Monday Made Easier

AI HANDBOOK

Monday Made Easier with Codex

A practical participant handbook for turning context into verified outputs, then turning repeated work into reusable systems.

CORE METHOD

Understand. Act. Verify. Reuse.

Codex becomes valuable when learners move beyond one-off prompting and build repeatable workflows with files, instructions, tools, review, and verification.

FOR PARTICIPANTS

Calmer work. Smarter systems.

This handbook is designed for operators, founders, creators, developers, and teams who want practical AI leverage without losing judgement or control.

SMART TOOLS. CALMER WORK. CONFIDENT GROWTH.

Contents

Main Conclusion

Codex is best taught as a practical agentic workbench, not as a normal chatbot and not as a replacement for human judgement. It can help users write, review, and ship code; work through local, worktree, and cloud modes; use AGENTS.md, skills, plugins, MCP, browser tools, Git tools, automations, and non-code artifacts; and support workflows beyond software engineering when the user provides context, constraints, and verification criteria.

The core course thesis is:

> **Codex is valuable when you use it to turn context into verified outputs, then turn repeated work into reusable systems.**

This handbook teaches that loop.

Before you use this handbook: Codex is a fast-changing product, and feature availability can differ by plan, platform, region, installed plugins, sandbox settings, and admin controls. Before running a live session, check that learners can access the specific Codex surfaces, tools, apps, and permissions used in the exercises.

How To Use This Handbook

This handbook is for people who want to use Codex for real work: documents, research, emails, spreadsheets, apps, websites, coding, design, automations, and connected workflows.

It is not a manual for memorising every button in Codex. It is a practical guide for understanding how Codex works, how to give it the right context, how to keep control, and how to turn useful workflows into repeatable systems.

Use it in four ways:

- Read it once to understand the mental model.
- Use the prompts and worksheets while working in Codex.
- Return to the labs when you want to practise a specific workflow.
- Use the safety and verification sections before giving Codex access to tools, files, external apps, or automations.

The Big Idea

Most people use AI like a chat box. They ask a question, get an answer, and then ask another question.

Codex is more powerful than that because it can work inside an environment. It can read files, create files, run commands, use tools, connect to apps, inspect pages, generate assets, review changes, and help turn work into repeatable workflows.

That power changes how you should use it.

With a normal chatbot, the main skill is asking a better question. With Codex, the main skill is setting up a better working loop:

1. Give a clear goal.
2. Provide the right context.
3. Set constraints.
4. Define what "done" means.
5. Let Codex investigate and act.
6. Review the output.
7. Verify the result.
8. Save the workflow if it will happen again.

The people who get the most value from Codex are not the people with the longest prompts. They are the people who know how to organise context, guide the work, inspect decisions, verify results, and turn repeated work into reusable systems.

What You Will Be Able To Do

By the end of this handbook, you should be able to:

- Explain what Codex is and how it differs from a normal chatbot.
- Use Codex as a thought partner rather than treating it as an all-knowing answer machine.
- Create a useful project structure so Codex has the right working context.
- Write prompts with goal, context, constraints, and done-when criteria.
- Ask Codex to diagnose before trying again.
- Use files, plugins, skills, browser tools, image generation, and automations responsibly.
- Build useful outputs such as documents, charts, landing pages, scripts, diagrams, images, reports, and draft emails.
- Understand what should be reviewed before Codex sends, publishes, deletes, commits, or automates anything.
- Turn a repeated workflow into a reusable skill or automation.
- Collect examples so Codex can learn the standards of your work or company.

The Operating Standard

Use this standard whenever the work matters:

```
Understand first.
Diagnose before fixing.
Keep changes small.
Validate boundaries.
Verify before shipping.
Protect sensitive information.
Turn repeated work into reusable workflows.
```

This is the core discipline of the course. It applies to code, emails, documents, spreadsheets, presentations, apps, automations, and research.

Part 1: What AI Is And How To Work With It

AI Is A Prediction System, Not A Mind

AI models generate likely outputs from context. They can write, compare, transform, summarise, reason, generate code, create images, and help execute workflows. But they do not automatically understand your business, your users, your preferences, or your real-world constraints.

This matters because AI can sound confident while still being wrong.

The right mindset is not:

> **The AI knows the answer.**

The better mindset is:

> **The AI can help me think, draft, execute, and verify if I give it enough context and keep judgement in the loop.**

What AI Is Good At

AI is useful at three levels:

1. Producing useful work.
2. Running structured workflows.
3. Supporting defined responsibilities inside a team or organisation.

At the first level, AI can help with:

- Turning rough notes into structured documents.
- Explaining unfamiliar material.
- Drafting emails, scripts, reports, and briefs.
- Comparing options and showing trade-offs.
- Generating examples, variations, and first drafts.
- Finding patterns in text, data, feedback, messages, or research.
- Creating code, tests, documentation, spreadsheets, diagrams, images, and visual assets.
- Transforming one format into another.

At the workflow level, AI becomes more valuable when it has the right context, files, tools, skills, plugins, browser access, app connections, and verification criteria. Then it can help:

- Inspect a real workspace, folder, page, spreadsheet, or repo before acting.

- Follow a repeatable process instead of answering one prompt at a time.
- Use specialist skills for work such as documents, presentations, spreadsheets, browser QA, Git review, image generation, and automations.
- Draft, check, revise, and package outputs in the format the next person or system needs.
- Monitor recurring inputs, prepare routine outputs, flag exceptions, and ask for approval when the work crosses a risk boundary.
- Turn a successful process into a reusable instruction file, skill, playbook, or automation.

At the organisational level, AI can support parts of a role when the work is clearly defined. For example, it can act like a first-pass researcher, analyst, QA reviewer, content assistant, documentation assistant, sales-ops assistant, inbox triage assistant, reporting assistant, or workflow coordinator.

That does not mean AI owns the role. The human or organisation still owns the goal, judgement, permissions, approvals, relationships, ethics, and final accountability. A better way to think about it is:

> **AI can take over repeatable responsibilities inside a well-designed workflow, but people remain accountable for the outcome.**

What AI Is Weak At

AI is weaker when:

- The goal is vague.
- The context is missing.
- The task depends on private information it cannot see.
- There are hidden constraints.
- The answer depends on current facts that have not been checked.

The "Try Again" Trap

Many people use AI like this:

1. Ask for something vague.
2. Get a weak answer.
3. Say "try again."
4. Get another weak answer.
5. Repeat until frustrated.

This wastes time and money because the AI may keep solving the wrong problem.

Instead, ask diagnostic questions:

```
Show me the workflow you followed.
What assumptions did you make?
What information is missing?
Ask me any questions you need to be sure of the task.
Where exactly is the problem?
What evidence supports your conclusion?
```

The goal is to move from guessing to diagnosis.

Part 2: What Codex Is

Codex is OpenAI's agentic workbench. It started from coding work, but the useful idea is broader: Codex can work inside a real environment with files, tools, apps, browser previews, skills, plugins, Git, and automations.

A normal chat tool mostly gives you text. Codex can help create, inspect, change, and verify work. That makes it more useful, but it also means you must give clearer direction: the goal, the context, the constraints, the permission level, and how the result should be checked.

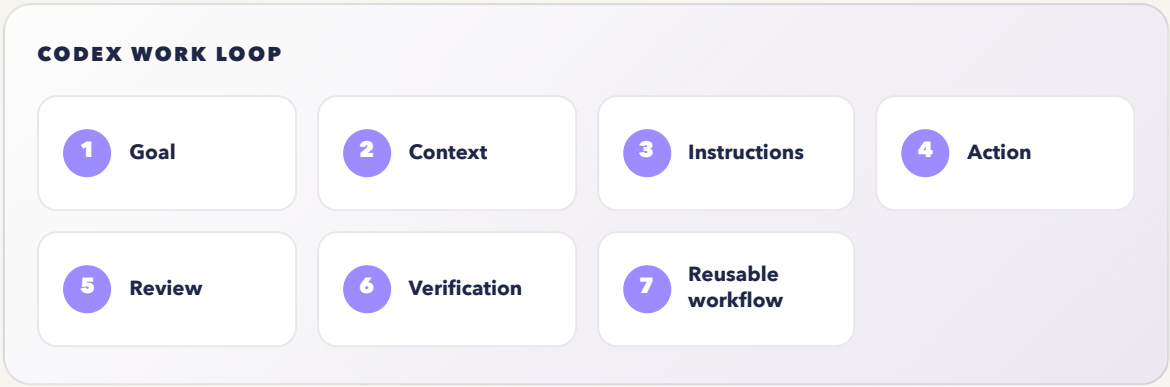
Where Codex Helps With Knowledge Work

Codex is most useful in knowledge work when the task needs more than a quick answer. It can be a decision partner, a fast way to test ideas, and a practical assistant for structured work:

- Compare options, surface trade-offs, and show the assumptions behind a recommendation.
- Try drafts, variants, prototypes, and workflow changes quickly before you commit.
- Turn rough notes, files, screenshots, or messy inputs into structured outputs.
- Inspect a folder, spreadsheet, page, repo, or draft and explain what matters.
- Create or improve documents, scripts, web pages, reports, spreadsheets, presentations, or code.
- Use connected tools to draft summaries, action lists, replies, or research notes.
- Support repeatable responsibilities with skills, plugins, app connections, and clear review points.
- Review work before it is sent, published, committed, deleted, or automated.

Do not treat Codex as magic. It is strongest when you give it context, ask it to explain its assumptions, and verify the result before relying on it.

Codex In One Diagram



Most beginners focus only on the prompt. The prompt matters, but it is only one layer. Codex becomes much more useful when you also control the context, instructions, tools, and verification loop.

The Main Codex Surfaces

Choosing the right surface matters. The same user can use more than one surface, but beginners should not try to learn all of them at once.

Codex App

Use the Codex app when you want a desktop command center for work.

Best for:

- Project folders.
- Local files.
- Multiple threads.
- Worktrees.
- Git review and commits.
- Browser previews.
- Documents, PDFs, spreadsheets, and presentations.
- Skills, plugins, image generation, and automations.

Beginner-friendly explanation:

The Codex app is the easiest place to learn Codex because you can see the project, files, chat, outputs, tools, and review flow together.

Best first use:

Create a project folder, add a small source document, ask Codex to create a local draft, then verify where it saved the output.

Codex CLI

Use the CLI when you are comfortable in the terminal.

Best for:

- Developer workflows.
- Fast repo inspection.
- Running tests and commands.
- Code review.
- Local scripting.
- Advanced workflows.

Beginner-friendly explanation:

The CLI is Codex inside your terminal. It is powerful, but it is more technical than the app.

Best first use:

Ask Codex to explain a small repository, then run one safe command or test.

Codex IDE Extension

Use the IDE extension when you want Codex inside your code editor.

Best for:

- Working on code while viewing files.
- Asking about selected code.
- Implementing focused changes.
- Reviewing code in context.

Best first use:

Select a file or function, ask Codex to explain it, then ask for a small improvement with tests.

Codex Web And Cloud

Use Codex cloud when you want to delegate work in a remote environment, usually against a GitHub repository.

Best for:

- Background tasks.
- Parallel work.
- Pull requests.
- Repo-connected work from another device.
- Tasks that do not need your local machine.

Best first use:

Give Codex a small GitHub issue or repository task, then review the resulting diff before merging anything.

Local, Worktree, And Cloud Modes

Codex can work in different modes:

- Local: works directly in your current project directory.
- Worktree: creates an isolated Git worktree so changes do not touch your main checkout.
- Cloud: runs remotely in a cloud environment.

Simple rule:

- Use Local for simple local work.
- Use Worktree for experiments or parallel code changes.
- Use Cloud for delegated repo tasks and background work.

Risk rule:

- Local can modify the files in front of you.
- Worktree isolates changes but still needs review before merging.
- Cloud can produce useful background work, but the environment, repo setup, internet access, and permissions must be configured correctly.

Part 3: The Seven Ways Codex Moves Beyond Chat

The easiest way to understand Codex is through seven practical capabilities.

1. Codex Can Work With Files

Codex can read and create files in a project. This means it can turn rough notes into a structured document, turn receipts into a spreadsheet, turn a brief into a landing page, or turn a transcript into a report.

Good first tasks:

- "Read this folder and tell me what is inside."
- "Turn these notes into a structured handbook."
- "Create a spreadsheet from this data."
- "Create a landing page from this document."

What to provide:

- The relevant folder.
- Source files.
- Desired output format.
- Any examples of a good result.

Verification:

- Open the file.
- Check headings and structure.
- Spot-check extracted facts or numbers.
- Confirm the file is saved where expected.

2. Codex Can Use Project Context And Memory

Codex can carry useful context through project files, AGENTS.md, skills, and memory-like systems.

Separate these ideas:

- Prompt context: what you tell Codex now.
- Project context: files and folders Codex can inspect.
- AGENTS.md: explicit durable instructions for the project.
- Skills: reusable workflow instructions.
- Memory: stable preferences or context that may carry across work.
- Compaction: a way of reducing long conversation context when needed.

Do not treat memory as magic. The more reliable approach is to put important instructions in files you can inspect, such as AGENTS.md or a skill.

Practical rule:

Use memory for stable personal preferences. Use AGENTS.md for project rules. Use skills for repeatable workflows. Use source files and examples for the work itself.

3. Codex Can Connect To Tools With Plugins

Plugins let Codex work with connected apps and services.

Examples:

- Gmail: search threads, summarise emails, draft replies.
- Google Drive: read and work with Docs, Sheets, Slides, and Drive files.
- Slack: summarise channels or draft responses.
- GitHub: inspect issues, PRs, reviews, and checks.
- Calendar: review events and help with scheduling.

4. Codex Can Use Skills

Skills are reusable instructions for a type of work. Think of a skill as a standard operating procedure for Codex.

A skill can tell Codex:

- When to use the workflow.
- What inputs it needs.
- What steps to follow.
- What standards to apply.
- What tools or references to use.
- How to verify the result.
- Whether approval is needed.

Use skills when:

- You repeat the same workflow often.
- Quality needs to be consistent.
- Codex keeps making the same mistake.
- A task has a specific method.
- A team needs shared standards.

Good skill examples:

- Engineering bug-fix skill.
- Code review skill.
- Course module builder skill.

- Research analyst skill.
- Presentation builder skill.
- Inbox triage skill.
- Weekly report skill.

Bad skill examples:

- A vague skill that says "make this better."
- A skill that does not say when to use it.
- A skill that tells Codex what to produce but not how to verify it.

5. Codex Can Generate Images And Visual Assets

Codex can help create or edit images as part of a broader workflow.

Use this for:

- UI placeholders.
- Product mockups.
- Presentation visuals.
- Course diagrams.
- Marketing concepts.
- Storyboards.
- Banners.
- Sprite sheets or game assets.

Good image prompt structure:

```
Create [type of image] for [purpose].
Subject:
Style:
Format:
Where it will be used:
Avoid:
```

Verification:

- Check whether the image actually shows the intended subject.
- Check whether text, logos, and details are correct.
- Check whether the image fits the final format.
- Regenerate or edit if it is unclear, generic, or misleading.

6. Codex Can Inspect Browser And Computer Workflows

Codex can use browser-related tools to inspect pages, test local apps, and verify visual results when available. Some environments also support computer use, where Codex can operate apps by seeing, clicking, and typing.

Use browser workflows for:

- Testing a landing page.
- Checking a local web app.
- Clicking through a user journey.
- Finding layout issues.
- Verifying mobile and desktop views.

Use computer workflows carefully for:

- Desktop app testing.
- GUI-only tasks.
- Settings that are not exposed through APIs.
- App flows that require clicking and typing.

Safety rule:

Computer use can affect app and system state. Keep tasks narrow and pause before sending, publishing, buying, deleting, or confirming anything.

Feature note:

Browser and computer-use availability can vary by platform, region, account, and rollout. Check current availability before teaching or relying on a live demo.

7. Codex Can Run Automations

Automations let Codex run recurring or delayed work.

Use automations for:

- Weekly inbox summary.
- Daily project brief.
- Monthly report.
- Recurring bug triage.
- PR status checks.
- Documentation updates.
- Long-running follow-up loops.

Only automate stable workflows.

Before automating, ask:

- What is the input?

- What is the output?
- How often should it run?
- What should happen if nothing changed?
- What permissions does it need?
- What can go wrong?
- How will I review the first few runs?
- Should this use a skill?

Automation rule:

A bad manual workflow becomes a bad automation. Test the workflow manually first.

Use worktrees for automations that may change files in a Git repository, especially if you do not want background work to interfere with unfinished local work.

Part 4: The Core Prompting Framework

Use this structure for most serious Codex tasks:

```
Goal:
Context:
Constraints:
Done when:
Verification:
Ask before acting if:
```

Goal

Say what you want Codex to accomplish.

Weak:

```
Make this better.
```

Stronger:

```
Turn these rough notes into a structured course module for beginner operators learning Codex.
```

Context

Tell Codex what files, examples, audience, background, or constraints matter.

Example:

```
Use the notes in this folder as source material. The audience is non-technical founders and operators. They want practical workflows, not theory.
```

Constraints

Set rules and boundaries.

Example:

```
Do not edit the original Google Doc. Create a local Markdown draft. Mark
assumptions and open questions clearly.
```

Done When

Define what completion means.

Example:

```
Done when there is a clean handbook draft with sections, exercises, prompts,
diagrams, and a final checklist.
```

Verification

Tell Codex how to check the work.

Example:

```
Verify that the file exists, scan the headings, and confirm the source docs
were not changed.
```

Ask Before Acting If

Define approval points.

Example:

```
Ask before sending emails, deleting files, installing dependencies, changing
account settings, or editing shared documents.
```

Prompt Templates

1. Ask Codex To Clarify First

```
I want to do [task].

Before giving an answer or making changes, ask me the questions you need to
understand:
- the goal
- the audience
- the constraints
- the output format
- what a good result looks like
```

2. Diagnostic Prompt

```
Before changing anything, inspect the relevant files and explain:
1. what path is actually failing
2. what evidence points to the root cause
3. what assumptions are still unverified
4. the smallest safe fix
5. how you will verify the fix
```

3. Context-First Prompt

```
Use the attached files as source context. Do not invent facts. If a section is
missing information, mark it as an open question. Produce a structured draft
with headings, examples, and next steps.
```

4. Verification-First Prompt

```
When you finish, verify the result in the same environment a user would use.
Tell me exactly what you checked and what remains unverified.
```

5. Workflow Capture Prompt

```
We just completed this task successfully. Turn the workflow into a reusable skill. Include trigger conditions, required inputs, steps, verification checks, and failure cases.
```

6. Automation Readiness Prompt

```
Assess whether this task is safe to automate. Identify the schedule, inputs, outputs, permissions, failure modes, review points, and what should happen when there is nothing to report.
```

7. Teach Me The System Prompt

Use this when you do not understand what Codex built:

```
Explain what you made in plain English.

Include:
- the files you changed or created
- the workflow
- the tech stack or tools used
- the assumptions you made
- how to run or inspect it
- how to verify it works
- what could break
```

Part 5: Projects And Organisation

Codex performs better when work is organised.

Use a project when:

- The work matters.
- There are multiple files.
- The task will continue over time.
- You want Codex to remember context.
- You need outputs saved locally.
- You are building a repeatable workflow.

First Project Checklist

1. Create a project folder.
2. Add source materials.
3. Add examples of good output if you have them.
4. Add AGENTS.md if the project has rules or standards.
5. Start a Codex thread in the project.
6. State goal, context, constraints, and done-when.
7. Ask Codex to plan first if the task is broad.
8. Let Codex create the first output.
9. Verify the output.
10. Save the workflow as a skill if it will repeat.

What To Put In A Project Folder

Useful files:

- Source notes.
- Examples.
- Brand guidelines.
- Prior reports.
- Previous scripts.
- Data files.
- Screenshots.
- Requirements.
- Checklists.
- AGENTS.md.
- Draft outputs.

Avoid:

- Unneeded files.
- Sensitive data that Codex does not need.
- Old drafts that may confuse the task.
- Multiple unrelated projects in one folder.

AGENTS.md

AGENTS.md is the project briefing Codex reads before working. It tells Codex the rules of the room.

Use AGENTS.md for:

- Project standards.
- How to run checks.
- What to avoid.
- Naming conventions.
- Review expectations.
- Safety rules.
- Output formats.
- Team preferences.

Example AGENTS.md:

```
# Project Instructions

## Working Standard
- Understand the request before changing files.
- Ask if the task is ambiguous.
- Preserve source documents unless explicitly told to edit them.
- Create local drafts before final exports.
- Verify outputs before handoff.

## Output Style
- Use clear headings.
- Keep paragraphs short.
- Include examples and checklists.
- Mark assumptions and open questions.

## Safety
- Do not send emails, publish files, delete files, or change shared documents without approval.
```

Part 6: The Permission Ladder

Not all Codex actions have the same risk.

Use this ladder to decide when to slow down.

Level	Action type	Examples	Risk	Review needed
1	Read-only	summarise docs, inspect files, search emails	Low	Usually no
2	Local draft	create local document, chart, webpage	Low to medium	Check output
3	Local edit	modify local files, update code	Medium	Review diff
4	Connected draft	draft email, PR, doc change	Medium	Review before action
5	External action	send, publish, merge, delete, buy, submit	High	Explicit approval
6	Automation	recurring unattended work	High	Controlled first run

Simple rule:

The more Codex can affect the outside world, the clearer the instruction and approval point must be.

Part 7: Workflows By Role

For Founders

Use Codex for:

- Turning rough ideas into briefs.
- Drafting landing pages.
- Creating investor or customer materials.
- Analysing feedback.
- Preparing weekly updates.
- Comparing tools.
- Creating simple internal apps.
- Building dashboards from exported data.

Good prompt:

```
Help me turn this rough business idea into a practical one-week test plan.

Include:
- target customer
- problem statement
- offer
- landing page outline
- outreach message
- success metrics
- risks
- what to build first

Ask clarifying questions before making the plan.
```

For Operators

Use Codex for:

- Inbox triage.
- Reports.
- SOPs.
- Project briefs.
- Onboarding docs.
- Spreadsheet cleanup.

- Meeting prep.
- Follow-up tracking.
- Recurring summaries.

Good prompt:

```
Turn these scattered notes into an operating procedure.

Include:
- purpose
- when to use it
- required inputs
- step-by-step process
- owner
- quality checks
- common failure cases
- escalation points
```

For Creators

Use Codex for:

- Script drafts.
- Research synthesis.
- Thumbnail or image concepts.
- Content calendars.
- Repurposing long content into posts.
- Diagrams.
- Course materials.
- Style matching from prior examples.

Good prompt:

```
Read these previous scripts and infer the style.

Then create a new script on [topic].

Constraints:
Do not copy phrases directly.
Preserve the pacing, structure, and voice.
Include visual ideas and diagram suggestions.
```

For Developers

Use Codex for:

- Understanding unfamiliar code.
- Bug diagnosis.
- Tests.
- Refactors.
- Migrations.
- Code review.
- API upgrades.
- Documentation.
- Frontend implementation.

Good prompt:

```
Find and fix the bug described below.

Before editing:
- trace the relevant execution path
- identify the root cause
- explain the minimal fix

After editing:
- run the smallest relevant test
- run broader checks if needed
- summarize the changed files and remaining risks
```

For Teams

Use Codex for:

- Shared standards.
- Example libraries.
- Review workflows.
- Reusable skills.
- Onboarding.
- Knowledge base updates.
- Recurring reports.
- Issue triage.

Good team prompt:

```
Create a team Codex operating guide.

Include:
- when to use Codex
- when not to use Codex
- approval rules
- data handling rules
- project setup rules
- AGENTS.md standard
- skill creation process
- automation review process
```

Part 8: Examples Are Assets

The best thing a company can do is collect good examples of the work it wants AI to help with.

Examples give Codex a standard to imitate, compare against, and improve.

Collect:

- Great sales emails.
- Great support replies.
- Good reports.
- Strong pitch decks.
- Approved brand copy.
- Good scripts.
- Good PRs.
- Good bug reports.
- Good dashboards.
- Good process docs.
- Good customer summaries.
- Bad examples with notes explaining what is wrong.

Store examples in:

- A project folder.
- Google Drive.
- Notion.
- A Git repository.
- A shared knowledge base.

Do not just collect finished outputs. Collect:

- The source material.
- The prompt.
- The output.
- The review notes.
- The final version.
- The lesson learned.

This becomes your practical AI dataset.

Part 9: Skills

What A Skill Is

A skill is a reusable workflow for Codex. It can include instructions, references, scripts, examples, templates, and quality checks.

Use a skill when you find yourself explaining the same process repeatedly.

Skill Creation Worksheet

Answer these before creating a skill:

- What task should this skill handle?
- Who is it for?
- When should it trigger?
- When should it not trigger?
- What inputs are required?
- What output should it create?
- What steps should Codex follow?
- What tools or plugins might it need?
- What should it never do?
- How should it verify the result?
- When should it ask for approval?

Good Skill Structure

```
---
name: example-skill
description: Use when Codex needs to [specific task]. Do not use for
[boundary].
---

# Example Skill

## Purpose

What this skill helps Codex do.

## Inputs

What Codex needs before starting.

## Workflow

1. Understand the goal.
2. Read the relevant source material.
3. Identify assumptions and missing information.
4. Create the output.
5. Verify the output.
6. Report what changed and what remains uncertain.

## Quality Checks

- Check [thing].
- Verify [thing].
- Ask before [high-risk action].

## Failure Cases

- If [condition], stop and ask.
- If [condition], produce a partial result and mark missing information.
```

When To Improve A Skill

Update a skill when:

- Codex repeats a mistake.

- A new edge case appears.
- The output format changes.
- A better example exists.
- The workflow needs a new approval point.
- The task becomes common enough to standardise.

Part 10: Automations

What Automations Are

Automations let Codex run a task later or on a recurring schedule.

Examples:

- Every Monday morning, create a project brief.
- Every Friday, summarise open action items.
- Every day, check for new support tickets.
- Every week, review recent code changes.
- Every month, create a budget variance report.

Automation Readiness Checklist

Before scheduling anything, answer:

- Is the workflow stable?
- Is the input predictable?
- Is the output clear?
- Can the output be reviewed?
- What happens if there is nothing to report?
- What permissions are needed?
- Could this accidentally send, publish, delete, or modify something?
- Should it run in a worktree?
- Should it use a skill?
- Have you tested the prompt manually?

Automation Prompt Template

```
Every [schedule], do the following:

Goal:
[describe recurring task]

Inputs:
[where to look]

Output:
[what to produce]

Scope:
[what to include and exclude]

If nothing important changed:
[what to do]

Safety:
Do not send, publish, delete, merge, or modify external systems without
explicit approval.

Verification:
Explain what sources were checked and what remains uncertain.
```

Part 11: Verification

A Codex task is not finished when something is generated. It is finished when the result is checked.

Verification By Output Type

Output	Verification method
Code	tests, lint, typecheck, build, manual smoke test
Web page	browser preview, mobile check, click path, screenshot
Document	heading scan, source comparison, render/export check
Spreadsheet	formula check, totals, sample rows, source preservation
Email draft	tone check, factual check, recipient check
Presentation	slide scan, overflow check, visual consistency
Image	subject accuracy, format, clarity, usage fit
Automation	manual dry run, first-run review, permission check
Research	source quality, citations, open questions, disconfirming evidence

Verification Prompt

```
Verify the result.

Check:
- the happy path
- important edge cases
- source accuracy
- output format
- whether anything remains unverified

Then give me:
1. what you checked
2. what passed
3. what failed
4. what you could not verify
5. recommended next step
```

Part 12: Safety And Boundaries

Do Not Trust External Input Blindly

If Codex reads emails, websites, documents, or tickets, remember that those inputs can contain mistakes or misleading instructions.

Ask Codex to separate:

- facts
- claims
- assumptions
- recommendations
- uncertain items

Do Not Log Or Share Secrets

Avoid putting secrets into prompts or files unless absolutely necessary and appropriate.

Sensitive information includes:

- passwords
- API keys
- tokens
- private customer data
- financial account details
- private medical information
- confidential contracts

Pause Before External Actions

Codex should pause before:

- sending emails
- publishing files
- merging PRs
- deleting files
- changing permissions
- buying anything
- submitting forms
- changing production systems
- running broad automations

Start Focused, Then Expand

Codex works best when the workspace matches the job. Start with the folder, files, tools, and permissions that are relevant to the task. If the work needs more access, add it deliberately.

Good setup questions:

- Which files should Codex inspect first?
- Which tools would make this workflow faster?
- What can Codex do on its own?
- What should Codex ask before doing?

Mark Uncertainty

When information is missing, do not let Codex fill the gap silently.

Use:

```
If information is missing, mark it as "Unknown" or "Open Question". Do not guess.
```

Part 13: Common Mistakes

Mistake 1: Treating Codex As Always Correct

Correction:

Ask what evidence supports the answer and what assumptions were made.

Mistake 2: Giving Vague Prompts

Correction:

Provide goal, context, constraints, done-when, and verification.

Mistake 3: Spamming "Try Again"

Correction:

Ask Codex to diagnose the problem before attempting another solution.

Mistake 4: Using One Giant Project For Everything

Correction:

Use separate project folders for separate workstreams.

Mistake 5: Not Reviewing External Actions

Correction:

Review before sending, publishing, deleting, merging, or automating.

Mistake 6: Not Saving Good Workflows

Correction:

Turn repeated successful tasks into skills, templates, or automations.

Mistake 7: Confusing Skills, Plugins, And MCP

Correction:

- Skill: reusable instructions.
- Plugin: installable bundle that can include skills, apps, and MCP servers.
- MCP: protocol for connecting models to tools and context.

Mistake 8: Assuming Every Feature Is Available

Correction:

Check current availability for browser use, computer use, image generation, plugins, plans, pricing, and regional rollout before relying on a demo.

Mistake 9: Saying Codex Replaces Every Specialist Tool

Correction:

Teach Codex as a central workbench for many workflows. The right tool still depends on the task, team, data, and output quality required.

Mistake 10: Saying Codex Is Only For Coding

Correction:

Codex starts from software work, but the same agentic work loop is valuable for knowledge work when the right tools and context are available.

Mistake 11: Saying AI Is Bad At Ideas

Correction:

AI can generate options quickly. Human context and judgement decide what is worth building.

Mistake 12: Automating Just Because It Is Convenient

Correction:

Automate workflows with clear inputs, outputs, success checks, and fallback rules. Start with a controlled first run, then let the automation run once it behaves as expected.

Mistake 13: Treating Memory Like Project Instructions

Correction:

Put durable project standards in AGENTS.md. Put repeatable workflows in skills. Use memory for stable preferences.

Part 14: Troubleshooting Codex Work

If Codex Gives A Weak Answer

Ask:

```
What context would improve this answer?
What assumptions did you make?
What examples would help?
What constraints are missing?
Ask me the questions needed to improve it.
```

If Codex Builds The Wrong Thing

Ask:

```
Before changing anything, explain what you understood the task to be.
Then compare that to my actual goal.
Identify the mismatch and propose the smallest correction.
```

If Codex Gets Stuck

Ask:

```
Summarise:
- what has been tried
- what worked
- what failed
- the current blocker
- the next three checks you would run
```

If The Output Looks Good But You Are Unsure

Ask:

```
Act as a critical reviewer.
Find the strongest reasons this output might fail.
Separate factual issues, missing context, usability issues, and verification
gaps.
```

If A Task Crosses Boundaries

Use this:

```
Identify every boundary in this workflow:
- UI to backend
- file to app
- local to cloud
- draft to published output
- read-only to write action
- manual workflow to automation

For each boundary, state the expected input, output, failure mode, and
verification check.
```

Part 15: Glossary

Agent

An AI system that can take a goal, use tools, make intermediate decisions, and work through steps toward an outcome.

Harness

The surrounding product or workflow that gives a model tools, context, UI, guardrails, and a path to produce a specific type of output.

Codex

OpenAI's agentic work environment for writing, reviewing, editing, creating, and automating work across code, files, tools, and connected services.

Project

A working area or folder where Codex can use files and context for a specific workstream.

Thread

A conversation or task session with Codex.

Local Mode

Codex works directly in your current local project directory.

Worktree

An isolated Git checkout used to keep changes separate from your main working directory.

Cloud Mode

Codex runs in a remote environment, often connected to a GitHub repository.

Plugin

An installable extension that can connect Codex to apps, skills, or MCP servers.

Skill

A reusable instruction package that tells Codex how to perform a specific workflow.

MCP

Model Context Protocol. A way to connect models to tools and context.

AGENTS.md

A project instruction file that Codex reads before working. It defines durable guidance for the project.

Automation

A scheduled or recurring Codex task.

Verification

The evidence that an output works or is accurate: tests, previews, readback, source checks, or human review.

Context Window

The amount of information the model can consider at once. When context gets too large, older information may need to be summarised or compacted.

Part 16: Job Redesign With Codex

The goal is not to automate everything. The goal is to redesign work intelligently.

Use this prompt:

```
I want to redesign my job using Codex.

Ask me questions to understand:
- my recurring tasks
- the tools I use
- the documents, spreadsheets, emails, and systems I touch
- the decisions I make often
- the work I avoid because it is repetitive
- the work that requires my judgement
- the workflows that create the most delays

Then create:
1. tasks I should keep doing myself
2. tasks Codex could assist with
3. tasks Codex could mostly automate
4. the files, plugins, examples, and skills needed for each workflow
5. the safety or approval points for each workflow
6. a first-week implementation plan

Do not assume. Ask clarifying questions first.
```

Task Sorting Table

Task	Keep human	Codex assist	Mostly automate	Why
High-judgement decision	Yes	Maybe	No	Human accountability matters
Repetitive report	Review only	Yes	Maybe	Stable input and output
Email reply	Review	Yes	Sometimes	External action risk
Data cleanup	Review	Yes	Maybe	Requires verification

Task	Keep human	Codex assist	Mostly automate	Why
Code review	Yes	Yes	Maybe	Quality aid, not full replacement
Scheduling	Review	Yes	Sometimes	Calendar constraints and people
Research brief	Review	Yes	Sometimes	Source quality matters

Part 17: Recommended Practice Plan

Step 1: Orientation

- Open Codex.
- Create a project.
- Ask Codex to explain the project folder.
- Create one local Markdown file.

Step 2: Prompting

- Run one vague prompt.
- Run one structured prompt.
- Compare the outputs.
- Save the better prompt pattern.

Step 3: Files

- Give Codex rough notes.
- Create a structured document.
- Verify the output against the source.

Step 4: Data

- Give Codex a spreadsheet or CSV.
- Ask for cleaning and summary.
- Verify totals and assumptions.

Step 5: Connected Tools

- Use one plugin in read-only mode.
- Create a summary or draft.
- Do not take external action.

Step 6: Skill

- Pick a repeated workflow.
- Create a simple skill.
- Test it once.

Step 7: Workflow Redesign

- List recurring tasks.
- Sort into keep human, Codex assist, mostly automate.
- Choose one workflow to improve next week.

Part 18: Closing Message

Codex is not valuable because it removes the need to think. Codex is valuable because it lets you move from thought to tested output faster.

The goal is not to hand over judgement. The goal is to build a better working relationship with an agent:

- You provide direction.
- Codex gathers context.
- You define constraints.
- Codex drafts and acts.
- You review.
- Codex verifies.
- You improve the workflow.
- Repeated work becomes a skill or automation.

The best users of Codex will not just ask better questions. They will build better systems for working with AI.