Purwadhika
Digital Technology School

Full Stack AI Software Development

# Intro to Git, Github, AI Tools and Exercise

Job Connector Program

## Outline

### Version Control Basics
Learn why developers use version control and how Git helps track changes in projects.

### Git Fundamentals
Understand Git’s workflow, architecture, and key commands for managing code history.

### GitHub for Collaboration
Discover how GitHub enables teamwork with branching, pull requests, and code reviews.

### AI Tools in Development
Explore how AI assistants accelerate coding and learning.

## What is Version Control?

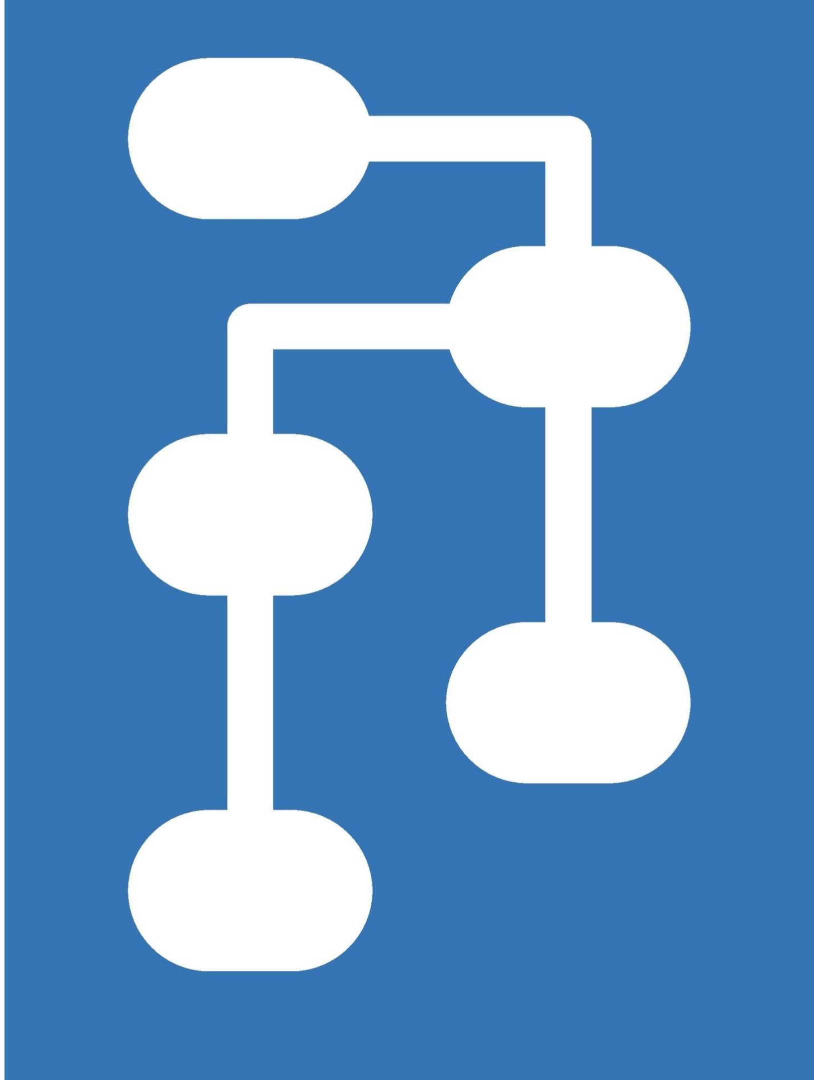
**A system to track and manage changes in code or documents over time.**

**Why It Matters**

- Keeps a history of changes
- Supports rollback to previous versions
- Allows collaboration between developers
- Prevents code from being overwritten
- Industry standard for software development

**Real-World Analogy**

- Like Google Docs with “track changes” but for code.



## Why Version Control is Important?

**Without Version Control:**

- Hard to track who changed what
- Risk of losing important work
- Difficult to collaborate safely
- Manual file versioning (e.g., final_code_v2_fix_reallyfinal.js)

**With Version Control:**

- Centralized project history
- Multiple people can work on the same project
- Easy recovery from mistakes
- Professional workflow used in real companies

## What is Git?

- Git = Distributed Version Control System ➔ https://git-scm.com/
- Created by Linus Torvalds (Linux creator) in 2005
- Works offline and stores the entire project history locally
- Enables branching, merging, and efficient collaboration

**Key Idea: Every developer has a complete copy of the repository.**

# Git Architecture & Workflow

## Main Concepts:

- **Working Directory** → Your actual files
- **Staging Area** → Prepares changes before saving
- **Repository (Local)** → Where commits are stored
- **Remote Repository** → Shared copy on a server **(GitHub, GitLab)**

## Workflow

**01** Write Code → **02** Stage → **03** Commit → **04** Push → **05** Collaborate

## What is GitHub?

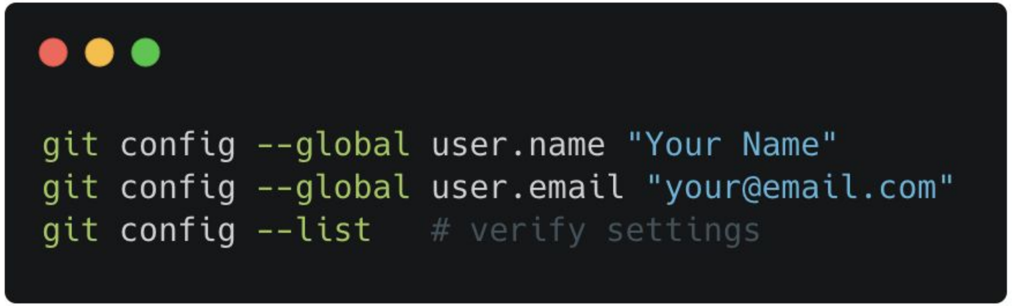
- A cloud-based Git hosting service ➔ https://github.com/
- Lets developers share and collaborate on projects
- Supports open-source contributions

**Features:**

- Repositories (public/private)
- Issues & Project Boards (task management)
- Pull Requests (code collaboration)
- GitHub Actions (automation/CI/CD)

## Setup Git

- Install Git
  - Download from [git-scm.com](https://git-scm.com)
  - Install using default settings
- Configure Git (first time only)



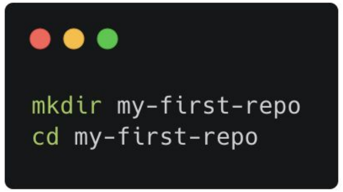
```
git config --global user.name "Your Name"
git config --global user.email "your@email.com"
git config --list   # verify settings
```

## Basic Git Commands

- Initialization & Setup
  - `git init` ➔ start a repo
  - `git config` ➔ set username/email
- Tracking Changes
  - `git status` ➔ check status
  - `git add <file>` ➔ stage changes
  - `git commit -m "message"` ➔ save snapshot
- Working with Remote
  - `git clone <url>` ➔ copy repo
  - `git push / git pull` ➔ sync with remote
- History & Branching
  - **git log** ➔ see history
  - **git branch** ➔ list branches
  - **git checkout -b feature-x** ➔ new branch

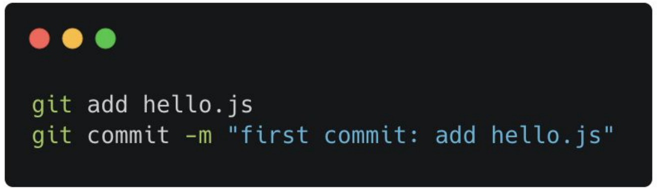
## Create Local Repository

- Create a new project folder:



```
mkdir my-first-repo
cd my-first-repo
```

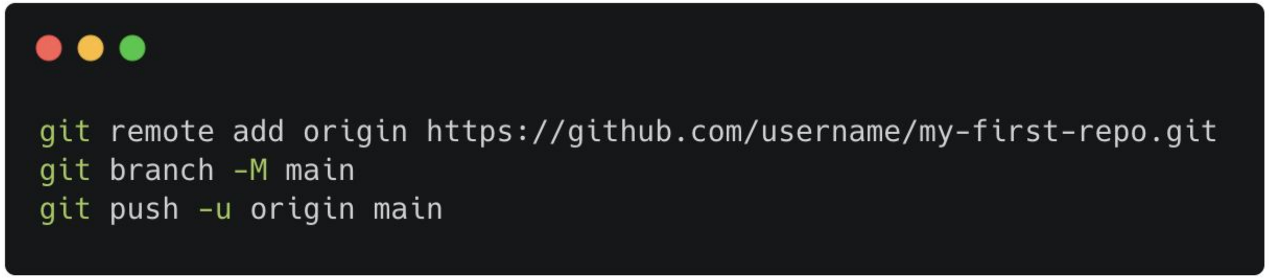
- Initialize Git:
  - `git init`
- Create a file:
  - `echo "Hello Git!" > hello.js`
- Stage & commit:



```
git add hello.js
git commit -m "first commit: add hello.js"
```

## Create & Connect GitHub Repository

- Log in to GitHub
- Click **New Repository**
- Fill in repo name ➔ e.g., **my-first-repo**
- Choose **Public** or **Private**
- Click **Create Repository**
- Connect **Local Repo** to **GitHub**:



```
git remote add origin https://github.com/username/my-first-repo.git
git branch -M main
git push -u origin main
```

## Conventional Commit Messages

**Why?**

- Keep commit history clear & consistent
- Make project easier to read, review, and automate (e.g., changelogs)

**Format:**

```
<type>(optional scope): <short description>
```

**Common Types:**

- **feat** ➔ new feature
- **fix** ➔ bug fix
- **docs** ➔ documentation only
- **style** ➔ formatting, no logic change
- **refactor** ➔ code restructure
- **test** ➔ add/update tests
- **chore** ➔ maintenance tasks

**Example:**

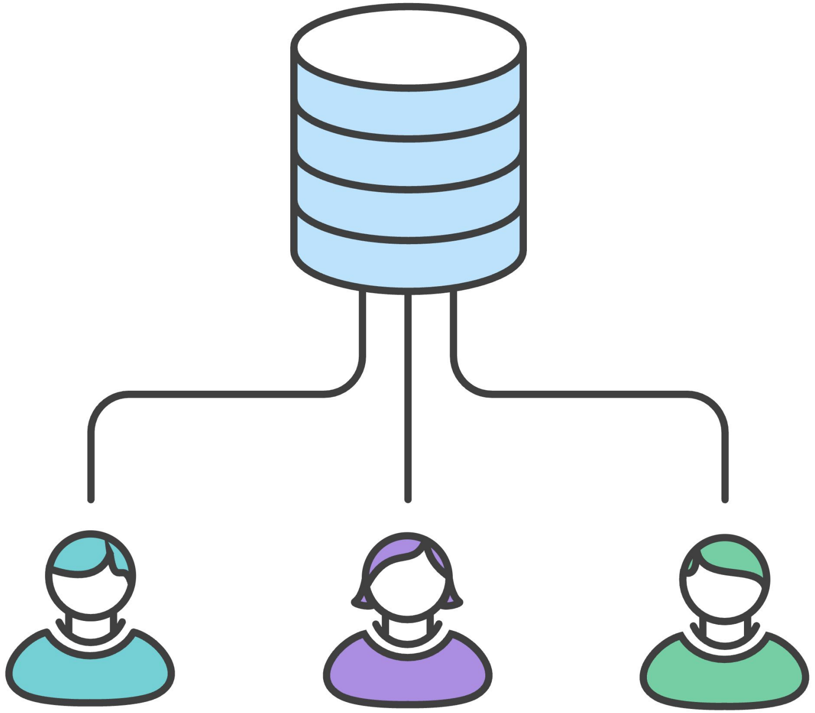
- feat(auth): add user login with JWT
- fix(api): handle null values in response
- docs(readme): update installation guide

## Working Together with Git and GitHub

**Learn how to collaborate effectively using GitHub — fork, branch, pull request, and teamwork workflow.**

**Why It Matters**

- Real-world projects involve multiple developers
- GitHub allows everyone to contribute safely
- Key idea: *“Work separately, merge confidently”*



## Collaboration Workflow Overview

**Typical steps in a team workflow:**

- Clone or fork a repository
- Create a new branch
- Make changes & commit
- Push your branch
- Create a Pull Request (PR)
- Review, discuss, and merge

## Using Branches

- Branch = safe copy of your code
- Use branches for new features or bug fixes
- Example commands:
  - `git checkout -b feature/login-page`
  - `git push origin feature/login-page`

***Tip:*** *Always name branches clearly.*

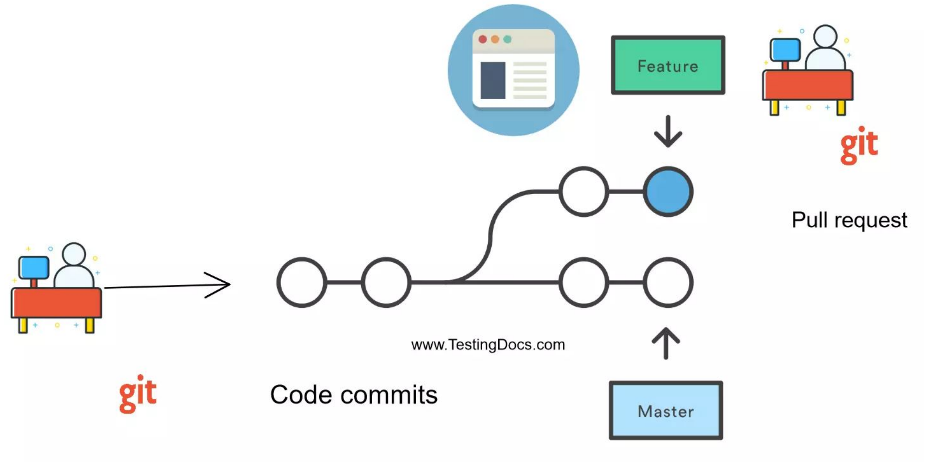
# Fork vs. Clone

## Main Concepts:

| | |
|---|---|
| ● **Working Directory** | → Your actual files |
| ● **Staging Area** | → Prepares changes before saving |
| ● **Repository (Local)** | → Where commits are stored |
| ● **Remote Repository** | → Shared copy on a server **(GitHub, GitLab)** |

## Pull Requests (PR)

- PR = request to merge your changes into main branch
- Used for code review and discussion
- Workflow:
  - Push your branch
  - Open PR on GitHub
  - Reviewer checks and merges it

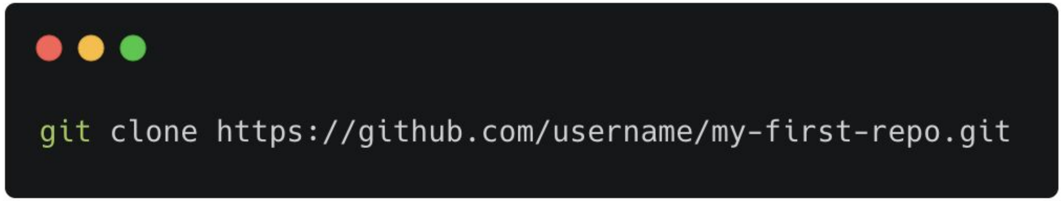


## Resolving Conflicts

- Happens when two people edit the same file
- Git will mark conflicts
- Steps to fix:
  - Pull latest main
  - Edit conflicting files manually
  - Commit and push again

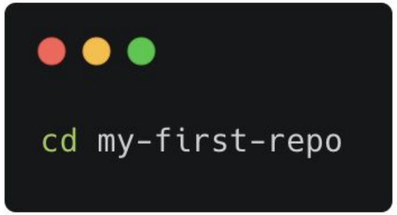
## Clone an Existing Repo

- Copy repo URL from GitHub
- Run:



```
git clone https://github.com/username/my-first-repo.git
```

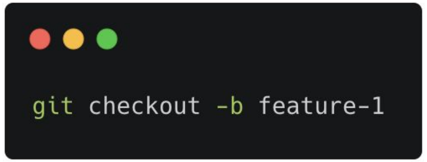
- Enter the project folder:



```
cd my-first-repo
```

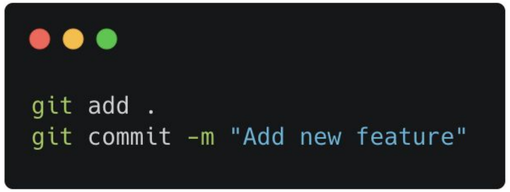
## Branching & Switching

- Create new branch:



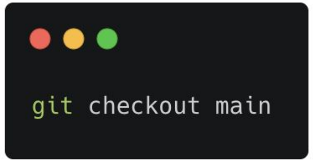
```
git checkout -b feature-1
```

- Make changes in files
- Stage & commit changes:



```
git add .
git commit -m "Add new feature"
```

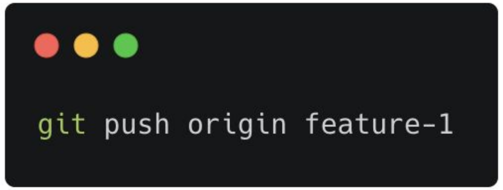
- Switch back to main:



```
git checkout main
```

## Push Branch & Pull Request

- Push branch to GitHub:

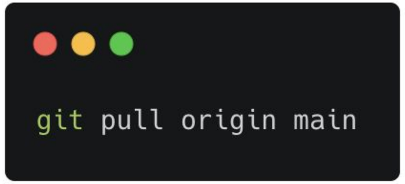


```
git push origin feature-1
```

- On GitHub ➜ Open a **Pull Request**
- Ask teammates to review
- Merge PR into main

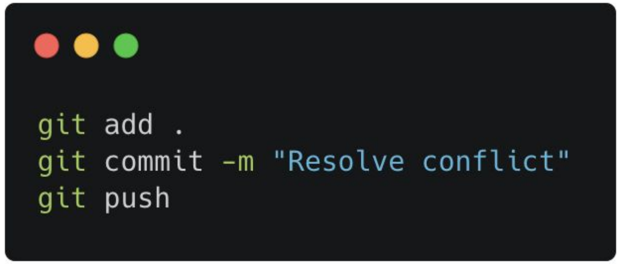
## Resolving Merge Conflicts

- If GitHub says “Conflict”, pull latest main:



```
git pull origin main
```

- Open conflicting file (Git marks conflicts with <<<<<<< >>>>>>>)
- Edit file to keep correct version
- Stage & commit:



```
git add .
git commit -m "Resolve conflict"
git push
```

## Conventional Commit Messages

**Why?**

- Keep commit history clear & consistent
- Make project easier to read, review, and automate (e.g., changelogs)

**Format:**

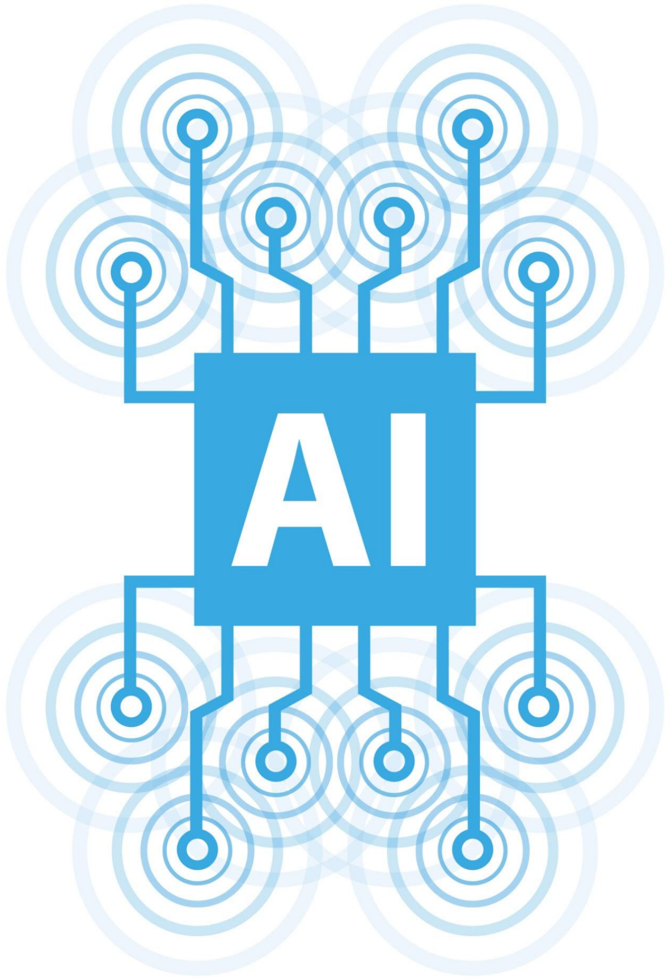
```
<type>(optional scope): <short description>
```

**Common Types:**

- **feat** ➔ new feature
- **fix** ➔ bug fix
- **docs** ➔ documentation only
- **style** ➔ formatting, no logic change
- **refactor** ➔ code restructure
- **test** ➔ add/update tests
- **chore** ➔ maintenance tasks

**Example:**

- feat(auth): add user login with JWT
- fix(api): handle null values in response
- docs(readme): update installation guide



## AI Tools in Software Development

### What is AI in Software Development?

- AI tools are assistants that help write and understand code.
- They use Machine Learning trained on large amounts of code.
- Think of them as a coding buddy:
  - Suggesting solutions
  - Explaining errors
  - Helping you learn faster

## How AI Helps Developer

### Code Completion Tools

- Suggest next lines of code automatically.
- Example: GitHub Copilot, Qodo.

### Debugging Assistants

- Help identify and fix errors.
- Example: ChatGPT, Tabnine.

### Documentation & Learning Helpers

- Explain code in simple words.
- Generate documentation automatically.
- Example: ChatGPT, AI Doc Generators.

### Project Management

- Help track tasks, write commit messages, or review pull requests.
- Example: Linear + AI, GitHub Copilot Chat.

## Popular AI Tools

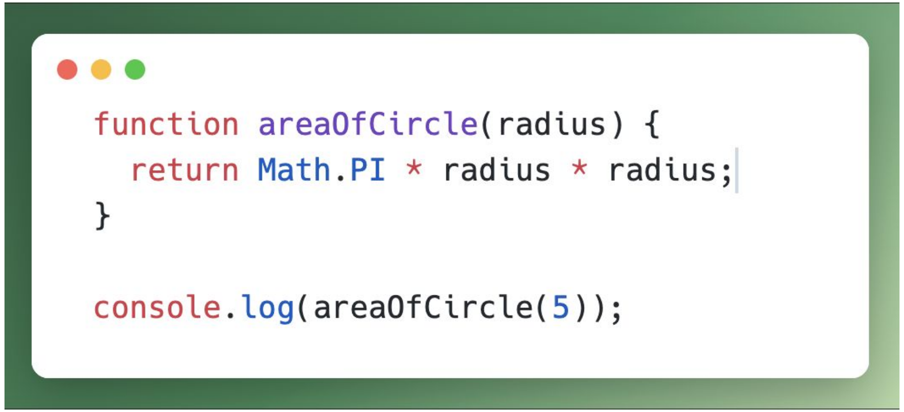
- **ChatGPT / Claude**
  - Ask coding questions in plain English.
  - Example prompt:
    - “Explain how variables work in JavaScript with a simple example.”
  - Can also debug:
    - “Why does this Python code give me an error? [paste code]”
- **Qodo / Github Copilot**
  - AI code completion tool.
  - Works inside IDEs like VS Code.
  - Suggests code as you type.
  - Helps generate functions, tests, and even documentation.

## Example: AI Suggests a Program

**Prompt to AI:**

*"Write a JavaScript program that calculates the area of a circle given the radius"*

**AI Output:**



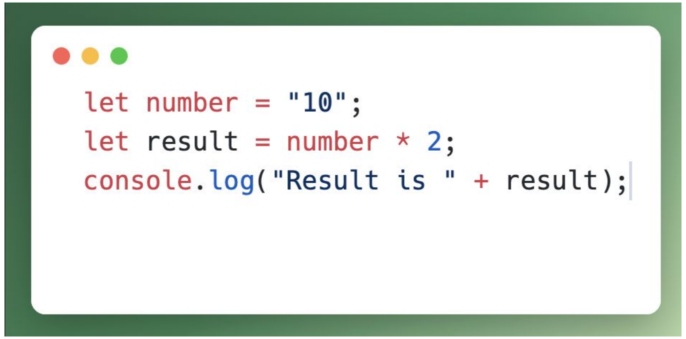
```
function areaOfCircle(radius) {
  return Math.PI * radius * radius;
}

console.log(areaOfCircle(5));
```

**Learning Point:** Students already know variables & data types → AI helps connect those to real problems.

## Example: Debugging with AI

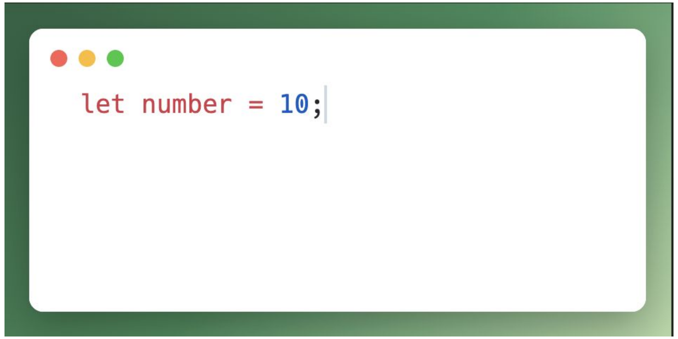
**Student Code:**



```
let number = "10";
let result = number * 2;
console.log("Result is " + result);
```

**AI Explains:**

- "10" is a string, not a number.
- JavaScript auto-converts it, but this may cause bugs.
- Suggestion:



```
let number = 10;
```

## Example: AI Explains Algorithms

**Prompt to AI:**

***"Explain bubble sort like I’m a beginner"***

**AI Answer (simplified):**

- Compare two numbers at a time
- Swap them if out of order
- Repeat until the list is sorted

AI can also generate code for bubble sort.

## Best Practices Using AI

**✅Use AI to:**

- Learn syntax
- Explore examples
- Debug errors
- Speed up writing boilerplate code

**❌Don’t rely only on AI:**

- AI can make mistakes
- You must understand the logic
- Always test the code

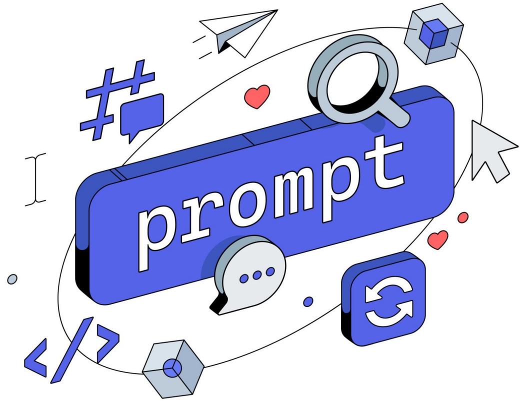
## Limitations of AI Tools

- Sometimes give wrong or insecure code
- May not understand project context fully
- You still need problem-solving skills
- Should be used as a mentor, not replacement

# Mastering Prompting for Developers

**The 5-Step Prompt Format**

- Role / Context
- Goal / Task
- Details / Constraints
- Input / Reference Code
- Output Format



## The 5-Step Prompt Format

### Step 1: Role / Context

- Define AI’s role (senior dev, code reviewer, tester).
- Mention tech stack and environment.
- Example:
    - Act as a senior backend developer using Go and PostgreSQL in Docker.

### Step 2: Goal / Task

- Be direct about what you need.
- Example tasks: generate code, debug, optimize, explain, document.
- Example:
    - Write a REST API endpoint for user login.

## The 5-Step Prompt Format

### Step 3: Details / Constraints

- Add requirements:
  - Libraries/frameworks
  - Coding style (functional, OOP, clean code)
  - Performance/security needs
- Example:
  - Use JWT for authentication. Password must be hashed with bcrypt.

### Step 4: Input / Reference Code

- Provide existing code if applicable.
- Keep it minimal but runnable.
- Helps AI give precise fixes/refactoring.

### Step 5: Output Format

- Specify how you want the answer:
  - Full code block
  - Step-by-step guide
  - Diff only
  - Explanation with comments

## Prompt Templates for Developers

- **Debugging:**
  - Find the bug in this code and explain step by step.
- **Refactoring:**
  - Refactor this code to be more readable using clean code principles.
- **Testing:**
  - Generate Jest unit tests for this function.
- **Documentation:**
  - Write API documentation in Markdown format for this code.

## Best Practice & Common Mistakes

### Best Practice

- Be specific, not vague.
- Break big tasks into smaller prompts.
- Use iterative prompting (refine with follow-ups).
- Provide context (framework, versions, libraries).
- Ask for explanations if you're learning.

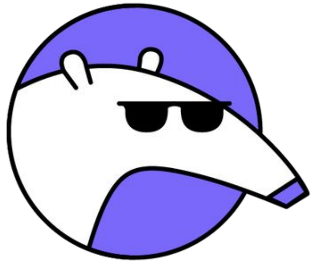
### Common Mistakes

- ❌ Being too vague ➔ bad results.
- ❌ Asking multiple tasks in one prompt.
- ❌ Forgetting to specify language/framework.
- ❌ Not giving input code for debugging.

## Using AI-powered Assistant

**What is Qodo?**

- AI tool that helps you write code faster.
- Works as an extension inside Visual Studio Code.
- Suggests code while you type (autocomplete).
- Supports many languages (JavaScript, Python, Java, etc.).



## Best Practices & Common Use Case

### Best Practices

- Use Qodo as a helper, not a replacement.
- Always review and test AI suggestions.
- Use it to learn new syntax and patterns.
- Combine with ChatGPT for explanations & debugging.

### Common Use Cases

- Writing functions quickly.
- Generating loops and conditions.
- Creating HTML/CSS boilerplates.
- Suggesting SQL queries.

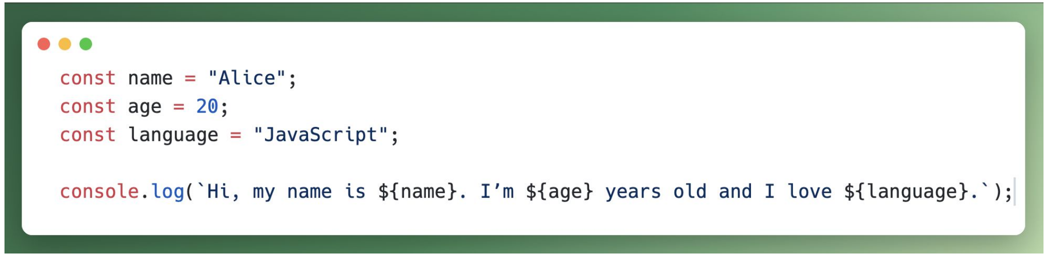
## Exercise

- Install Qodo in VSCode.
- Create a file app.js.
- Type function greet(name) { and see Qodo’s suggestion.
- Accept the suggestion and run it.

# Exercise

Practice JS fundamentals & use AI to improve.

- Create a file intro.js.
- Write a program that:



```
const name = "Alice";
const age = 20;
const language = "JavaScript";

console.log(`Hi, my name is ${name}. I’m ${age} years old and I love ${language}.`);
```

- Use Qodo to:
    - Suggest improvements (e.g., make it interactive using prompt-sync).
    - Add input validation.
- Commit with:
    - feat: add introduction program

## Exercise

- Create a GitHub repo called js-basics.
- Clone it.
- Add a file hello.js that prints "Hello AI!".
- Create a branch feature/greet-user.
- Modify the program to ask for user input (use prompt-sync).
- Push branch and create a Pull Request.
- Use Qodo AI to review the PR and suggest improvements.

## Thank you