Table of contents

1 CLAUDE.md

This file provides guidance to Claude Code (claude.ai/code) when working with code in this repository.

1.1 Project Overview

This is a comprehensive Python tutorial repository using Quarto to generate multiple output formats (HTML book, PDF book, RevealJS slides) in both English and Japanese.

1.2 Development Workflow Commands

1.2.1 Environment Management

- **Primary**: `uv venv` to create virtual environment, `uv add <package>` to add dependencies, `uv run <command>` to execute
- **Alternatives**: Poetry (`poetry install`, `poetry add`), Miniforge (`conda env create`, `conda install`)

1.2.2 Quarto Build Commands

- `quarto render en/` - Build English content (book + slides)
- `quarto render ja/` - Build Japanese content (book + slides)
- `quarto render` - Build entire project (all languages, all formats)
- `quarto preview en/` - Preview English content during development
- `quarto preview ja/` - Preview Japanese content during development

1.2.3 Code Quality Commands

- `uv run ruff check .` - Run linting (replaces black/flake8)
- `uv run ruff format .` - Auto-format code
- `uv run pyright` - Type checking for complex examples
- `uv run pytest` - Run tests (ensure code compiles in Quarto)

1.2.4 Git Workflow Commands

- Always commit completed features immediately after implementation
- Use meaningful commit messages describing the feature/change
- `git add .` then `git commit -m "feat: description"` after each feature
- Push changes regularly to keep repository updated

1.3 Repository Structure

1.3.1 Content Organization

- `en/book/` - English book chapters (.qmd files)
- `en/slides/` - English slide presentations (.qmd files)
- `ja/book/` - Japanese book chapters (.qmd files)
- `ja/slides/` - Japanese slide presentations (.qmd files)
- `shared/code/` - Reusable code examples
- `shared/data/` - Sample datasets
- `shared/images/` - Images and diagrams

1.3.2 Project Tracking

- `plans/active/` - Current implementation plans
- `plans/completed/` - Finished planning documents
- `.states/active/` - Current implementation status
- `.states/completed/` - Completed implementation records

1.3.3 Output

- `docs/` - Generated content for GitHub Pages hosting
- Multiple formats: HTML book, PDF book, RevealJS slides
- Both English and Japanese versions

1.4 Tutorial Content Structure

1. **Environment Setup & Git** - Foundation tools and version control
2. **Python Basics** - Syntax, data types, control flow, functions
3. **Object-Oriented Programming** - Classes, inheritance, advanced OOP
4. **Advanced Topics** - Type hints, async, multiprocessing
5. **Applications** - Data science, automation, general programming, web development

1.5 Development Guidelines

1.5.1 Code Standards

- Target Python 3.12+ features
- Use type hints for complex examples
- Include working code with execution output
- Add exercises and projects at section ends

1.5.2 Content Requirements

- Maintain parallel EN/JP content structure
- Keep shared code examples in `shared/code/`
- Test all code examples compile correctly in Quarto
- Use linear chapter progression within each language

1.5.3 Commit Workflow

- **CRITICAL**: Commit each completed feature immediately after implementation
- Use descriptive commit messages (feat:, fix:, docs:, etc.)
- Don't batch multiple features into single commits
- Push regularly to maintain up-to-date repository

1.5.4 Multi-Language Support

- Keep content synchronized between English and Japanese
- Use consistent chapter numbering and structure
- Share code examples and datasets via `shared/` directory
- Maintain parallel development of both language versions

1.6 Claude Code Tool Usage

1.6.1 Context7 - Library Documentation

- Use `mcp__context7__resolve-library-id` to find Context7-compatible library IDs
- Use `mcp__context7__get-library-docs` to retrieve up-to-date documentation
- Always resolve library ID first before getting docs unless user provides exact ID format
- Prefer Context7 for Python library documentation over web search when available
- Examples: `/python/docs`, `/fastapi/fastapi`, `/requests/requests`

1.6.2 Playwright - Browser Automation

- Use `mcp__playwright__browser_navigate` to visit URLs
- Use `mcp__playwright__browser_snapshot` for accessibility snapshots (preferred over screenshots)
- Use `mcp__playwright__browser_click`, `mcp__playwright__browser_type` for interactions
- Use `mcp__playwright__browser_take_screenshot` for visual captures when needed
- Install browser if needed with `mcp__playwright__browser_install`
- Close browser with `mcp__playwright__browser_close` when done

1.7 VS Code Configuration

- Use ruff for linting and formatting (not black/flake8)
- Configure pyright for type checking
- Install Error Lens extension for debugging
- Set up Quarto extension for .qmd file support