

Programming Today

Reading time: ~10 min · Prereqs: Lessons 01–04 · *Snapshot 2026 — AI assistants and IDE landscape change quickly; see [refresh-checklist.md](#)*.

Why this matters to you

"Programmers" don't sit in dark rooms typing arcane symbols. In 2026, writing software looks more like an editor-meets-chat-meets-version-control-meets-AI-assistant — a craft that has changed more in the last three years than in the previous twenty. Even if you'll never write a line yourself, knowing what the daily activity actually looks like makes you a much better partner to people who do: product managers, designers, executives, technical recruiters, and anyone who has to estimate, scope, or sell software. This lesson is a quick tour of the toolkit, the languages, and the workflow that defines what "writing code" means today.

1. What a programming language really is

A programming language is a formal grammar — strict enough that a tool called a **compiler** or an **interpreter** can read it and tell the computer exactly what to do. There are hundreds; most paid jobs use one of about ten. Languages aren't religions — they're tools. Most working engineers move fluently between two and four daily, usually in some mix like "Python at work plus a sprinkle of SQL, a few bash scripts, and a little JavaScript when configuring the build."

Compiled languages (C, C++, Rust, Go) are translated to machine code ahead of time and run very fast. Interpreted languages (Python, Ruby, JavaScript) are read and executed line-by-line at runtime, which trades some speed for flexibility. The line has blurred — modern JavaScript engines compile on the fly, and Python has just-in-time options — but the mental model still helps.

2. The 2026 language landscape

A rough roll-call of what teams actually hire for, *as of 2026*:

- **Python** — everywhere in data, AI, scripting, and a huge slice of backends. If you only ever learn one language, this is the safest pick.
- **JavaScript** and its typed sibling **TypeScript** — the only languages that run natively in browsers, and now powering most backends too via Node.js. TypeScript has displaced bare JavaScript on most new projects.
- **Java** — the workhorse of enterprise systems and Android applications. Still enormous in banking, telecom, and large e-commerce.
- **C#** — the Microsoft / Unity-games equivalent; dominant in Windows-heavy enterprises and a huge slice of game dev.
- **Go** — dominates cloud infrastructure (Docker, Kubernetes, much of the modern CLI ecosystem). The default for new infra tooling.
- **Rust** — replacing C/C++ in performance-critical and memory-safety-critical spots; the darling of systems engineers, embedded teams, and the crypto world.
- **Swift** and **Kotlin** — own iOS and Android development respectively.
- **C** and **C++** — still huge in operating systems, browsers, game engines, and embedded devices.
- **SQL** — its own thing; you'll meet it any time a database comes up (Lesson 07).

Trends *as of 2026*: Rust adoption is growing in places that used to be C++; Go is the default for new infra tooling; TypeScript has displaced bare JavaScript on most new projects; Python's grip on AI and data hasn't loosened. Refresh annually — [refresh-checklist.md](#) calls this out explicitly.

3. An IDE in 2026

An **IDE** ("integrated development environment") is a text editor with a lot of brains bolted on: syntax coloring, autocomplete, jump-to-definition, inline error squiggles, a debugger, git integration, a terminal, and — new in this generation — an AI assistant in the side panel. Think of it as the difference between Notepad and a full word processor: both let you type, but only one catches your typos in real time, suggests the next sentence, and remembers every revision.

The dominant choices *as of 2026* are **VS Code** (Microsoft, free, hugely extensible — sometimes 70%+ market share in developer surveys) and the **JetBrains** family

(PyCharm, IntelliJ, GoLand, WebStorm — paid, very polished, popular at companies with budget for tooling). A growing third camp is **VS Code-with-AI-baked-in**: forks like **Cursor** and **Windsurf** have native AI editing built into the editor rather than added as a plugin. The terminal-and-Vim crowd is still alive and well; **Neovim** with the right plugins is a full IDE for a meaningful slice of engineers, and it's the default in environments where you SSH into a remote machine and edit there.

4. AI coding assistants — the big shift

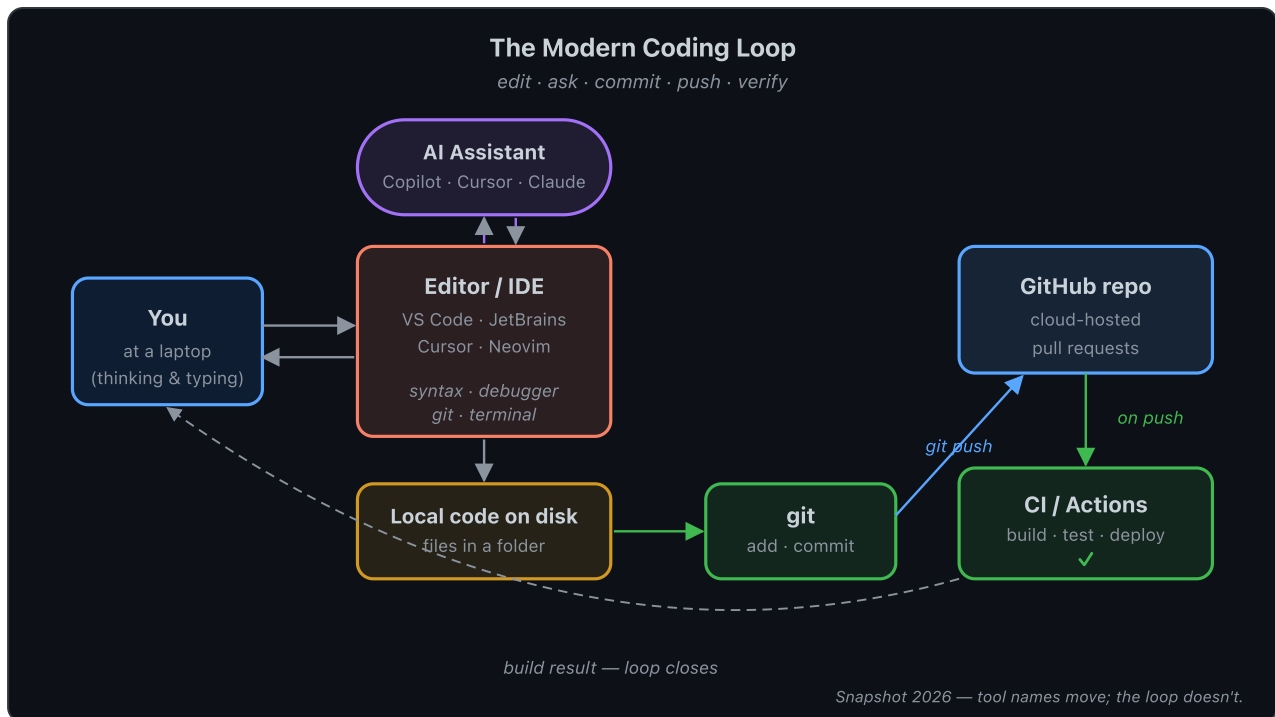
Three years ago this section didn't exist. *As of 2026*, most working engineers use an AI coding assistant daily. The notable players right now:

- **GitHub Copilot** — autocomplete-style; the first to land at scale.
- **Cursor** — popularized "chat plus agentic edits in the same editor."
- **Claude Code** — brought the same energy to the terminal.
- **Windsurf, Cody, Continue.dev** — all play in the same space with slightly different bets.

What they actually do: suggest the next 1–10 lines as you type, answer "what does this file do?" in a side panel, refactor across a whole repo on request, and — increasingly — *do tasks* agent-style, with the engineer reviewing the result. The engineer's job has shifted: less typing, more reading, more reviewing, more deciding what to build, more owning the output. *(This whole section is on `refresh-checklist.md` — the brand names will move.)*

5. The picture: the modern coding loop

Here's how the pieces fit together in a typical day. A person edits files in an IDE with an AI assistant whispering suggestions, snapshots the work into git, pushes it to GitHub, and watches automated pipelines run checks on every change.



6. Git in two paragraphs

Git is a version-control system: it remembers every change you make to a project. When you've changed a few files, you `git add` the ones you want to save, then `git commit` to snapshot them with a short message ("fix login bug"). You can branch off the main line, work on a side line in isolation, and merge back when ready. Each commit has a fingerprint; you can go back to any previous state in seconds. Think of it as "Track Changes" for code, except it scales to thousands of contributors on the same project without anyone overwriting each other.

GitHub (and its alternatives **GitLab**, **Bitbucket**, **Codeberg**) is the most popular place to host your git repositories online so teammates can pull a copy, propose changes (**pull requests** / **merge requests**), discuss them inline, and run automated checks (**GitHub Actions**) on every change. GitHub is also where most open-source software lives — including a meaningful share of the dependencies in the apps you use every day. An engineer's "public profile" today is often their GitHub page, not their resume.

What people argue about

- **Will AI replace engineers?** Three years in, AI assistants have made engineers significantly faster, especially at junior and grunt-work tasks. Headcount hasn't dropped meaningfully across the industry; ambition per engineer has

gone up. Predictions for the next 3–5 years range from "no further change" to "10x productivity again." Pick a camp at your peril.

- **TypeScript vs plain JavaScript.** TypeScript has won most new projects, but maintainers of widely-used libraries (e.g., Svelte famously dropped TS internally) sometimes argue the type-safety tax outweighs the benefit at certain scales. The debate is alive in every JS-shop hiring loop.
- **VS Code monoculture.** Microsoft's editor is now so dominant that some engineers worry about the platform risk of one vendor owning the editor, the AI assistant (GitHub Copilot), the cloud (Azure), and the largest code-hosting site (GitHub) all at once.

Reading on

- **Roadmap.sh** — roadmap.sh. Community-curated learning roadmaps for every common role (frontend, backend, DevOps, AI engineer). A solid first map of the territory.
- **GitHub's "Hello World" tutorial** — docs.github.com/get-started/quickstart. About 15 minutes, no prior knowledge needed; you'll have a real repo with a real commit when you're done.
- **learnxinyminutes.com** — every major language on one scrollable page, when you just want to see what the syntax looks like before committing to learning it.

LAST-REVIEWED: 2026-05-25