Tutorial Script 1: Introduction & Getting Started with Claude Code

Duration: 8-10 minutes **Objective:** Introduce the course, Claude Code, and demonstrate basic setup with a simple demo

---

PART 1: Course Introduction (2 minutes)

[SCREEN: Show course title slide]

Professor:

"Welcome to this tutorial series on building full-stack applications with AI assistance! In this course, you'll be working with a real-world flashcard application called Flash-Decks.

This is a toy project with a "production-ready" stack: React with TypeScript on the frontend, FastAPI with Python on the backend, and PostgreSQL for the database. You'll learn how to implement features like spaced repetition algorithms, user authentication, and database design.

But here's what makes this course unique: you'll be using Claude Code, an AI-powered development assistant, to help you understand the codebase, plan new features, and implement them efficiently.

By the end of this series, you'll not only understand how to build a complex web application, but you'll also learn how to effectively collaborate with AI tools to accelerate your development workflow."

[SCREEN: Show minimal-flashcards repository overview]

Professor:

"The application we're starting with is a simplified version of a full-featured flashcard app. It has the core functionality working:

- User can create decks of flashcards
- Add basic question-and-answer cards
- Study using a review mode with spaced repetition
- Track their progress

Your job throughout this course will be to add new features to this application, like multiple-choice questions, exam mode, and more. But first, let's get familiar with our AI assistant."

PART 2: Introducing Claude Code (2 minutes)

[SCREEN: Navigate to claude.ai/code]

Professor:

"Claude Code is an AI-powered coding assistant developed by Anthropic. Think of it as an expert developer sitting next to you who has deep knowledge of software engineering best practices, can read and understand your entire codebase, and can help you write code.

What makes Claude Code different from tools like GitHub Copilot?

First, it has full codebase awareness. It can read multiple files, understand the relationships between them, and give you architecture-level insights.

Second, it can execute commands. It can run tests, make git commits, and even search the web for documentation.

Third, it's conversational. You can ask it questions, have it explain code, and iterate on solutions together.

Let me show you how to get started."

PART 3: Setting Up Claude Code (3 minutes)

[SCREEN: Terminal with fresh directory]

Professor:

"First, let's install Claude Code. You'll need Node.js version 18 or higher installed on your machine.

Open your terminal and run:"

```
npm install -g @anthropic-ai/claude-code
```

[PAUSE while command runs]

Professor:

"Once installed, you'll need to authenticate. Run:"

```
claude auth login
```

[SCREEN: Shows browser opening for authentication]

Professor:

"This will open your browser. Sign in with your Anthropic account or create one if you don't have it. Once authenticated, you'll see a success message in your terminal.

Now, let's clone our project repository. For this course, you should have received access to the minimal-flashcards repository."

```

```

```
cd ~/Documents/projects
git clone <repository-url>
cd minimal-flashcards
```

Professor:

"Now we can start Claude Code in this directory:"

```
claude
```

[SCREEN: Shows Claude Code interface launching]

Professor:

"And just like that, we're ready to go! You'll see a chat interface where we can interact with Claude."

PART 4: Quick Demo (3 minutes)

[SCREEN: Claude Code interface]

Professor:

"Let me show you what Claude can do with just a few simple commands. First, let's ask it to understand our project structure."

[TYPE in Claude Code]:

```
What is the overall structure of this flashcard application? Give me a
high-level overview.
```

[SCREEN: Shows Claude's response analyzing the project]

Professor:

"Notice how Claude reads through multiple files and gives us a comprehensive overview. It identified:

- The React frontend with TypeScript
- The FastAPI backend
- The PostgreSQL database
- The key features like deck management and spaced repetition

This is incredibly useful when you're joining a new codebase. Instead of spending hours reading through files, you can ask directed questions.

Let's try something more specific. Let's ask about the database structure."

[TYPE in Claude Code]:

```
Show me the database models and explain the relationships between User,
Deck, and Card.
```

[SCREEN: Shows Claude opening model files and explaining relationships]

Professor:

"See how it opened the specific model files and explained the one-to-many relationships? This is the power of codebase awareness.

Now, let's try something practical. Let's ask Claude to help us set up the development environment."

[TYPE in Claude Code]:

```
Help me set up the development environment. I have Docker installed. Walk
me through the steps.
```

[SCREEN: Shows Claude providing step-by-step setup instructions]

Professor:

"Claude gives us specific commands for:

- Setting up the backend virtual environment
- Starting PostgreSQL with Docker
- Running database migrations
- Installing frontend dependencies
- Starting both servers

These are all specific to OUR codebase, not generic instructions. Let me actually run these commands to show you it works."

[SCREEN: Split screen showing terminal running commands]

```
# Backend setup
cd backend
python -m venv .venv
source .venv/bin/activate  # On Windows: .venv\Scripts\activate
pip install -r requirements.txt
```

Professor:

"While that's installing, let me start the database:"

```
# In another terminal
docker run -d \
```

```
  --name flashdecks-postgres \
  -e POSTGRES_USER=flashdecks \
  -e POSTGRES_PASSWORD=flashdecks \
  -e POSTGRES_DB=flashdecks \
  -p 5432:5432 \
  postgres:15-alpine
```

Professor:

"Now let's create the .env file and run migrations:"

```
# Back in backend directory
cp .env.example .env
alembic upgrade head
```

Professor:

"And start the backend server:"

```
uvicorn app.main:app --reload --port 8000
```

[SCREEN: Shows backend starting successfully with Swagger docs at localhost:8000/docs]

Professor:

"Perfect! Now in a new terminal, let's set up the frontend:"

```
cd frontend
npm install
npm run dev
```

[SCREEN: Shows frontend starting at localhost:5173]

Professor:

"And there we have it! The application is running. Let me open it in the browser."

[SCREEN: Browser showing the landing page of the flashcard app]

Professor:

"Beautiful! We have our flashcard application running. I can click around, see the dashboard, and everything works.

What we just did in about 5 minutes could have taken 30-60 minutes if you were figuring it out from scratch: reading the README, debugging environment issues, figuring out the right Docker commands.

This is the power of Claude Code: it accelerates your workflow by being a knowledgeable assistant that understands YOUR specific codebase."

CLOSING (1 minute)

[SCREEN: Back to Claude Code interface]

Professor:

"In this tutorial, we:

1. Installed and authenticated Claude Code
2. Cloned our project repository
3. Used Claude to understand the codebase structure
4. Set up our entire development environment with Claude's help
5. Got the application running

In the next tutorial, we'll dive deeper into HOW to use Claude effectively. We'll cover:

- How to reference specific files
- Understanding context windows
- Using slash commands for special operations
- The CLAUDE.md file and why it matters
- Best practices for writing good prompts

The key takeaway: Claude Code isn't just about generating code. It's about understanding codebases faster, making fewer mistakes, and learning as you build.

See you in the next tutorial!"

INSTRUCTOR NOTES:

Timing Breakdown:

- Part 1 (Introduction): 2 min
- Part 2 (Claude Code intro): 2 min
- Part 3 (Setup): 3 min
- Part 4 (Demo): 3 min
- Closing: 1 min
- **Total: ~10 minutes**

Key Points to Emphasize:

1. Claude Code is different from autocomplete tools - it understands entire codebases
2. It can execute commands and interact with your development environment
3. The setup process is straightforward
4. Real demo showing actual commands and outputs builds credibility

Preparation Needed:

- Have repository cloned in a separate directory for clean demo
- Test all commands beforehand
- Have Docker running
- Clear terminal history for clean demonstration
- Bookmark localhost:8000/docs and localhost:5173

Common Issues to Address:

- If Docker isn't running: mention students should start Docker Desktop first
- If npm install is slow: mention this is normal for first-time setup
- If any command fails: this is a teaching moment about debugging (which Claude can help with!)