CODEX ROUTE / BEGINNER MODULE 03

Prompting, Context, and Structured Outputs: Codex Route

BUILD OUTCOME

Write prompts that produce predictable outputs and can be tested before you wire them into real workflows.

BEST FOR

Use this PDF when you want exact prompts and clicks for this route instead of general theory.

MODEL TIER

Light for extraction; mid for planning; heavy for rare ambiguous strategy prompts.

FIRST INSTRUCTION

```
Inspect this repo for the Prompting, Context, and Structured Outputs build.
Explain the files a beginner needs: README.md, package.json, src, public, scripts, .env.local, and any
Supabase files.
Do not edit yet. Give me the smallest safe build plan.
```

Before using the agent, make the workspace boring and clear.

01 Open the project folder. If you do not have one yet, create a new folder on Desktop and name it after the lesson.

02 Install Node.js if npm is not available. Beginners can check by opening Terminal and typing npm -v.

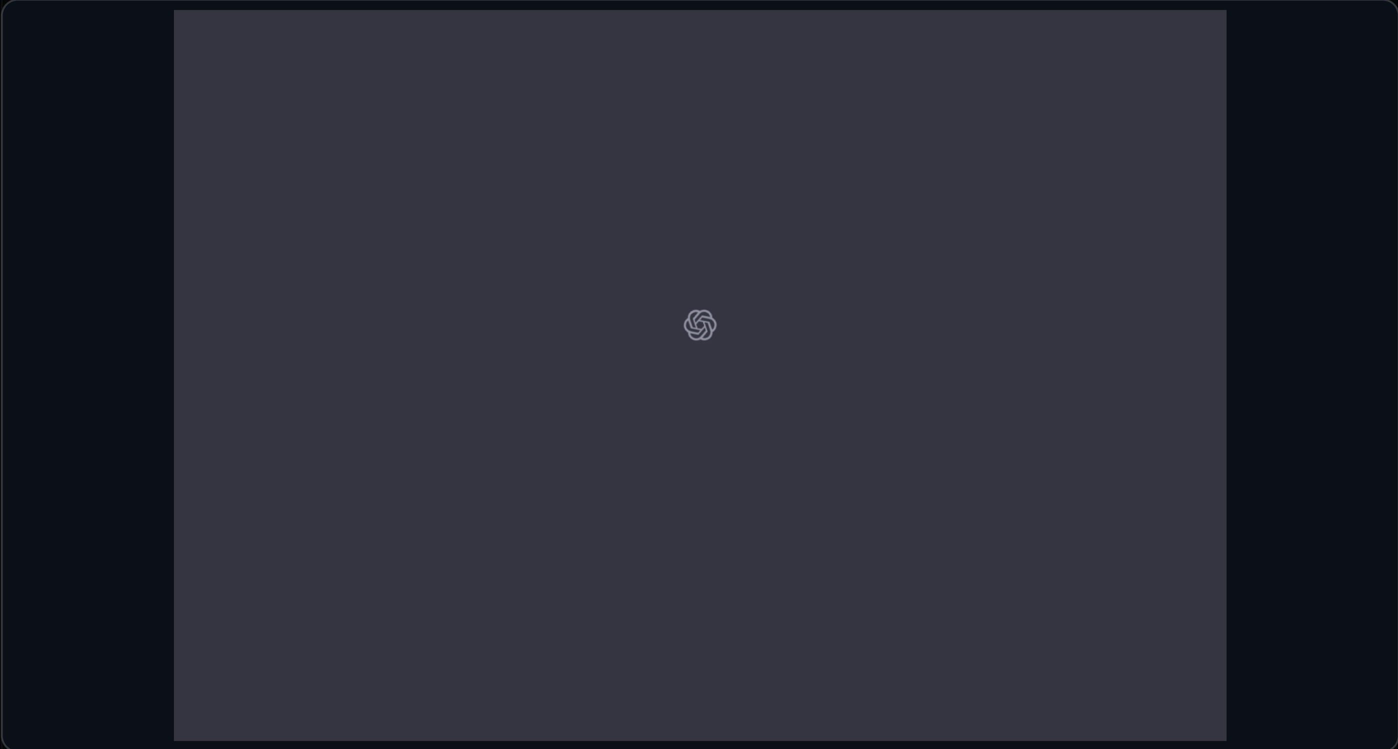
03 Create .env.local in the project root and add only the keys this lesson needs.

04 Confirm .env.local is listed in .gitignore so secrets do not go to GitHub.

05 Open the official dashboard for the first two tools in this lesson and create one tiny test project.

06 Decide where proof will live: Supabase row, screenshot, live URL, or Skool post.

OpenAI: click path for this route.



WHERE TO CLICK

Open the OpenAI platform, create an API key, add billing limits, and start with one small Responses API call.

ENV NAMES

```
OPENAI_API_KEY
```

Claude: click path for this route.

WHERE TO CLICK

Open Anthropic Console, create a key, and use Claude for planning, review, and long-context reasoning.

ENV NAMES

```
ANTHROPIC_API_KEY
```

Copy this box exactly, then let the agent ask clarifying questions only if needed.

INSPECT PROMPT

```
Inspect this repo for the Prompting, Context, and Structured Outputs build.
Explain the files a beginner needs: README.md, package.json, src, public, scripts, .env.local, and any
Supabase files.
Do not edit yet. Give me the smallest safe build plan.
```

BEGINNER CHECK

If the agent proposes a huge rebuild, ask it to shrink the scope to the smallest artifact that proves this lesson.

DO NOT PASTE

Do not paste service-role keys, payment secrets, signing keys, or private client data into the chat.

Copy this box exactly, then let the agent ask clarifying questions only if needed.

BUILD PROMPT

```
Implement the smallest useful version of Prompting, Context, and Structured Outputs.
Outcome: Write prompts that produce predictable outputs and can be tested before you wire them into real
workflows.
Tools allowed: OpenAI, Claude, Codex, Context7, TypeScript, Supabase, GitHub
Keep secrets server-side. Add code comments only where a beginner would get lost.
```

BEGINNER CHECK

If the agent proposes a huge rebuild, ask it to shrink the scope to the smallest artifact that proves this lesson.

DO NOT PASTE

Do not paste service-role keys, payment secrets, signing keys, or private client data into the chat.

TYPE THIS INTO FILES

```
# README.md lesson block
Lesson: Prompting, Context, and Structured Outputs
Outcome: Write prompts that produce predictable outputs and can be tested before you wire them into real workflows.
Tools: OpenAI, Claude, Codex, Context7, TypeScript, Supabase, GitHub

# .env.local placeholders
NEXT_PUBLIC_SUPABASE_URL=your_value_here
NEXT_PUBLIC_SUPABASE_ANON_KEY=your_value_here
OPENAI_API_KEY=server_only
ANTHROPIC_API_KEY=server_only

# Verify
npm run build
```

WHERE IT GOES

README.md gets instructions. .env.local gets real local values. Vercel gets production values. Source files get implementation.

Basic English. Click by click. Do not skip ahead.

01 Open chatgpt.com or claude.ai and sign in. Click New chat.

02 Paste this template into the chat box: You are [ROLE]. Goal: [GOAL]. Inputs: [INPUTS]. Constraints: [CONSTRAINTS]. Output JSON shape: { 'task': string, 'owner': string, 'due': string, 'next_action': string }. Return only valid JSON.

03 Replace [ROLE] with task triager. Replace [GOAL] with turn a messy note into one task. Replace [INPUTS] with raw text from a meeting note. Replace [CONSTRAINTS] with no extra prose, no markdown.

04 Find a real messy meeting note on your computer. Paste it after the template, with the prefix Input: on its own line.

05 Press Enter. Read the JSON the model returns. Confirm it has all four fields.

06 Repeat with five different messy notes. Save each output in a text file called prompt-test-1.txt through prompt-test-5.txt.

Finish the build. The last step always posts proof in Skool.

01 Open Notion or any text editor. Create a page called Prompt v1. Paste your full prompt template at the top.

02 Below the template, paste three of your input-output pairs as worked examples.

03 Now feed the model a deliberately broken input: a 1-line note with missing fields and weird spacing. See if it still returns valid JSON or breaks.

04 If it broke, add this line to your template: If a field is missing, use the string 'unknown' instead of guessing.

05 Re-run with the broken input. Confirm the JSON is valid this time. Save the new template as Prompt v2.

06 Save Prompt v2 in your repo at prompts/triager.md. Commit and push so you can version it later.

The build is not finished until verification is boring.

01 Does the prompt return valid JSON five times in a row?

02 Can a cheaper model do the task well enough?

03 Do bad inputs fail safely instead of hallucinating?

04 Run npm run build or the focused test command.

05 Download or open the produced artifact and inspect it visually.

06 Write the failure and fix in README.md or the Skool proof post.

01. Commit safe files to GitHub after reviewing the diff.
02. Open Vercel and deploy a preview first.
03. Add missing env vars under Project Settings -> Environment Variables if the build fails.
04. Run the workflow with fake or test data in preview.
05. Promote or deploy production only after the smoke test passes.
06. Save the live URL and proof artifact in Skool.

AGENT DEPLOY PROMPT

```
Verify the preview build, list any missing env vars, and give me the exact smoke test before production.
```

If something goes wrong, ask for a small repair instead of a total rewrite.

01. Letting the prompt decide the schema after the fact.
02. Mixing strategy, execution, and formatting in one giant prompt.
03. Ignoring failed outputs because they looked fluent.
04. Ask the agent to explain the failing layer: local app, env var, database, external API, prompt, deploy, or auth.
05. Ask for one fix, one test, and one rollback step.
06. Do not let the agent change unrelated files to hide the failure.

SKOOL PROOF

Write a JSON-output prompt that turns a messy note into one task, one owner, one due date, and one next action.

WHAT TO POST

Artifact, screenshot or URL, what worked, what broke, exact prompt used, and the next target.

SOURCES

Codex: https://developers.openai.com/codex

OpenAI: https://platform.openai.com/docs/quickstart

Claude: https://docs.anthropic.com/en/docs/overview

Context7: https://context7.com/docs

TypeScript: https://www.typescriptlang.org/docs/