



GETTING STARTED WITH CURSOR

A Quick Guide for Engineering Students

Version 1.0 • Fall 2026

Taner Onsay, PhD & Sebnem Onsay
Published by AI First Engineering (AIFE) LLC

A concise introduction for engineering students who are new to Cursor and AI-assisted coding.

Getting Started with Cursor

A Quick Guide for Engineering Students

Goal: be productive in Cursor in about 10–15 minutes. This is a quick start, not a manual. You do not need prior experience with Cursor, and only a little with Python.

1. What is Cursor?

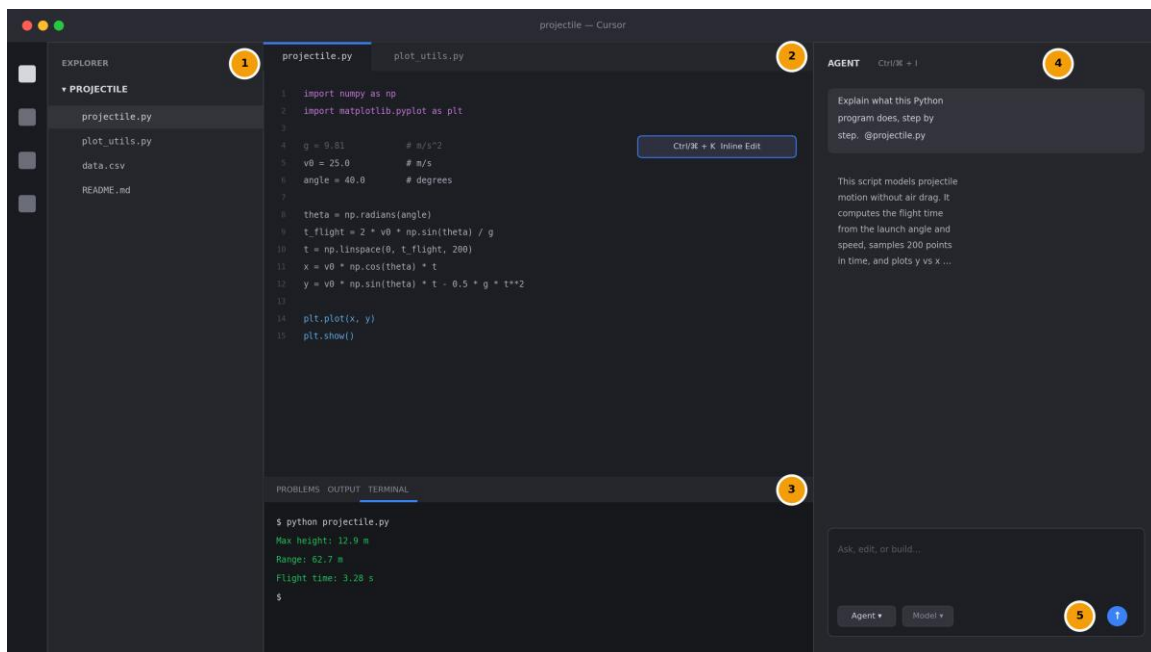
Cursor is an AI-assisted code editor. It looks and works like a modern code editor (it is built on the same foundation as VS Code), but it has an AI assistant built directly into the window. In one place you get:

- a **code editor** with syntax highlighting and autocomplete,
- **file and project management** for working with folders of code and data,
- a **terminal** for running your programs, and
- an **AI assistant** that can read your files, explain code, edit it, and even run it.

Why this matters for engineers: you can write Python (or other) code, understand code someone else wrote, find and fix bugs, modify a program to answer a new question, work across several files, and use the AI as a programming partner — all without leaving the editor.

2. The Cursor Workspace

When you open a project, the window is divided into a few areas. The illustration below is simplified; your colors and icons may differ slightly.



1 Explorer

The list of files and folders in your project. Click a file to open it.

2 Editor	Where you read and write code. Open files appear as tabs. Select code and press <code>Ctrl+K</code> (Windows/Linux) or <code>⌘K</code> (Mac) to ask for an inline edit right where the cursor is.
3 Terminal	A command line inside the editor. Run your program here, e.g. <code>python projectile.py</code> . Open it from the View menu or with <code>Ctrl+`</code> .
4 Agent panel	The AI assistant. Open it with <code>Ctrl+I</code> / <code>⌘I</code> . Type a question or a request, and add <code>@filename</code> to point the AI at a specific file.
5 Mode picker	A small dropdown in the Agent panel that switches between Ask (answers only, never edits files), Agent (can edit and run code), and Plan (writes a plan first). <code>Shift+Tab</code> cycles through them.

Opening a project: File → Open Folder... and choose the folder that contains your code. Always open the folder, not a single file, so the AI and the terminal can see the whole project.

Creating and saving files: right-click in the Explorer → New File... (give it a `.py` extension for Python). Save with `Ctrl+S` / `⌘S`. A dot on the tab means the file has unsaved changes.

3. Three Important Ways to Use Cursor

Think of the AI in three increasingly capable roles. Start with the first; move to the others as you get comfortable.

Tutor → Coding Assistant → Agent
Ask / Learn → Edit / Modify → Agentic Work

A. Ask / Learn — an AI tutor inside the editor

Use **Ask** mode (or simply ask a question) when you want to understand something. Nothing in your files changes. Try:

- “Explain what this Python code does.”
- “Why am I getting this error?” (paste the error message, or reference the terminal)
- “Explain this loop step by step.”
- “What does `numpy.linspace` do?”

B. Edit / Modify Code — a coding assistant

Ask for a specific change to existing code, in the Agent panel or with `Ctrl+K` / `⌘K` on a selection. Cursor shows the proposed edit as a **diff** (old lines in red, new lines in green). **Read the diff, then choose **Keep** or **Reject****. Try:

- “Add labels and units to this plot.”
- “Change this program so the user enters the initial velocity.”
- “Add comments explaining the important calculations.”
- “Fix this error without changing the rest of the program.”

C. Agentic Work — larger tasks across files

In **Agent** mode, Cursor can take on a bigger job on its own: inspect the project, identify the relevant files, modify several of them, run the code in the terminal, read the errors, and iterate until it works. You watch it work, answer its questions, and review every change at the end. Use it for tasks like “Split this script into a

functions file and a main file, then make sure it still runs.” Bigger jobs mean bigger reviews — check the result even more carefully.

4. A Simple Engineering Workflow

Example: you have a folder containing `projectile.py`, a short program that computes and plots the trajectory of a projectile launched at 25 m/s and 40°. Here is a typical 10-minute session:

1. **Open the project folder** in Cursor (File → Open Folder...).
2. **Open** `projectile.py` from the Explorer and skim it.
3. **Run it** in the terminal: `python projectile.py`. Look at the numbers and the plot.
4. **Ask Cursor to explain the code** (Ask mode): “Explain how this program computes the range, step by step.”
5. **Ask for one modification:** “Add air resistance as an optional drag coefficient. Keep the no-drag case as the default.” Review the diff, then Keep it.
6. **Run the modified program** again.
7. **Inspect the result.** Does the plot still look like a trajectory? Are the units right?
8. **Validate it physically.** With drag set to zero, the range should be unchanged (≈ 62.7 m). With drag on, the range must be shorter, not longer. If not, something is wrong — go back to step 4.

This is the engineering workflow you already know:

Problem → Model → Compute → Analyze → Validate → Improve

Cursor speeds up Model and Compute. It does not remove Validate. The AI cannot know whether 62.7 m is a sensible answer for your problem — you can.

5. How to Prompt Cursor Effectively

A good request gives the AI the same things you would give a lab partner: **Context + Task + Constraints + Expected result**. Compare:

Weak	Better
“Fix this.”	“This Python program calculates projectile motion. The trajectory plot works, but I want the x- and y-axes labeled with units in meters. Do not change the underlying equations.”
“Make it faster.”	“This loop in <code>simulate()</code> runs 10 000 time steps and takes about 30 s. Speed it up using NumPy arrays instead of a Python loop. The output array must stay identical.”
“It doesn’t work.”	“When I run <code>beam.py</code> I get <code>ValueError: math domain error</code> on line 42. Explain the cause first; then suggest the smallest fix.”

Tips: reference the file with `@filename` so the AI reads the right code; ask for one change at a time; and say what should not change.

6. Review Before Accepting

Cursor is fast and often right — but not always. It can invent a function that does not exist, drop a unit conversion, change an equation you did not ask it to touch, or write code that runs but computes the wrong thing. Before you trust a change:

- Read the diff. Understand every important modification, especially in equations and constants.
- Run the program and check the outputs, not just that it runs without errors.
- Verify units and equations against your notes or textbook.
- Test reasonable and unreasonable inputs (zero, negative, very large). Does the program respond sensibly?
- Compare results with engineering expectations — a hand estimate, a limiting case, or a known answer.

**AI CAN GENERATE CODE.
THE ENGINEER IS RESPONSIBLE FOR VALIDATING THE RESULT.**

7. Five Things to Remember

1. **Open the correct project folder** — the whole folder, not just one file.
2. **Give Cursor context** before asking for changes: what the program does, what you want, what must not change.
3. **Ask for small, clear changes** when possible. One change, then run, then the next.
4. **Review and test AI-generated code** — read the diff, run it, check the outputs.
5. **Validate the result using engineering reasoning** — units, limiting cases, physical sense.

Continue Learning

For current features and deeper training, use Cursor's official resources:

- **Cursor Documentation** — full reference: <https://cursor.com/docs>
- **Cursor Quickstart** — from install to your first useful change: <https://cursor.com/docs/get-started/quickstart>
- **Cursor Agent Guide** — Ask, Agent, Plan, and Debug modes: <https://cursor.com/help/ai-features/agent>

Cursor changes frequently; menu names and shortcuts may differ slightly in your version. When in doubt, ask Cursor itself: "How do I open the terminal in this version?"