# Prompts for Learning to Code with AI

## Introduction

Use these prompts to get started learning how to code from scratch or simply to have AI write and modify code for you (via tools such as Cursor, Windsurf, and VS Code Copilot). This set of prompts should help you build your understanding of practical software engineering skills and concepts.

At the bottom of this doc is a glossary of high-level terms that you may want to plug into the prompts to learn more.

\*\*Important note\*\*: you \*\*\*need\*\*\* to use a high-quality model such as \*\*Claude Sonnet 3.5\*\* to make this work. Using GPT-3.5, Llama, or other similar models will only leave you frustrated and stuck. High-end models will generate much better code and explain things much more clearly (and correctly!). Also, do not be deceived by benchmarks - models such as OpenAI o3 may perform exceptionally well on coding benchmarks but are not necessarily the best choice for real-world coding.

## Exploring Concepts

- \*\*Starter Prompts\*\*

- What is [X] / How does [X] work / What does [X] mean? How do I use [X]? `<--- simple but super effective`

- I am a new developer learning to code. Please help me create a basic [app idea] web app using the MERN stack

- I am a new developer learning how to code. Please explain what [X] is and what role it plays in building software.

- What are the alternatives to [X]? When and why would I use them?

- I am a new developer interested in understanding the difference between [X] and [Y]. Could you explain?

- In what scenarios would I choose [X] over [Y]? Please explain and provide examples.

- Please explain what each line in this code does: ```[paste the code]```

- How do you [accomplish certain task]? Answer with a high level explanation and specific steps.

- \*\*Follow-up Prompts\*\*

- Please rephrase this in a few simple sentences and include an example that would be really easy to understand.

- I still don't get it; can you explain it in a simpler or different way

- How can I use this? / What do I do with this? / Where do I put this?

- What should I do next? / How can I build on this? / Are there any other ways to do this?

- Are there any concerns or tradeoffs with this approach?

- [Open a new Chat and start over] ```<--- good when AI gets stuck rehashing the same ideas```

\*\*Note\*\*: Remember to always give the LLM (AI) the relevant context, whether it is the code you are investigating or your relative understanding and experience working with a certain technology.

## Writing Code

- \*\*Planning & Design\*\*

- I want to build [feature]. Here's how it should work: [list requirements]

- What technologies would you recommend for building [feature]?

- Let's use [tech stack]. Can you help design the solution before we implement?

- \*\*Implementation\*\*

- Please create [feature] using [technology list]

- Here's some working sample code to follow: ```[code]```

- Please implement the following requirements:

1. [Requirement #1]

2. [Requirement #2]

3. [Requirement #3]

- \*\*Modifying Code\*\*

- I have this code: ```[code]```. Please modify it to [do X].

- Please refactor this code to keep files under 300 lines: ```[code]```

- Please add error handling to this code: ```[code]```

- That code doesn't work because it also needs to [do X]. Can you modify it to do that?

- \*\*If working outside of a code editor tool\*\*

- I had to make a change to that code so the latest code is: ```[code]```

- Please state the \*full\* code in your response

## Code Quality

- \*\*Reviews and Improvements\*\*

- Can you review this implementation and suggest any improvements?

- Are there any edge cases we haven't considered?

- How can we make this code more secure and performant?

- Can you check if this is following best practices for [technology]?

- What potential security vulnerabilities should we address?

- \*\*Documentation\*\*

- Please add clear comments explaining this code

- Can you create documentation for this feature in our README?

- Please explain this code's functionality for new developers

- Can you document the expected inputs and outputs?

## Troubleshooting

- \*\*Basic Troubleshooting\*\*

- I tried to run that but got this error: ```[error]``` in the ```[where you saw the error]```

- It's still not working; what should I check to make sure I am doing this the right way?

- I got this error: ```[error]```. Is there a different way we can do this to avoid this error?

- I tried [X, Y, Z] but it is still not working. Any other ideas?

- \*\*Advanced Troubleshooting\*\*

- Please add logs throughout this code so we can track exactly what's happening (The Beaver Method): ```[code]```

- Here are the logs from running that code: ```[logs]``` Can you help me understand where things went wrong?

- Let's try a radically different approach to solving this problem

- Can you take a step back and explain exactly what this code is doing right now?

- Here's a screenshot of what's happening: [screenshot]

## Learning Strategies

- \*\*Building Understanding\*\*

- I understand [concept] but am unclear about [concept]. Can you connect these?

- Can you explain this at a high level first, then break it down?

- Please show me some examples of how this works in practice

- How would you architect a [type of app]? What are the key considerations?

- \*\*Technology Deep Dives\*\*

- What are the core components of [technology]?

- How does [technology] fit into the bigger picture of software development?

- Can you show me a simple example of [technology] in action?

- What are the most important concepts I should understand about [technology]?

## Glossary / Terminology

These are various terms that you might not be familiar with but can act as starting points for learning with AI. Ask ChatGPT or Claude about these concepts to start building an understanding of how they all fit together. Feel free to incorporate these into your prompts to improve the quality

### Practical Terms (web development)

- \*\*Frontend:\*\* The part of a website or web application that users interact with directly in their web browsers. It includes the design, layout, and some client-side logic of web pages.

- \*\*Backend:\*\* The server-side logic of a web application, dealing with data management, authentication, server logic, and application functionality that users don't directly interact with.

- \*\*Server:\*\* A physical or virtual machine (ultimately just a computer) that runs the backend code and serves the frontend to users. It processes requests, runs the application's backend logic, and manages resources.

- \*\*Environment:\*\* The setup or context in which an application or development tool runs. In real-world settings, there will typically be a development, testing / staging, and production environments, each with its own configurations and purposes. Your own computer would be categorized as your "local environment".

- \*\*API (Application Programming Interface):\*\* A part of the backend server that exists to enable different parts of your software to communicate with each other. An API typically consists of 'endpoints' that handle specific requests to the backend server.

- \*\*Database:\*\* A backend component used for long-term data storage. Databases allow for secure data retrieval, insertion, updates, and deletion.

- \*\*Authentication:\*\* The process by which an application verifies the identity of a user, typically through login credentials, ensuring that users are who they claim to be.

- \*\*Authorization:\*\* After authentication, the application determines what resources and operations the authenticated user has permission to access and perform within an application.

- \*\*Framework:\*\* A comprehensive set of tools and libraries designed to simplify the development process of software projects by providing a structured foundation to build upon.

- \*\*Library:\*\* A collection of pre-written code that developers can utilize without having to write code from scratch. Libraries typically server a specific purpose, such as integrating with a particular software or implementing commonly used logic.

- \*\*Package Manager:\*\* A tool that automates the process of installing, updating, configuring, and removing packages/libraries from a computer in a consistent manner.

- \*\*Component:\*\* A reusable element in UI design OR a modular unit in software architecture, each with its own functionality.

- \*\*CDN (Content Delivery Network):\*\* A network of servers distributed across different locations that delivers content (files) efficiently to users based on their geographical location.

- \*\*DNS (Domain Name System):\*\* Essentially the phonebook of the Internet, translating human-friendly domain names (like [www.example.com](http://www.example.com/)) into IP addresses that computers use to identify each other on the network.

- \*\*Proxy:\*\* A server that acts as an intermediary for requests from clients seeking resources from other servers, providing various functionalities such as load balancing, privacy, or security.

- \*\*Domain:\*\* The name that identifies a website on the Internet. It's part of the URL that allows users to easily find and access websites.

- \*\*HTTP (Hypertext Transfer Protocol):\*\* The foundational protocol used by the World Wide Web, defining how messages are formatted and transmitted, and how web servers and browsers should respond to various commands.

- \*\*Cookies:\*\* Small pieces of data sent from a website and stored on the user's computer by the user's web browser while the user is browsing, used to remember information about the user for their next visit.

- \*\*Sessions:\*\* A way to store information across multiple page requests for a particular user, enabling persistence of state across the web application.

- \*\*Service:\*\* A function or set of functions provided by one part of a software system to others, focusing on accomplishing specific tasks. Typically a specific program running somewhere that can be invoked.

- \*\*Microservice:\*\* An architectural style that structures an application as a collection of small, autonomous services, each focusing on a single function or business capability.

- \*\*Container:\*\* Lightweight, standalone, executable software packages that include everything needed to run a piece of software, including the code, runtime, system tools, libraries, and settings.

- \*\*Serverless:\*\* An approach used by cloud web hosting providers to deploy/manage backend servers in a way that enables the real-time dynamic provisioning and allocation of servers. Enables developers to build and run applications and services without thinking about individual servers.

- \*\*Cloud:\*\* Computing services offered over the internet (the cloud), including servers, storage, databases, networking, software, analytics, and intelligence, to offer faster innovation, flexible resources, and economies of scale.

- \*\*Network:\*\* A group of interconnected computers (servers), and devices that can exchange data and resources with each other.

- \*\*ETL (Extract, Transform, Load):\*\* A process in data warehousing that involves extracting data from various sources, transforming it into a suitable format, and loading it into a destination database for analysis or reporting.

- \*\*Data Normalization:\*\* The process of organizing the columns (attributes) and tables (relations) of a database to minimize data redundancy and improve data integrity.

- \*\*Logging:\*\* The process of recording activities and events in a software application or system, which is crucial for debugging and monitoring the application's behavior in development and production environments.

- \*\*Localhost:\*\* The standard hostname for the local computer being used, often for testing software locally (e.g. your computer).

- \*\*Port:\*\* A numerical identifier in networking used to route data to specific programs or services on a computer.

### Computer Science Terms (relevant for practical use)

- \*\*Algorithm:\*\* A step-by-step procedure or set of rules designed to perform a specific task or solve a particular problem.

- \*\*Language:\*\* In the context of programming, a language is a set of syntax rules and structures used to write software programs that computers can execute.

- \*\*Executable:\*\* A type of file that contains a program capable of being executed or run as a program in the computer.

- \*\*Function:\*\* A block of organized, reusable code that performs a single action or returns a value.

- \*\*Variable:\*\* A symbolic name associated with a value and whose associated value can be changed.

- \*\*Environmental Variable:\*\* A variable that is set outside of the program, typically through the operating system, to pass configuration information to the application.

- \*\*Loop:\*\* A programming construct that repeats a block of code multiple times until a specified condition is met.

- \*\*Infinite Loop:\*\* A loop that continues indefinitely because the loop condition is never satisfied or fails to become false.

- \*\*Call Stack:\*\* A data structure used by programming languages to keep track of active subroutines or functions in a program's execution, where the last function called is the first to be completed and removed.

- \*\*Error:\*\* An issue that interrupts the expected flow of a program, often due to syntax, logic, or resource errors.

- \*\*Stack Trace:\*\* A report showing the sequence of function calls that led to an error or exception, used for debugging.

- \*\*Conditional Statement:\*\* A programming feature that performs different actions based on whether a specified condition evaluates to true or false. (if / else statement)

- \*\*Recursion:\*\* A programming technique where a function calls itself in order to solve a problem.

- \*\*Syntax:\*\* The set of rules that defines the correct combination of symbols that are considered to be a valid part of the language.

- \*\*Queue:\*\* A collection used to manage items in a sequence where items are added and removed according to specific algorithms, with FIFO (First In, First Out) being the most common, but others like LIFO (Last In, First Out) and priority-based removal are also used, depending on the application's requirements.

- \*\*Runtime:\*\* The period during which a program is running, starting from program execution to program termination.

- \*\*Compilation:\*\* The process of translating code written in a high-level programming language into a machine level language that can be executed by the computer.

- \*\*Data Structure:\*\* Organized ways of storing and organizing data in a computer so that it can be accessed and modified efficiently.

- \*\*Memory:\*\* Refers to the temporary storage used by a computer to run programs and process data. "In memory" signifies that data is stored in the main, directly accessible RAM area.

- \*\*Cache:\*\* A temporary storage area that allows for the fast retrieval of data by storing copies of frequently accessed data items or computations.

- \*\*Async vs Sync:\*\* Asynchronous programming allows a program to do more than one thing at a time, while synchronous programming has tasks run in sequence, causing subsequent tasks to wait until the current task finishes.

- \*\*Dependency:\*\* A piece of software or a module that another piece of software relies on to function properly.

- \*\*Error Handling:\*\* The process of responding to and managing errors in a program, often through the use of try-catch blocks or similar constructs.

- \*\*State Management:\*\* The practice of managing the state of one or multiple user interfaces controls like text fields, OK buttons, etc., in a consistent manner.

- \*\*Serialization/Deserialization:\*\* The process of converting an object into a format that can be stored or transmitted (serialization) and the process of converting that format back into an object (deserialization).

- \*\*Multithreading:\*\* A type of execution that allows a single process to have multiple threads of execution running concurrently.

- \*\*Unit Tests:\*\* Tests that cover the smallest parts of an application, like individual functions or methods.

- \*\*Functional Tests:\*\* Tests that assess the specific functionality of an application or its modules, often involving testing UI interactions or API integrations.

- \*\*Performance:\*\* Refers to how fast a web page or application loads and runs. It is critical to user experience and is influenced by various factors, including server speed, database optimization, and efficient coding practices.

- \*\*Load Balancing:\*\* The distribution of network or application traffic across multiple servers to ensure no single server becomes overwhelmed, improving reliability and availability.

### Technologies

- \*\*HTML:\*\* The standard markup language for documents designed to be displayed in a web browser. It forms the structure of web pages.

- \*\*CSS:\*\* A style sheet language used for describing the look and formatting of documents written in HTML, controlling the layout of multiple web pages all at once.

- \*\*JavaScript:\*\* A programming language that allows you to implement complex features on web pages, being the only language that can be executed in web browsers, enabling dynamic content, interactive maps, animated 2D/3D graphics, scrolling video jukeboxes, etc.

- \*\*TypeScript:\*\* JavaScript but with clearly defined data types which make code more resilient and easier to understand (when implemented properly). Works particularly well with AI.

- \*\*Frontend Frameworks\*\*: (e.g. React, Vue, Angular).These are JavaScript frameworks and libraries designed to simplify the development of the user interface (UI) of web applications. React is known for its virtual DOM feature for high performance, Angular for its comprehensive solution including tools for routing, forms, HTTP client, and more, and Vue for its simplicity and progressive nature, making it easily adoptable for parts of existing projects.

- \*\*NodeJS:\*\* An open-source, cross-platform JavaScript runtime environment that executes JavaScript code outside a web browser, allowing for the development of server-side and networking applications.

- \*\*NextJS:\*\* A full-stack JavaScrupt framework that uses server-side-rendering to generate html pages just-in-time. Practically speaking, this enables web crawlers such as Google to properly index pages without loading JavaScript, at the cost of greater implementation complexity.

- \*\*Express:\*\* A minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications, making it easy to build single-page, multi-page, and hybrid web applications.

- \*\*Databases\*\*: (e.g. MongoDB, Postgres) MongoDB is a popular NoSQL database known for its flexibility and scalability, while PostgreSQL is a powerful open-source relational database system. The key difference lies in how they store data: relational databases structure data into predefined tables and rows, whereas nonrelational databases store data without a fixed schema, often making them more flexible and scalable for certain types of applications.

- \*\*Docker:\*\* A platform as a service (PaaS) that uses OS-level virtualization to deliver software in packages called containers, allowing developers to package applications with their dependencies and deploy as one package.

- \*\*Git:\*\* A distributed version-control system for tracking changes in source code during software development, enabling multiple developers to work on a project concurrently.

- \*\*Cloud Providers:\*\* (e.g. AWS, Azure, GCP) These services offer a wide range of cloud computing resources and services. AWS (Amazon Web Services) is known for its robust, scalable, and affordable cloud solutions. Microsoft Azure offers a wide range of cloud services supporting various operating systems, databases, and developer tools. Google Cloud Platform provides cloud computing services that run on the same infrastructure Google uses internally for its end-user products.

- \*\*Design System:\*\* (e.g. MUI, ShadCN, Chakra, Bootstrap, Tailwind) A set of standards for design and code along with components that unify both practices. These systems help teams develop digital products faster by making design reusable.

- \*\*Redis:\*\* An in-memory data structure store, used as a database, cache, and message broker. Redis supports data structures such as strings, hashes, lists, sets, sorted sets with range queries, bitmaps, hyperloglogs, geospatial indexes with radius queries, and streams.

- \*\*Websocket:\*\* A communication protocol that provides full-duplex communication channels over a single TCP connection, enabling web servers and clients to exchange data more efficiently, facilitating real-time data transfer and interaction in web applications.

- \*\*GraphQL:\*\* A query language for APIs that allows clients to request exactly the data they need, making it possible to aggregate data from multiple sources with a single request. Generally speaking, this should only be used in specific situations where performance improvements are worth the increased code complexity.

- \*\*Webpack:\*\* A static module bundler for JavaScript applications, which bundles JavaScript files for usage in a browser, yet it is also capable of transforming, bundling, or packaging just about any resource or asset.

- \*\*Nginx:\*\* A high-performance reverse proxy and load balancer that sits "in front" of applications and provides increased security, scalability, and speed for web applications.

- \*\*LLM (Large Language Models):\*\* Advanced AI models capable of understanding and generating human-like text, facilitating tasks such as content creation, code generation, and natural language processing.

- \*\*Flutter:\*\* An open-source UI software development kit by Google used for building natively compiled applications for mobile, web, and desktop from a single codebase.

- \*\*Postman:\*\* A collaboration platform for API development, which simplifies each step of building an API and streamlines collaboration so you can create better APIs—faster.

- \*\*curl:\*\* A command-line tool used for transferring data with URLs. It supports various protocols including HTTP, HTTPS, FTP, and more, making it a versatile tool for testing, downloading files, and making API requests directly from the terminal.

- \*\*Terminal:\*\* A text-based interface to the system, allowing for the execution of commands, scripts, and programs. It provides direct access to the underlying operating system through a command-line interface (CLI), essential for software development, system administration, and troubleshooting.

- ## Process

- \*\*Version Control:\*\* A system that records changes to a file or set of files over time so that specific versions can be recalled later. It's essential for collaborative development projects.

- \*\*SDLC (Software Development Life Cycle):\*\* A process for planning, creating, testing, and deploying an information system, with phases including requirements analysis, design, implementation, testing, deployment, and maintenance.

- \*\*Testing:\*\* The practice of executing a program or application with the intent of finding errors and verifying that the software meets the specified requirements.

- \*\*Accessibility (a11y):\*\* The practice of making your websites usable by as many people as possible, including those with impairments or disabilities, ensuring all users have equal access to information and functionality.

- \*\*Internationalization (i18n):\*\* The practice of designing software to facilitate easy adaptation to different languages and regions, ensuring global usability.

- \*\*Responsive Design:\*\* A design approach aimed at making web pages render well on a variety of devices and window or screen sizes, ensuring usability and satisfaction across different devices.

- \*\*Deployment:\*\* The process of delivering a software application to a live production environment where it can be accessed by users.

- \*\*Monitoring:\*\* The continuous observation of a software application's operation and performance, often using specialized tools, to ensure it functions as expected and to identify issues as they arise.

- \*\*Code Review:\*\* The process of examining written code by one or multiple developers with the purpose of finding bugs or errors, and improving the quality of the code.

- \*\*Continuous Integration (CI):\*\* A development practice where developers integrate code into a shared repository frequently, preferably several times a day, to catch integration errors quickly.

- \*\*Continuous Delivery (CD):\*\* A software development practice where code changes are automatically built, tested, and prepared for a release to production, enabling rapid and reliable software delivery.

- \*\*DevOps:\*\* A set of practices that combines software development (Dev) and IT operations (Ops), aiming to shorten the system development life cycle and provide continuous delivery with high software quality.