# Setting Up Your Development Environment with Cursor AI (Windows 11)

In this guide, we’ll outline a complete toolchain setup on **Windows 11** for developing **LangOne** – including deciding the tech stack, installing all necessary tools, and leveraging Cursor AI for coding assistance. The goal is to help you start coding **LangOne** quickly and efficiently, with an AI-augmented workflow.

## 1. Technology Stack Decision (Rust + LLVM for LangOne)

After evaluating LangOne’s ambitious feature list (native code generation with JIT/AOT, concurrency, safety, etc.), using **Rust** for the implementation and **LLVM** for the backend is a robust choice. Rust offers memory safety and performance, which is ideal for writing a compiler or runtime, while LLVM provides battle-tested optimization and code generation across platforms. LangOne is envisioned to *“compile down to native code with JIT and AOT options”*[[1]](file://file-TxnvydkFLeAZ9yxBdh7CYo#:~:text=,Flutter%20for%20instant%20feedback%20loops), and LLVM excels at exactly that. In fact, Rust’s own compiler targets LLVM, benefiting from its optimizations and multi-target support.

**Why Rust + LLVM?** As a system language, Rust gives you low-level control with high-level safety, which is crucial for implementing complex features like LangOne’s concurrency and memory management. Using LLVM as the backend means LangOne can generate highly optimized machine code for multiple architectures out of the box. To interface Rust with LLVM, you can use the Rust **Inkwell** crate, which provides safe Rust bindings to LLVM’s C++ API. Experts recommend Inkwell because using LLVM’s C API directly (via llvm-sys) would require lots of unsafe code, whereas *“Inkwell covers all the LLVM API needed for writing a language backend”* and avoids most unsafe pitfalls[[2]](https://users.rust-lang.org/t/is-inkwell-complete-enough/79511#:~:text=I%20would%20recommend%20Inkwell%2C%20it%27s,because%20the%20crate%20is%20unstable). In short, Rust + LLVM (with Inkwell) aligns perfectly with LangOne’s needs for performance, safety, and cross-platform native code generation.

**Alternative Consideration:** If LLVM’s heavy optimizations are not a strict requirement initially, you could consider [Cranelift](https://github.com/bytecodealliance/wasmtime/tree/main/cranelift) (a lightweight code generator in Rust) as a backend[[3]](https://www.reddit.com/r/rust/comments/ej0yn4/whats_the_best_way_to_generate_llvmir_in_rust/#:~:text=If%20you%20don%27t%20have%20to,should%20also%20consider%20using%20cranelift). Cranelift is easier to integrate (pure Rust) but produces less optimized code than LLVM. Given LangOne’s “performance at scale” goals, LLVM remains the **“best of the best”** choice for the backend.

## 2. Development Tools Chain Setup on Windows 11

With the tech stack decided, let’s set up the development environment step by step. We will install the Cursor AI editor, the Rust toolchain, LLVM, and other supporting tools. We’ll also prepare the environment for future needs like containerization and ensure the editor is configured for Rust development.

### **Step 1: Install Cursor – The AI Code Editor**

Cursor is a VS Code–forked editor with built-in AI pair programming capabilities. Download the latest **Cursor** for Windows from the official site (Windows 10/11 64-bit is supported)[[4]](https://cursor.com/downloads#:~:text=Windows). Choose the installer type – “User” (installs for your account only) or “System” (requires admin, but makes available to all users). Run the installer and launch Cursor.

Once Cursor is installed, you can sign in (if you have an account) or start using it in free mode. Cursor’s interface will feel familiar if you’ve used VS Code. It even lets you import your favorite VS Code extensions, themes, and keybindings easily (Cursor can import all your extensions in one click)[[5]](https://cursor.com/en#:~:text=,Cursor%20is%20SOC%202%20certified).

**Configure Rust support in Cursor:** Since Cursor supports VS Code extensions, you should install the **rust-analyzer** extension for excellent Rust language support (syntax highlighting, code completion, inline error checking, etc.). In Cursor, open the Extensions panel and search for “rust-analyzer”, then install it. This is the same process as in VS Code – rust-analyzer is the de-facto language server for Rust[[6]](https://learn.microsoft.com/en-us/windows/dev-environment/rust/setup#install-visual-studio-code#:~:text=1,Visual%20Studio%20Code%20for%20Windows). You may also install **CodeLLDB** for debugging Rust code if needed[[7]](https://learn.microsoft.com/en-us/windows/dev-environment/rust/setup#install-visual-studio-code#:~:text=2,Ctrl%2BShift%2BX). With these, Cursor will provide a full IDE experience for Rust development.

### **Step 2: Install the Rust Toolchain**

Next, install Rust on Windows 11. The easiest method is via **Rustup**, the Rust toolchain installer.

1. **Download Rustup:** Go to the official Rust site’s install page and download the **rustup-init.exe (64-bit)** installer[[8]](https://www.rust-lang.org/tools/install#:~:text=It%20looks%20like%20you%E2%80%99re%20running,Other%20Installation%20Methods). (Alternatively, you can run a PowerShell command to download and run it in one go, as given on the Rust website.)
2. **Run the Installer:** Execute the rustup-init.exe and follow the prompts. It will install the latest stable Rust compiler (rustc), the Cargo build tool, and Rustup for managing toolchains. During installation, if you are prompted to install the **Visual Studio C++ Build Tools**, accept it. Rust (when using the MSVC toolchain on Windows) requires the MSVC linker and libraries. The installer can download the necessary Visual Studio Build Tools for you if they’re missing[[8]](https://www.rust-lang.org/tools/install#:~:text=It%20looks%20like%20you%E2%80%99re%20running,Other%20Installation%20Methods)[[9]](https://www.rust-lang.org/tools/install#:~:text=Windows%20considerations).
3. *Note:* The Rust team recommends using the MSVC toolchain on Windows for better compatibility. By default, Rustup will set up Rust with MSVC. If not, you can manually set it by running rustup default stable-msvc after installation. Ensuring the MSVC toolchain is active will allow Rust code to link against MSVC libraries (needed for LLVM later)[[10]](https://learn.microsoft.com/en-us/windows/dev-environment/rust/setup#install-visual-studio-code#:~:text=Rust%20works%20very%20well%20on,apps%20for%20Windows%20using%20Rust).
4. **Verify Rust Installation:** After installation, open a new Command Prompt or PowerShell (the installer adds Rust to your PATH) and run cargo --version and rustc --version. You should see version outputs, confirming Rust is installed. For example, Cargo is Rust’s package manager and build tool (it manages project creation, dependencies, and compilation)[[11]](https://learn.microsoft.com/en-us/windows/dev-environment/rust/setup#install-visual-studio-code#:~:text=If%20you%27re%20curious%20about%20the,and%20their%20dependencies). We will use Cargo to build LangOne’s compiler.
5. **(Optional) Visual Studio Community:** If you didn’t let rustup install the C++ build tools, you can manually install **Visual Studio Community 2022** with the “Desktop development with C++” workload. This also satisfies Rust’s MSVC prerequisites[[12]](https://learn.microsoft.com/en-us/windows/dev-environment/rust/setup#install-visual-studio-code#:~:text=Install%20Visual%20Studio%20,the%20Microsoft%20C%2B%2B%20Build%20Tools). During VS installation, you can select additional components like “Git for Windows” (Rust projects default to using Git for version control[[13]](https://learn.microsoft.com/en-us/windows/dev-environment/rust/setup#install-visual-studio-code#:~:text=While%20installing%20Visual%20Studio%2C%20there,simpler%20to%20select%20all%20three)) so that Git is available in your environment.

### **Step 3: Install LLVM (Libraries and Tools)**

Since we plan to integrate LLVM for code generation, having the LLVM tools and libraries on your system is useful. You have a few options on Windows:

* **Install via Package Manager (winget):** Windows 11 includes the **Winget** package manager. You can open PowerShell and run:
* winget install LLVM
* This command will download and install the LLVM toolkit (includes the LLVM libraries, llvm-config, clang, etc.)[[14]](https://stackoverflow.com/questions/76862060/how-to-install-llvm-on-windows#:~:text=Native%20Windows%20method%20without%20any,rd%7D%20party%20tools). Using Winget is convenient as it fetches a pre-built LLVM package (as of early 2025, Winget’s LLVM package was up-to-date with LLVM 19.x).
* **Install via Official Binaries:** Alternatively, you can download the pre-built LLVM Windows binaries from the [LLVM releases page](https://github.com/llvm/llvm-project/releases/) and run the installer. Ensure you add LLVM’s bin directory to your PATH if the installer doesn’t do so.

Having LLVM installed will be helpful for two reasons: (1) The **Inkwell** crate in Rust may need to know the path/version of your LLVM to link against it. (You can set an environment variable or Cargo feature to match the LLVM version; e.g., use feature llvm16-0 if you have LLVM 16). (2) You can use LLVM’s tools like llvm-config or llc for debugging the generated IR or assembly if needed.

**Note:** If you use Inkwell, ensure you enable the feature flag corresponding to your installed LLVM version in Cargo.toml. For example, if you installed LLVM 16, you’d enable inkwell = { version = "...", features = ["llvm16-0"] }. This tells Inkwell which LLVM to target[[15]](https://createlang.rs/01_calculator/basic_llvm.html#:~:text=The%20code%20is%20available%20in,0%22%5D%60%20in%20inkwell). The Inkwell documentation and repo will have details on supported versions.

### **Step 4: Install Docker Desktop (for Future Containerization)**

Though containerization is planned for later, it’s wise to prepare the setup now. **Docker Desktop for Windows** allows you to run Docker containers on Windows 11, leveraging WSL2 or Hyper-V under the hood. To install Docker Desktop:

* Download the Docker Desktop installer from the official [Docker website](https://www.docker.com/products/docker-desktop) (there’s a direct link for Windows x86\_64) and run it.
* During installation, Docker will prompt to enable the **WSL 2** backend and/or **Hyper-V**. For best results on Windows 11, enable WSL 2 integration. (Ensure WSL 2 is enabled on your system: on Windows 11, you may need to go to **Turn Windows Features on/off** and enable “Windows Subsystem for Linux” and restart, or use wsl --install in PowerShell.) Docker’s documentation notes that you should *“turn on the WSL 2 feature on Windows”* before using Docker[[16]](https://docs.docker.com/desktop/setup/install/windows-install/#:~:text=,Windows%2010%20or%20Windows%2011).
* After installation, launch Docker Desktop and verify it’s running (you should see the whale icon in your system tray). You can also test with docker run hello-world in PowerShell to ensure it works.

Having Docker set up will allow you to containerize the LangOne development or runtime environment later. For instance, you could create a Docker image that contains the LangOne compiler and its dependencies, making it easier to test on Linux or deploy. We’re not using Docker immediately for development (since we have a local setup), but this prepares you to use it when the time comes.

### **Step 5: Additional Tools and Setup**

* **Version Control (Git):** If you haven’t installed Git yet, install **Git for Windows** (from [git-scm.com](https://git-scm.com/downloads)). Git is essential for source control. As noted, Rust projects are git-initialized by default[[17]](https://learn.microsoft.com/en-us/windows/dev-environment/rust/setup#install-visual-studio-code#:~:text=required%20that%20we%20feel%20it%27s,simpler%20to%20select%20all%20three), so you’ll want Git to manage your LangOne source code. After installation, you can use Git Bash or integrate Git with Cursor’s terminal to commit and push code to a platform like GitHub.
* **Build System:** For Rust projects, **Cargo** is the build system and package manager. We’ve already got Cargo via rustup. You don’t need a separate build tool like CMake for purely Rust code. Cargo will handle building your compiler and managing any Rust dependencies (e.g., adding inkwell or parser libraries to Cargo.toml). If in the future LangOne integrates some C/C++ code or needs to build libraries, you can revisit build tools then, but initially Cargo is sufficient.
* **Testing and CI:** Rust includes a test framework built-in (you can write tests in each module and run cargo test). Plan to write unit tests for your language features as you implement them. For continuous integration, you can set up GitHub Actions (as mentioned in the manifesto) once your repo is on GitHub, but this is outside the local dev setup scope.

With the above steps, your development environment should be ready:

* **Cursor AI Editor** – for writing code with AI assistance.
* **Rust & Cargo** – for implementing LangOne’s compiler and tools.
* **LLVM (via Inkwell)** – for LangOne’s backend code generation.
* **Git** – for version control.
* **Docker** – (optional now, for later containerization/deployment).

## 3. Using Cursor AI to Kickstart the LangOne Project

One of the biggest advantages of using Cursor is its AI pair-programming capabilities. You mentioned you’ll use Cursor AI “mostly for coding,” so let’s leverage it to accelerate development of LangOne.

**Cursor AI Features Recap:** Cursor provides AI completion and an “agent mode” to handle multi-step coding tasks. For example, it has a Composer (Agent) feature that can execute high-level instructions to generate or modify code across multiple files. *“Cursor’s agent mode completes tasks end to end... Try it out by hitting Ctrl + I and instructing a change.”*[[18]](https://cursor.com/en/features#:~:text=Agent) This means you can press Ctrl + I (Cmd + I on Mac) to open the **Composer**, then type an instruction for a coding task, and the AI will attempt to carry it out. It can even create new files and edit multiple files in one go when using Composer mode[[19]](https://forum.cursor.com/t/agent-create-new-files/408#:~:text=%40Apologin%20%20%2014%20%2C).

To jump-start LangOne’s development, you can use the AI to generate a skeleton of your project. For instance, you might ask Cursor to create the initial project structure (using Rust) for the LangOne compiler. Here’s a **prompt** you can use in Cursor’s Composer:

I am building a new programming language called \*\*LangOne\*\*. Please help me set up a new Rust project (Cargo package) named "langone" that will serve as the compiler/interpreter for this language.  
  
\*\*Requirements and Structure:\*\*  
- Use Rust as the implementation language, and include the necessary configuration to use LLVM via the `inkwell` crate for code generation.  
- Create a module/file structure with placeholders for the main compiler components:  
 1. \*\*main.rs\*\* – entry point that will parse input and invoke the compiler pipeline.  
 2. \*\*lexer.rs\*\* – for lexical analysis (tokenizer).  
 3. \*\*parser.rs\*\* – for parsing tokens into an AST.  
 4. \*\*ast.rs\*\* – defining the Abstract Syntax Tree data structures.  
 5. \*\*codegen.rs\*\* – for code generation using LLVM (via inkwell).  
 6. \*\*errors.rs\*\* (optional) – for error handling and diagnostics.  
- In each of these files, include basic struct or function definitions as needed. For example:  
 - In `lexer.rs`: define a `Token` enum and a stub function `lex()` that returns a list of tokens (with a TODO comment).  
 - In `parser.rs`: define a `Parser` struct or function and a stub `parse()` that produces an AST (with TODO).  
 - In `ast.rs`: define simple AST node structs/enums (e.g., for expressions, statements) with derived traits.  
 - In `codegen.rs`: use the `inkwell` crate to initialize an LLVM context and module. Provide a placeholder `codegen()` function that will eventually traverse the AST and generate IR. Include a TODO note to implement actual code generation.  
- In \*\*Cargo.toml\*\*, add the `inkwell` dependency (and any other suggested crate like `anyhow` for error handling or `clap` for CLI parsing, if needed). Ensure the correct LLVM feature for inkwell is enabled (e.g., `llvm16-0` or as appropriate).  
- In `main.rs`, set up argument parsing (e.g., using `std::env` or Clap if included) to accept a source file, and call the lexer -> parser -> codegen pipeline functions in order. For now, these can just demonstrate the flow (like call the stub functions and print a message or dummy output).  
- Ensure all modules are referenced in the project (using `mod` statements or in `lib.rs` if a library).  
  
Provide the above as a ready-to-compile project. Use `// TODO` comments in places where functionality will be filled in. The code should compile (even though functionality is incomplete) so that we have a starting scaffold.

Copy the above prompt (everything in the markdown block) into Cursor’s Composer (the chat/command input after hitting Ctrl+I). The instructions are detailed, which helps the AI agent understand the full scope. When you execute it, Cursor’s agent will likely do the following:

* Create a new Cargo project (it might either run cargo new for you, or propose the file structure directly in a diff view). It may ask for confirmation to run commands – allow it to initialize the project if prompted.
* Generate the files: main.rs, lexer.rs, parser.rs, ast.rs, codegen.rs, etc., populating them with the requested boilerplate code.
* Edit the Cargo.toml to add dependencies (like inkwell and others).
* Possibly open a diff or preview of the changes for you to **Accept**. Review the AI’s output; you can usually **“Accept All”** changes if it looks good, and the files will be created in your workspace[[20]](https://forum.cursor.com/t/multi-file-edits-0-37-update/6425/24#:~:text=And%20then%20after%20clicking%20%E2%80%98Accept,All%E2%80%99)[[21]](https://forum.cursor.com/t/agent-create-new-files/408#:~:text=,Th%E2%80%A6).

Cursor’s multi-file edit capability was demonstrated by another user who prompted it to create a whole website project with HTML/CSS/JS files – and Cursor successfully generated all files and structure automatically[[19]](https://forum.cursor.com/t/agent-create-new-files/408#:~:text=%40Apologin%20%20%2014%20%2C). So, it should handle creating a multi-file Rust project as well. If the agent doesn’t automatically create the Cargo project directory for you, you might need to manually do cargo new langone in the terminal, then open that folder in Cursor and run the Composer prompt for the internal file structure. (This is because sometimes the AI might only suggest changes; if so, create the project yourself, then let AI fill in files.)

**After generation:** You’ll have a baseline Rust project for LangOne. Open each file to review the content. Build the project by opening the integrated terminal in Cursor (Ctrl+) and runningcargo build`. The project should compile (since we asked the AI to ensure it compiles). This scaffold gives you a huge head start: all major components are in place with stub functions and TODOs.

Now you can iteratively implement each part, perhaps also with AI assistance. For example, you can go into lexer.rs, write a comment like // TODO: implement lexing for LangOne and ask Cursor (via a prompt or using the inline Ctrl+K feature) to implement that function. Cursor’s **Inline Edit** (Ctrl+K) lets you select a piece of code or an empty function and describe what you want – the AI will then fill in the code for you[[22]](https://cursor.com/en/features#:~:text=Inline%20Edit). Similarly, you can use Cursor’s **chat** to ask questions about how to use inkwell, get suggestions for parsing libraries, etc., with references (@web search or documentation if needed[[23]](https://cursor.com/en/features#:~:text=Get%20up,information%20to%20answer%20your%20question)).

Throughout development, keep leveraging Cursor’s AI features: - **Autocomplete**: As you write code, Cursor will suggest completions (its AI-powered autocomplete is very advanced, sometimes anticipating multi-line edits[[24]](https://cursor.com/en#:~:text=The%20Cursor%20tab%20completion%20while,at%20the%20speed%20of%20thought)[[25]](https://cursor.com/en#:~:text=I%20love%20writing%20code%20and,tab%27%20more%20than%20anything%20else)). Hit <kbd>TAB</kbd> to accept suggestions. - **Chat/Agent**: Use the sidebar chat to ask for help or improvements. For example, *“How do I implement a recursive descent parser for this grammar?”* – the AI can generate code or explain concepts with references. - **Refactoring**: You can instruct Cursor to refactor code by selecting it and describing the change. For instance, highlight a function and say “Make this function asynchronous” or similar; the AI will modify code accordingly if possible. - **Error fixing**: If you see a compiler error or clippy lint, the AI can often suggest a fix. Cursor can *“detect lint errors automatically and apply fixes”*[[26]](https://cursor.com/en/features#:~:text=Loops%20on%20Errors).

By combining your expertise with Cursor AI’s speed, you’ll iterate quickly. As the **Chief Architect**, you define how each component should work, and let the AI handle boilerplate and repetitive parts – a synergy also envisioned in the LangOne manifesto (using AI agents as a “force multiplier” in development[[27]](file://file-TxnvydkFLeAZ9yxBdh7CYo#:~:text=,documentation)[[28]](file://file-TxnvydkFLeAZ9yxBdh7CYo#:~:text=Leverage%20AI%20to%20Move%20Fast,syntax%20features%20before%20full%20implementation)).

## 4. Final Thoughts

You now have a complete development toolchain and a plan to start building LangOne:

* **Operating System**: Windows 11 (with WSL2 enabled for Docker).
* **Editor/IDE**: Cursor AI Code Editor (for an AI-augmented coding experience).
* **Language & Compiler**: Rust (installed via rustup) – chosen for safe and fast systems programming.
* **Backend Infrastructure**: LLVM (installed, and used via Inkwell in Rust) – provides powerful optimization and multi-platform native code generation (crucial for LangOne’s performance goals).
* **Build & Dependency Manager**: Cargo – to build the project and manage Rust crates (Inkwell, etc.).
* **Version Control**: Git – for managing your codebase history and collaboration.
* **Containerization**: Docker – ready for use when you containerize the compiler or services (e.g., a LangOne playground server).
* **AI Assistance**: Cursor’s AI features – to speed up coding, enforce best practices, and even generate code based on high-level descriptions.

With this setup, you can focus on designing LangOne’s language features and let your tools handle the rest. **Happy coding, and enjoy building LangOne with your AI pair-programmer!** 🚀

**Sources:**

* Cursor AI official site – Windows download options[[4]](https://cursor.com/downloads#:~:text=Windows), and features of the AI editor[[18]](https://cursor.com/en/features#:~:text=Agent)[[19]](https://forum.cursor.com/t/agent-create-new-files/408#:~:text=%40Apologin%20%20%2014%20%2C).
* Rust documentation – installing Rust on Windows (Rustup) and MSVC requirements[[8]](https://www.rust-lang.org/tools/install#:~:text=It%20looks%20like%20you%E2%80%99re%20running,Other%20Installation%20Methods)[[9]](https://www.rust-lang.org/tools/install#:~:text=Windows%20considerations)[[10]](https://learn.microsoft.com/en-us/windows/dev-environment/rust/setup#install-visual-studio-code#:~:text=Rust%20works%20very%20well%20on,apps%20for%20Windows%20using%20Rust).
* Microsoft Learn – advice on Windows Rust development (Visual Studio, Git, rust-analyzer)[[13]](https://learn.microsoft.com/en-us/windows/dev-environment/rust/setup#install-visual-studio-code#:~:text=While%20installing%20Visual%20Studio%2C%20there,simpler%20to%20select%20all%20three)[[6]](https://learn.microsoft.com/en-us/windows/dev-environment/rust/setup#install-visual-studio-code#:~:text=1,Visual%20Studio%20Code%20for%20Windows).
* Stack Overflow – installing LLVM on Windows via winget[[14]](https://stackoverflow.com/questions/76862060/how-to-install-llvm-on-windows#:~:text=Native%20Windows%20method%20without%20any,rd%7D%20party%20tools).
* Rust community – using Inkwell vs llvm-sys for LLVM in Rust[[2]](https://users.rust-lang.org/t/is-inkwell-complete-enough/79511#:~:text=I%20would%20recommend%20Inkwell%2C%20it%27s,because%20the%20crate%20is%20unstable).
* LangOne Manifesto (user file) – key goals like native code compilation (JIT/AOT)[[1]](file://file-TxnvydkFLeAZ9yxBdh7CYo#:~:text=,Flutter%20for%20instant%20feedback%20loops).

[[1]](file://file-TxnvydkFLeAZ9yxBdh7CYo#:~:text=,Flutter%20for%20instant%20feedback%20loops) [[27]](file://file-TxnvydkFLeAZ9yxBdh7CYo#:~:text=,documentation) [[28]](file://file-TxnvydkFLeAZ9yxBdh7CYo#:~:text=Leverage%20AI%20to%20Move%20Fast,syntax%20features%20before%20full%20implementation) langone\_manifesto.md

<file://file-TxnvydkFLeAZ9yxBdh7CYo>

[[2]](https://users.rust-lang.org/t/is-inkwell-complete-enough/79511#:~:text=I%20would%20recommend%20Inkwell%2C%20it%27s,because%20the%20crate%20is%20unstable) Is Inkwell complete enough? - The Rust Programming Language Forum

<https://users.rust-lang.org/t/is-inkwell-complete-enough/79511>

[[3]](https://www.reddit.com/r/rust/comments/ej0yn4/whats_the_best_way_to_generate_llvmir_in_rust/#:~:text=If%20you%20don%27t%20have%20to,should%20also%20consider%20using%20cranelift) What's the best way to generate LLVM-IR in Rust? : r/rust

<https://www.reddit.com/r/rust/comments/ej0yn4/whats_the_best_way_to_generate_llvmir_in_rust/>

[[4]](https://cursor.com/downloads#:~:text=Windows) Downloads | Cursor - The AI Code Editor

<https://cursor.com/downloads>

[[5]](https://cursor.com/en#:~:text=,Cursor%20is%20SOC%202%20certified) [[24]](https://cursor.com/en#:~:text=The%20Cursor%20tab%20completion%20while,at%20the%20speed%20of%20thought) [[25]](https://cursor.com/en#:~:text=I%20love%20writing%20code%20and,tab%27%20more%20than%20anything%20else) Cursor - The AI Code Editor

<https://cursor.com/en>

[[6]](https://learn.microsoft.com/en-us/windows/dev-environment/rust/setup#install-visual-studio-code#:~:text=1,Visual%20Studio%20Code%20for%20Windows) [[7]](https://learn.microsoft.com/en-us/windows/dev-environment/rust/setup#install-visual-studio-code#:~:text=2,Ctrl%2BShift%2BX) [[10]](https://learn.microsoft.com/en-us/windows/dev-environment/rust/setup#install-visual-studio-code#:~:text=Rust%20works%20very%20well%20on,apps%20for%20Windows%20using%20Rust) [[11]](https://learn.microsoft.com/en-us/windows/dev-environment/rust/setup#install-visual-studio-code#:~:text=If%20you%27re%20curious%20about%20the,and%20their%20dependencies) [[12]](https://learn.microsoft.com/en-us/windows/dev-environment/rust/setup#install-visual-studio-code#:~:text=Install%20Visual%20Studio%20,the%20Microsoft%20C%2B%2B%20Build%20Tools) [[13]](https://learn.microsoft.com/en-us/windows/dev-environment/rust/setup#install-visual-studio-code#:~:text=While%20installing%20Visual%20Studio%2C%20there,simpler%20to%20select%20all%20three) [[17]](https://learn.microsoft.com/en-us/windows/dev-environment/rust/setup#install-visual-studio-code#:~:text=required%20that%20we%20feel%20it%27s,simpler%20to%20select%20all%20three) Set up your dev environment on Windows for Rust | Microsoft Learn

<https://learn.microsoft.com/en-us/windows/dev-environment/rust/setup>

[[8]](https://www.rust-lang.org/tools/install#:~:text=It%20looks%20like%20you%E2%80%99re%20running,Other%20Installation%20Methods) [[9]](https://www.rust-lang.org/tools/install#:~:text=Windows%20considerations) Install Rust - Rust Programming Language

<https://www.rust-lang.org/tools/install>

[[14]](https://stackoverflow.com/questions/76862060/how-to-install-llvm-on-windows#:~:text=Native%20Windows%20method%20without%20any,rd%7D%20party%20tools) installation - How to install llvm on Windows? - Stack Overflow

<https://stackoverflow.com/questions/76862060/how-to-install-llvm-on-windows>

[[15]](https://createlang.rs/01_calculator/basic_llvm.html#:~:text=The%20code%20is%20available%20in,0%22%5D%60%20in%20inkwell) Basic Example - Create Your Own Programming Language with Rust

<https://createlang.rs/01_calculator/basic_llvm.html>

[[16]](https://docs.docker.com/desktop/setup/install/windows-install/#:~:text=,Windows%2010%20or%20Windows%2011) Windows | Docker Docs

<https://docs.docker.com/desktop/setup/install/windows-install/>

[[18]](https://cursor.com/en/features#:~:text=Agent) [[22]](https://cursor.com/en/features#:~:text=Inline%20Edit) [[23]](https://cursor.com/en/features#:~:text=Get%20up,information%20to%20answer%20your%20question) [[26]](https://cursor.com/en/features#:~:text=Loops%20on%20Errors) Features | Cursor - The AI Code Editor

<https://cursor.com/en/features>

[[19]](https://forum.cursor.com/t/agent-create-new-files/408#:~:text=%40Apologin%20%20%2014%20%2C) [[21]](https://forum.cursor.com/t/agent-create-new-files/408#:~:text=,Th%E2%80%A6) Agent create new files? - How To - Cursor - Community Forum

<https://forum.cursor.com/t/agent-create-new-files/408>

[[20]](https://forum.cursor.com/t/multi-file-edits-0-37-update/6425/24#:~:text=And%20then%20after%20clicking%20%E2%80%98Accept,All%E2%80%99) Multi-file edits - 0.37 update - #24 by litecode - Feedback - Cursor - Community Forum

<https://forum.cursor.com/t/multi-file-edits-0-37-update/6425/24>