChatGPT desktop app, Codex CLI in a terminal, IDE extensions, and Codex cloud. Work may also be started or monitored through web and remote experiences. The interface, working location, and review method differ according to the entry point.

Create deliverables with Codex

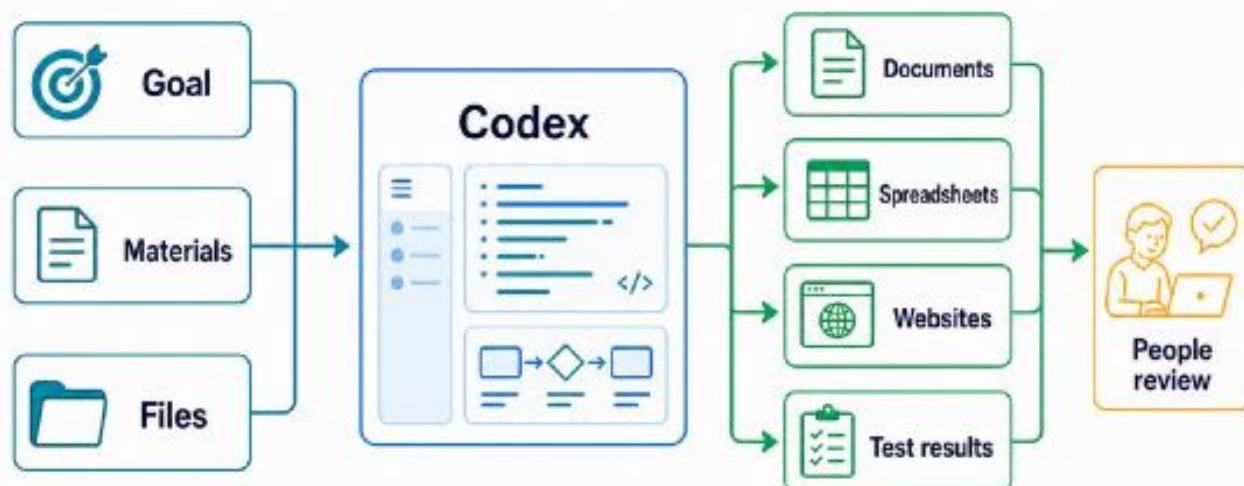


Figure 3. Codex receives a goal and source material, then connects them to a deliverable that a person can inspect.

Figure 3 shows that Codex can work through several processes and produce more than one type of output.

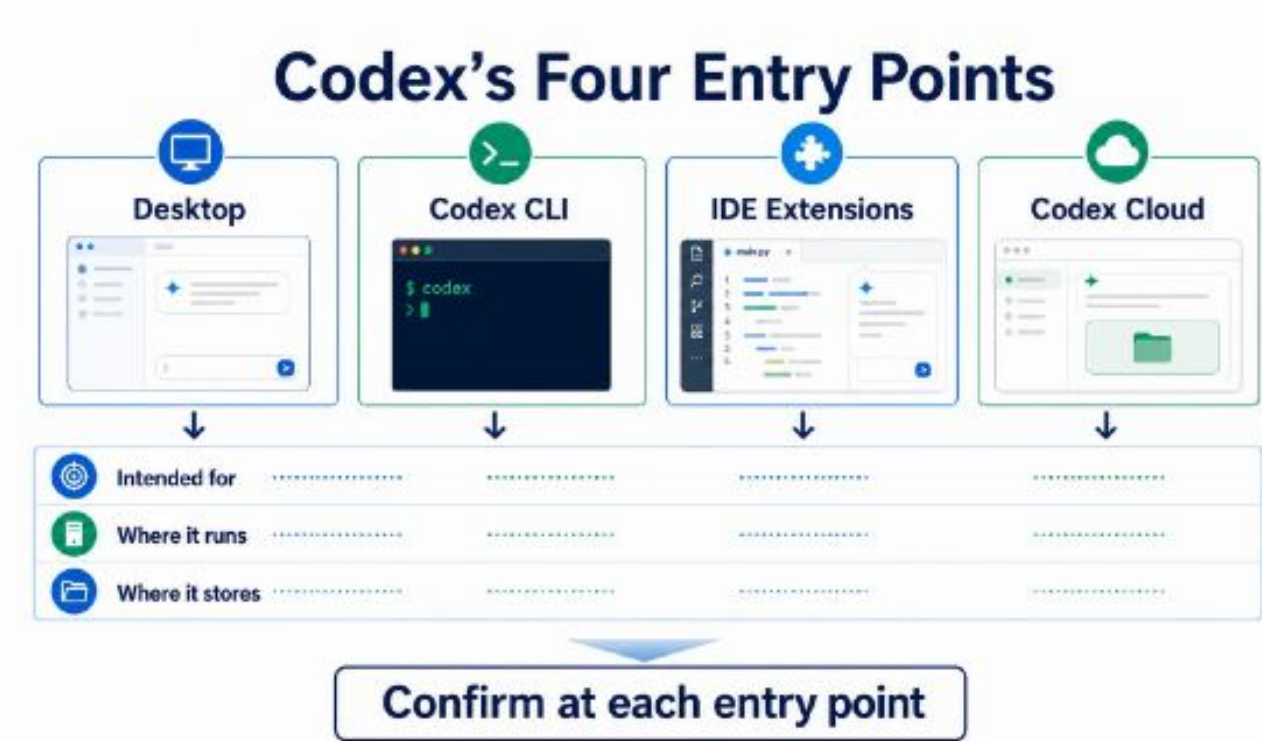


Figure 3-2. Desktop, CLI, IDE, and cloud are different entry points. For each one, check the target, execution location, and destination of the result.

The product name alone does not tell you where the work happens or which material it can use.

Separate the Product from the Model

Descriptions of AI often place the word *Codex* beside the name of a GPT model. Confusing the two makes product comparisons harder to understand.

The model is like the reasoning engine that works with language and code. Codex is the working environment in which a model can use files, tools, permissions, and execution capabilities to move a task forward.

The models available in Codex can change with time and subscription. Even when the same model name is selected, Codex in the ChatGPT desktop app and a custom system built with an API may not offer the same tools or approval process.

This book therefore does not make “Which model is strongest?” the center of its explanation. Model names change quickly. The principles for defining safe work remain useful longer.

Model and Codex work in separate environments

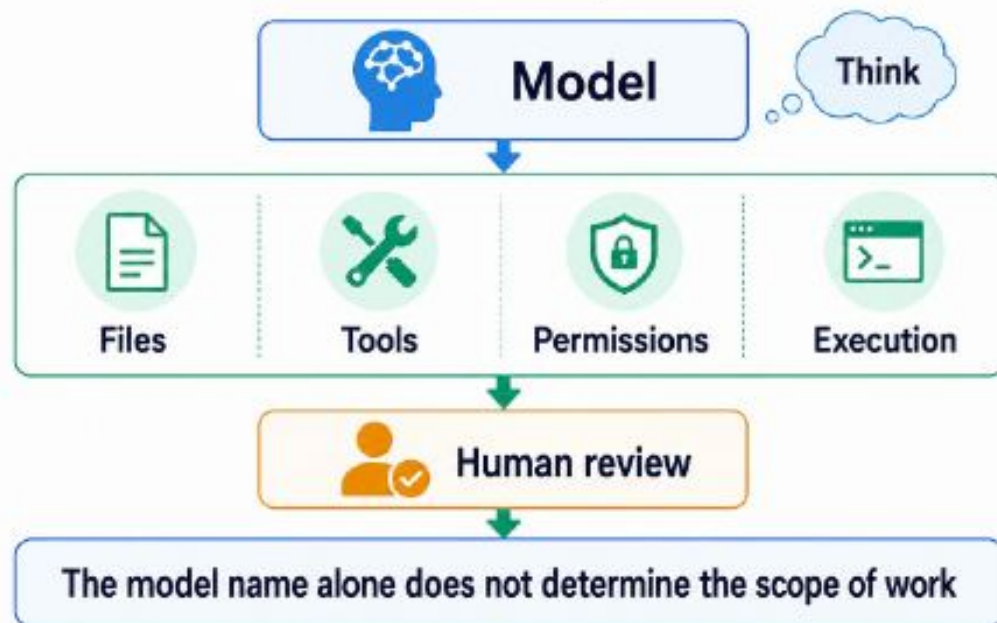


Figure 3-3. Separate the reasoning model from the environment that provides tools, permissions, and execution.

Confirm three things:

- Which working environment are you using?
- Which tools are available there?
- How much authority have you granted?

Codex Has More Than One Entry Point

The following overview explains the main entry points without assuming a programming background.

Codex in the ChatGPT Desktop App

The ChatGPT desktop app lets you begin in ChatGPT or Codex. In Codex, you can follow the conversation, files, changes, and ongoing work through a visual interface. That may make several jobs and deliverables easier to understand than a terminal-only workflow.

You can choose a project and make a request in everyday language: “Inspect this folder,” or “Revise this document.” Depending on the task, the app can show changed files or newly created outputs for review.

A clear interface does not eliminate the need for verification. Check which project is open, where the changes were saved, and what operation the agent is requesting.

Web and Remote Use

Web and remote experiences can be used to begin work or monitor its progress when you are away from the primary computer, or when the work continues in a cloud environment.

The place where you hold the conversation and the place where processing occurs are not necessarily the same. One workflow may remotely control work associated with your computer. Another may run in a cloud environment.

Codex CLI

CLI stands for *command-line interface*. It is a way to use Codex through commands and text output in a terminal.

For a developer, the CLI is close to the current folder, logs, commands, and established development workflow. It may look difficult to a non-programmer, but you do not need to memorize every command.

The important fact is that the folder from which the CLI starts affects the working context. Start it in a different folder and the visible project may change. Always confirm which folder is in scope.

IDE Extension

An IDE is an editing environment designed for software development. A Codex IDE extension places assistance and change review near the code or selection already open in the editor.

If you use Visual Studio Code or a similar environment, this reduces the need to switch between tools. If you do not develop software, you do not need to learn an entire IDE merely to use Codex. Choose the entry point that fits the work.

Codex Cloud

A cloud environment runs the work somewhere other than your own computer. It can be useful for a long task or several tasks that need to proceed separately.

Do not assume that “cloud” automatically means safe or dangerous. The practical questions are which repository or data is connected, how internet access is controlled, how the result is reviewed, and how it returns to the project.

The Workflow Codex Carries Forward

The investigate–act–verify cycle from Chapter 1 also applies to Codex.

Imagine this request: “The confirmation message does not appear after someone submits the contact form.” Within the tools and authority available, Codex may:

1. inspect the project structure and documentation;
2. find the files related to the contact form;
3. compare the current process with the expected behavior;
4. identify the likely cause and possible correction;
5. modify files within the permitted scope;
6. run tests or inspect the page in a browser; and
7. report the changes, verification results, and remaining unknowns.

That entire process is not automatic in every environment. If the browser is unavailable, visual inspection remains a human task. If the project has no tests, another verification method is needed. If the production destination is not connected, a local correction and a deployment are separate jobs.

“We gave it to Codex” does not establish how far the work progressed. Define the completion criteria in the request and read the actual verification scope in the report.

Think in Projects, Not Isolated Files

For continuing work, it is helpful to think of the target as a project.

A project is more than one file. A website contains copy, images, code, configuration, and deployment instructions. A business automation may involve input data, processing steps, output files, documentation, and tests.

The following information helps an AI understand the whole:

- the purpose of the project;
- its primary users;
- the areas that may be changed;
- protected files or data;
- the verification procedure; and
- the deliverables required at completion.

Instead of repeating those details in every request, you can keep them in project documentation or a reusable instruction. That creates a new risk: an old instruction may no longer match the current project.

A useful request is:

Read the project instructions. If any instruction conflicts with the current implementation, report the conflict before changing anything.



Figure 3-4. Follow instructions, changes, and tests, then stop for a human decision before an external action.

When the Work Uses Tools Beyond Local Files

Codex work does not always stay inside local files. Depending on the environment, it may be able to use web search, a browser, image generation, or a connected external service.

This can connect several kinds of work in one flow: research official information, organize the findings, create a document; open a website and inspect the display; or produce an image for a design proposal.

More tools also create more things to check:

- Is the web source official, and when was it updated?
- Is confidential information being sent to an outside service?
- Are the image and text usage rights appropriate?
- Is the system about to send something to a real customer or third party?
- Will the action trigger paid generation or a purchase?

“It can connect” is different from “it may use the connection without review.” Place approval before important external actions.

Permissions and Approvals Define a Boundary

Codex may request approval before editing a file or running a command, depending on the settings and operation. Its environment can also restrict work to a defined scope.

An approval screen may feel like an interruption, but it separates the boundary the AI may cross on its own from the boundary that requires human judgment.

You do not need to understand every character of a difficult command. You should still check:

1. What will the operation do?
2. Which location will it affect?
3. Can it be reversed?

4. Does it include external transmission or publication?
5. Could it cost money?

If you are unsure, ask:

Before I approve, explain why this operation is necessary and exactly what it will change.

If the explanation is still not enough to make a safe decision, do not approve. Ask for a narrower alternative.

A Good First Task for a Non-Programmer

You do not need to begin with a production website or important customer data. Start with copied documents or a small folder where a failed attempt will not cause harm.

For example:

This folder contains material from earlier meetings. Do not change the file names or contents yet. Report the number of files by type and propose a way to organize them. If you find possible personal information, do not copy it into the report; identify only the relevant file name.

This request defines the target, prohibited actions, desired result, and treatment of sensitive information. After reviewing the report, you can separately decide whether to create new folders or rename files.

The first step with Codex is not completing a large application in one attempt. Separate the work into “explanation only,” “proposal only,” “create a copy,” and “verification only,” then establish what was actually completed at each stage.

Chapter Summary

- Codex is a working environment that can inspect, change, and verify projects, not merely answer coding questions.
- Separate Codex as a product and environment from the model used inside it.
- Entry points include the ChatGPT desktop app, web and remote experiences, CLI, IDE extensions, and cloud environments.
- The location of the interface and the location of execution are not always the same.
- The more external tools and services a task uses, the more carefully you must review transmission, publication, and cost.
- Permissions and approvals are boundaries for human judgment.

Chapter 3 applies the same framework to Claude Code: what it can do, how people enter it, and where the work runs.

Chapter 3: What Is Claude Code?

Claude Code Is a Tool for Working Inside a Project

Claude Code is an agentic development tool from Anthropic. Official documentation describes it as a tool that can read a codebase, edit files, run commands, and work with development tools.

Although its name includes *Code*, its work is not limited to writing programs. Within the operations available from a terminal, it can also help create documents, search files, build projects, and conduct research. For work involving many files, such as a website or business system, its ability to investigate the whole project and coordinate related changes is especially important.

As of August 31, 2026, Anthropic's official documentation lists several ways to use Claude Code, including the terminal, IDE extensions, the desktop app, and the web. The old description of Claude Code as something used only in a black terminal window no longer captures the full product.

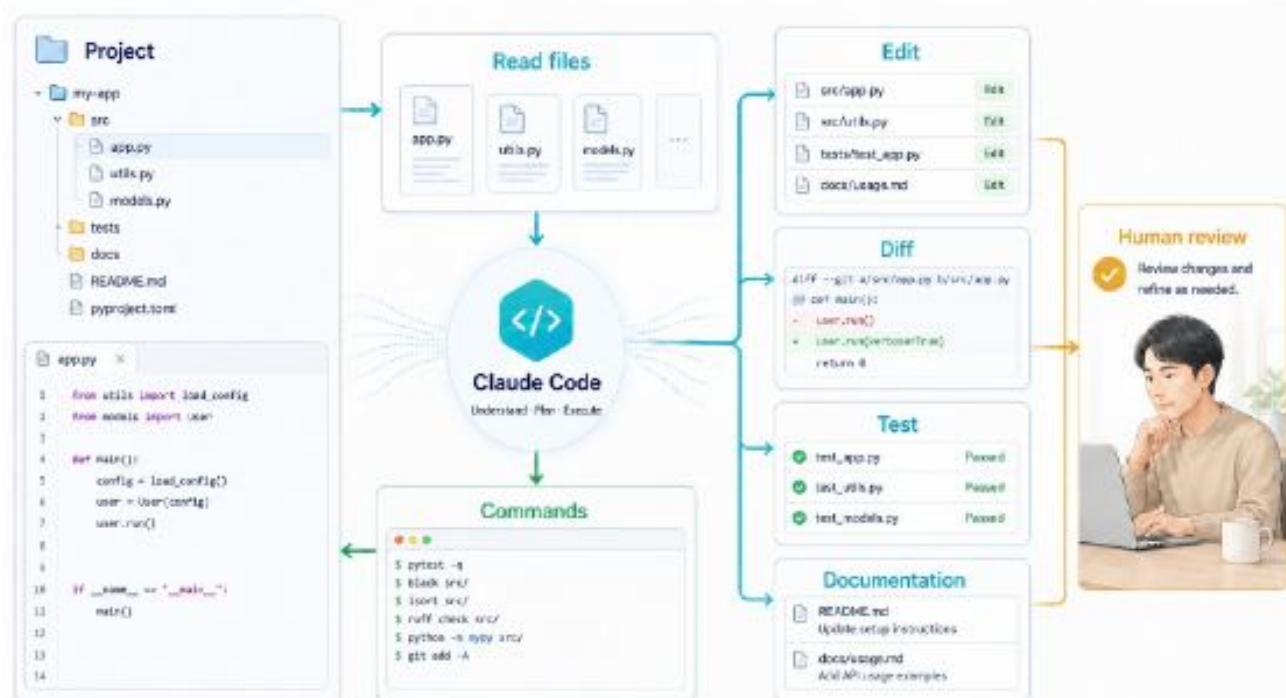


Figure 4. Claude Code brings project inspection, changes, commands, diffs, and tests into one workflow.

Figure 4 shows how Claude Code can bring code, terminal commands, Git, and tests together in a single flow.



Figure 4-2. Even when two screens look similar, confirm where the code and files are actually processed.

The terminal, IDE, desktop app, and web are interfaces through which a person gives instructions. Local execution, cloud execution, and Remote Control describe where the files and processing reside. Keeping those two questions separate is fundamental to understanding Claude Code.

Do Not Confuse Claude with Claude Code

Claude’s standard chat experience and Claude Code should be treated as distinct ways of working.

Asking Claude in a chat to write a code sample does not necessarily mean you have used Claude Code. Likewise, a Claude screen may display features related to “code” or file creation, but the label alone does not prove that the work is taking place in Claude Code.

To identify a Claude Code workflow, check whether it can:

- read the project files directly;
- edit files;