Tutorial Script 5: Claude Code Tips, Tricks, and Advanced Techniques

Duration: 10-12 minutes **Objective:** Share advanced techniques, common pitfalls, debugging strategies, and productivity tips for using Claude Code effectively

---

INTRODUCTION (1 minute)

[SCREEN: Claude Code interface with minimal-flashcards]

Professor:

"Welcome to our final tutorial in this series! Over the past videos, we've learned how to set up Claude Code, use it effectively, understand codebases, and implement features.

Now I want to share the tips and tricks I've learned from using Claude Code extensively - the things that aren't obvious from the documentation but can dramatically improve your productivity.

We'll cover:

1. Advanced prompting techniques
2. How to handle errors and debugging with Claude
3. Using Claude for code review and refactoring
4. Performance tips and context management
5. Common mistakes to avoid
6. Integration with your development workflow

Let's dive in!"

PART 1: Advanced Prompting Techniques (2 minutes)

[SCREEN: Claude Code interface]

Professor:

"Let's start with prompting techniques that go beyond the basics.

Technique 1: Chain of Thought Prompting

Instead of asking Claude to implement something directly, ask it to think through the problem first:"

[TYPE in Claude Code]:

```
I want to add an export feature to export a deck of flashcards to CSV.

Before implementing, let's think through:
1. What data should be included in the CSV?
```

```
2. Should we handle multiple choice and basic cards differently?
3. What edge cases might we encounter (special characters, empty fields)?
4. What's the best approach for the backend (streaming vs buffering)?

Think through each of these, then propose an implementation plan.
```

[SCREEN: Shows Claude analyzing each point before proposing solution]

Professor:

"See the difference? Claude thinks through the problem systematically before proposing a solution. This leads to better, more thoughtful implementations.

Technique 2: Constraint-Based Prompting

Give Claude explicit constraints to work within:"

[TYPE in Claude Code]:

```
Add a search feature for finding cards within a deck.

Constraints:
- Must use PostgreSQL full-text search (we're already using Postgres)
- No additional dependencies
- Response time must be under 200ms for decks with 1000 cards
- Must search both question and answer fields
- Case-insensitive

Given these constraints, what's the best implementation?
```

Professor:

"By providing constraints upfront, you get solutions that fit your requirements instead of generic approaches that might not work.

Technique 3: Role-Based Prompting

Ask Claude to take on a specific role:"

[TYPE in Claude Code]:

```
Act as a senior software architect reviewing this code. Look at
backend/app/services/study.py and identify:
1. Potential performance issues
2. Code that violates SOLID principles
3. Places where error handling could be improved
4. Security concerns

Be critical and thorough.
```

[SCREEN: Shows Claude providing architectural review]

Professor:

"Role-based prompting makes Claude focus on specific aspects. Try roles like:

- 'Security expert' for security audits
- 'Performance engineer' for optimization
- 'QA engineer' for test coverage analysis
- 'Junior developer' for code explanations (simpler language)

Technique 4: Comparative Prompting

Ask Claude to compare multiple approaches:"

[TYPE in Claude Code]:

```
I need to implement real-time notifications when a user completes a study
session.

Compare these approaches:
1. WebSockets
2. Server-Sent Events (SSE)
3. Long polling
4. Simple polling every 30 seconds

For each approach, explain:
- Implementation complexity in our FastAPI + React stack
- Performance implications
- Browser compatibility
- Which you recommend and why
```

Professor:

"This helps you make informed decisions rather than just accepting the first solution Claude suggests."

PART 2: Debugging and Error Handling (2 minutes)

[SCREEN: Terminal showing an error]

Professor:

"Let's talk about debugging with Claude. Eventually, you WILL encounter errors. Here's how to use Claude effectively when things break.

Technique 1: The Error Sandwich

When you get an error, give Claude three pieces of information - I call it the error sandwich:"

[TYPE in Claude Code]:

```
**What I was trying to do:**
I'm trying to create a new multiple choice card through the API.

**What I expected:**
A 200 status with the created card returned.

**What actually happened:**
500 Internal Server Error with this error in the backend logs:
```

sqlalchemy.exc.IntegrityError: (psycopg2.errors.NotNullViolation) null value in column "options" of relation "cards" violates not-null constraint

```
The request body I sent:
{
  "type": "multiple_choice",
  "prompt": "Test question",
  "answer": "Option A",
  "options": ["Option A", "Option B", "Option C", "Option D"]
}

**Help me debug this.**
```

[SCREEN: Shows Claude analyzing the error]

Professor:

"See the structure? Context (what you were doing), expectation (what should happen), reality (what actually happened), plus all relevant details.

Claude can immediately see the problem: the database column `options` doesn't allow nulls, but somewhere in the code path, we're not passing the options correctly.

Technique 2: Incremental Debugging

Break debugging into steps:"

[TYPE in Claude Code]:

```
I'm getting a 500 error when creating a card. Let's debug systematically:

Step 1: Verify the request is reaching the endpoint. Add logging to
routes/decks.py to log the incoming request body.

Step 2: Check if the Pydantic schema validation is passing. Add logging in
schemas/card.py.

Step 3: Check the service layer. Add logging in services/decks.py before
```

```
the database insert.

Give me the code for each logging statement, and we'll narrow down where
the issue is.
```

Professor:

"This systematic approach helps isolate exactly where things go wrong.

Technique 3: Rubber Duck Debugging with Claude

Sometimes just explaining the problem helps you solve it:"

[TYPE in Claude Code]:

```
I'm confused about why this isn't working. Let me explain what I
understand:

1. The Card model has an options field
2. The CardCreate schema includes options
3. I'm sending options in my request
4. But the database is saying options is null

Walk through the code path with me from the API endpoint to the database
INSERT and help me understand where options might be getting lost.
```

[SCREEN: Shows Claude tracing through the code]

Professor:

"Often, explaining the problem to Claude helps YOU spot the issue. And if you don't, Claude's code trace will show you.

Technique 4: Stack Trace Analysis

When you get a complex stack trace, ask Claude to explain it:"

[TYPE in Claude Code]:

```
I got this stack trace. Explain what's happening at each level and
identify the root cause:

[paste full stack trace here]
```

Professor:

"Claude can parse complex stack traces and point you directly to the problematic line."

PART 3: Code Review and Refactoring (2 minutes)

[SCREEN: Claude Code interface with a code file open]

Professor:

"Claude is excellent for code review and refactoring. Let me show you how to use it effectively.

Technique 1: Pre-Commit Code Review

Before committing, ask Claude to review your changes:"

[TYPE in Claude Code]:

```
I'm about to commit changes to backend/app/services/study.py. Review my
changes and check for:

1. Breaking changes to existing functionality
2. Missing error handling
3. Potential race conditions
4. Code that's hard to test
5. Missing documentation
6. Performance issues

Here are my changes:
[paste git diff or describe changes]
```

[SCREEN: Shows Claude reviewing]

Professor:

"Claude acts as a second pair of eyes, catching issues before they get committed.

Technique 2: Refactoring for Maintainability

Ask Claude to suggest refactoring opportunities:"

[TYPE in Claude Code]:

```
Look at backend/app/services/study.py. This file has grown to 400 lines.

Suggest how to refactor it for better maintainability:
1. What functions could be extracted?
2. What's the right level of abstraction?
3. How would you organize the extracted code?
4. What would you name the new modules?

Provide specific examples.
```

[SCREEN: Shows Claude analyzing and suggesting refactoring]

Professor:

"Claude can spot code smells and suggest patterns like:

- Extracting helper functions
- Creating service classes
- Separating concerns
- Applying design patterns

Technique 3: Test Coverage Analysis

Ask Claude to identify untested code:"

[TYPE in Claude Code]:

```
Look at backend/app/services/study.py and the corresponding tests in
tests/test_api_study.py.

Identify:
1. Functions that have no tests
2. Edge cases that aren't tested
3. Error conditions that aren't tested
4. Suggest specific test cases to add

Prioritize by risk level.
```

Professor:

"This helps you achieve better test coverage systematically.

Technique 4: Documentation Generation

Ask Claude to generate or improve documentation:"

[TYPE in Claude Code]:

```
Generate comprehensive docstrings for all functions in
backend/app/services/study.py.

Use Google-style docstrings with:
- Brief description
- Args with types
- Returns with type
- Raises (for exceptions)
- Examples (where helpful)
```

[SCREEN: Shows Claude generating docstrings]

Professor:

"Claude generates consistent, thorough documentation that you can review and adjust."

PART 4: Context Management and Performance (2 minutes)

[SCREEN: Claude Code interface]

Professor:

"Let's talk about managing Claude's context effectively for better performance.

Tip 1: Use /clear Strategically

When switching between unrelated tasks, clear the context:"

[TYPE in Claude Code]:

```
/clear
```

Professor:

"This prevents Claude from being confused by irrelevant previous context. Use it when:

- Switching from frontend to backend work
- Moving to a completely different feature
- Claude seems confused by previous context

Tip 2: Load Relevant Files Upfront

For complex tasks, explicitly load files at the start:"

[TYPE in Claude Code]:

```
I'm going to work on optimizing the study session queries. Load these
files:
- backend/app/services/study.py
- backend/app/models/study.py
- backend/app/models/card.py
- backend/app/api/routes/study.py

Tell me when you're ready and you understand the current implementation.
```

Professor:

"This ensures Claude has all necessary context before you start asking questions.

Tip 3: Reference Previous Responses

Claude remembers the conversation. Reference earlier discussions:"

[TYPE in Claude Code]:

```
Earlier in this conversation, you suggested extracting the SRS algorithm
into a separate utility. Let's implement that now. Start with the plan you
outlined.
```

Professor:

"This maintains continuity and saves you from re-explaining.

Tip 4: Break Large Changes into Conversations

For very large features, don't try to do everything in one conversation:"

[SCREEN: Show example of breaking up work]

Professor:

"Instead of:

- 'Implement entire user authentication system'

Do:

- Conversation 1: Plan authentication architecture
- Conversation 2: Implement user model and database
- Conversation 3: Implement JWT token handling
- Conversation 4: Implement login/signup endpoints
- Conversation 5: Implement frontend auth

Each conversation stays focused, and you can test between them.

Tip 5: Save Important Prompts

Keep a personal 'prompt library' of effective prompts:"

[SCREEN: Show a text file with saved prompts]

```
# My Claude Code Prompt Library

## Code Review Template
```

Review [file path] for:

1. Breaking changes
2. Error handling
3. Performance issues
4. Security concerns
5. Test coverage

```
## Implementation Plan Template
```

I want to implement [feature].

Requirements:

- [requirement 1]
- [requirement 2]

Constraints:

- [constraint 1]
- [constraint 2]

Create a step-by-step implementation plan.

```
## Bug Investigation Template
```

What I was trying to do: [context] What I expected: [expected behavior] What happened: [actual behavior] Error message: [error] Relevant code: [file paths]

Help me debug this.

```

```

Professor:

"Build your own library of prompts that work well for you."

PART 5: Common Mistakes to Avoid (2 minutes)

[SCREEN: Claude Code interface]

Professor:

"Let's talk about common mistakes I see students make with Claude Code.

Mistake 1: Blindly Accepting All Suggestions

Bad practice:"

[Example on screen]:

```
User: "Add caching to this endpoint"
Claude: [Provides implementation]
```

```
User: [Copies and pastes without reading]
```

Professor:

"Always read and understand Claude's code. Ask follow-up questions:

- Why did you choose this approach?
- What are the trade-offs?
- Are there any potential issues?

Mistake 2: Not Testing Incrementally

Bad practice: Implementing an entire feature before testing anything.

Good practice:"

[TYPE in Claude Code]:

```
We just implemented the backend for multiple choice cards. Before moving
to the frontend, let's test the backend thoroughly. Give me:
1. A curl command to create a multiple choice card
2. A curl command to test validation (intentionally invalid data)
3. A curl command to retrieve the card

I want to verify the backend works before touching frontend code.
```

Professor:

"Test after each logical unit of work.

Mistake 3: Vague Error Reports

Bad:"

```
User: "It doesn't work"
```

Professor:

"Good:"

```
User: "When I POST to /api/v1/decks with this JSON body [...], I get a 422
error with message 'field required: description'. Looking at the
DeckCreate schema in schemas/deck.py, is description required or
optional?"
```

Professor:

"Be specific. Include error messages, request data, and what you've already investigated.

Mistake 4: Ignoring CLAUDE.md

Students often forget to update CLAUDE.md as the project evolves.

Good practice: After implementing a feature, update CLAUDE.md:"

[TYPE in Claude Code]:

```
We just implemented multiple choice cards. Update CLAUDE.md to document:
1. The new card type
2. How to create multiple choice cards
3. Any new validation rules
4. Testing considerations

This helps future Claude instances (and other developers) understand the
feature.
```

Professor:

"Keep CLAUDE.md current. It's your project's living documentation.

Mistake 5: Over-Relying on Claude

Claude is a tool, not a replacement for understanding. If you find yourself:

- Not understanding the code Claude writes
- Unable to debug without Claude
- Not learning the underlying concepts

Stop and learn the fundamentals. Use Claude to ACCELERATE learning, not replace it."

PART 6: Workflow Integration (2 minutes)

[SCREEN: Terminal and IDE side by side]

Professor:

"Finally, let's talk about integrating Claude into your development workflow.

Workflow Pattern 1: The Planning Session

Start each feature with a planning session:"

```
1. Open Claude Code
2. Explain the feature you want to build
3. Ask Claude for implementation plan
4. Review and refine the plan
5. Save the plan in a comment or TODO file
```

```
6. Close Claude
7. Implement following the plan, reopening Claude for specific questions
```

Professor:

"This prevents you from becoming dependent on Claude being open constantly.

Workflow Pattern 2: The Code Review Buddy

Before committing:"

```
1. Make your changes
2. Run tests locally
3. Open Claude Code
4. Ask for code review
5. Address any issues Claude finds
6. Commit
```

Professor:

"Claude becomes your pre-commit reviewer.

Workflow Pattern 3: The Debugging Partner

When stuck on a bug:"

```
1. Try to debug yourself for 10-15 minutes
2. If stuck, gather information:
   - What you tried
   - Error messages
   - Relevant code
3. Open Claude with structured debugging request
4. Follow Claude's suggestions systematically
5. Explain the solution back to Claude (solidifies learning)
```

Professor:

"This balances independence with getting help when needed.

Integration with Git

Claude can help with git workflows:"

[TYPE in Claude Code]:

```
I need to create a pull request for the multiple choice feature. Help me:
1. Generate a good commit message for my changes
```

```
2. Write a PR description that explains what changed and why
3. Suggest what reviewers should pay attention to
```

[SCREEN: Shows Claude generating PR description]

Professor:

"Claude can generate professional, detailed PR descriptions.

Integration with Testing

Use Claude to help write tests faster:"

[TYPE in Claude Code]:

```
I just implemented the multiple choice card creation endpoint. Generate
comprehensive pytest test cases covering:
1. Happy path (valid multiple choice card)
2. Validation errors (wrong number of options, answer not in options)
3. Edge cases (empty options, special characters)
4. Integration with existing deck

Use our existing test patterns from tests/test_api_decks.py as a
reference.
```

Professor:

"Claude generates tests that match your existing patterns.

Keyboard Shortcuts and Efficiency

Learn to use Claude Code efficiently:

- Use @ for quick file references
- Keep common prompts in a text file for copy-paste
- Use /clear liberally to reset context
- Learn keyboard shortcuts in your terminal

The 80/20 Rule

80% of your time should be:

- Reading code
- Understanding architecture
- Designing solutions
- Testing

20% of your time should be:

- Asking Claude questions
- Getting code generated

- Getting reviews

If those ratios are flipped, you're over-relying on Claude."

CLOSING (1 minute)

[SCREEN: Summary slide]

Professor:

"Let's wrap up with the key takeaways from this entire tutorial series:

Advanced Techniques:

- Chain of thought prompting for complex problems
- Constraint-based prompting for specific requirements
- Role-based prompting for focused analysis
- Comparative prompting for decision making

Debugging:

- Use the 'error sandwich' (context, expectation, reality)
- Debug incrementally and systematically
- Analyze stack traces with Claude's help
- Explain problems to clarify your thinking

Code Quality:

- Pre-commit reviews with Claude
- Refactoring suggestions
- Test coverage analysis
- Documentation generation

Efficiency:

- Manage context strategically
- Reference previous responses
- Break large tasks into conversations
- Build your prompt library

Common Mistakes to Avoid:

- Don't blindly accept suggestions
- Test incrementally
- Be specific with errors
- Keep CLAUDE.md updated
- Don't over-rely on Claude

Remember: Claude Code is a powerful tool that can 10x your productivity, but only if you:

1. Understand what you're building

2. Ask good questions
3. Verify the results
4. Learn from the process

You've now completed this tutorial series. You know how to:

- Set up and use Claude Code
- Understand new codebases quickly
- Plan and implement features effectively
- Debug and optimize code
- Integrate Claude into your workflow

Now go build something amazing! Use Claude Code to accelerate your learning and productivity, but never forget that YOU are the developer. Claude is your tool, your assistant, your force multiplier.

Good luck, and happy coding!"

INSTRUCTOR NOTES:

Timing Breakdown:

- Introduction: 1 min
- Part 1 (Advanced Prompting): 2 min
- Part 2 (Debugging): 2 min
- Part 3 (Code Review): 2 min
- Part 4 (Context Management): 2 min
- Part 5 (Common Mistakes): 2 min
- Part 6 (Workflow Integration): 2 min
- Closing: 1 min
- **Total: ~12 minutes**

Key Points to Emphasize:

1. Advanced techniques build on basics
2. Debugging is systematic, not random
3. Code quality matters - use Claude to improve it
4. Context management improves performance
5. Avoid common pitfalls
6. Integrate Claude into existing workflow

Preparation Needed:

- Prepare examples of good vs. bad prompts
- Have error examples ready
- Create a sample prompt library file
- Prepare workflow diagrams
- Have git/PR examples ready

Demonstrations:

- Show actual debugging session
- Show code review in action
- Show prompt library in practice
- Show PR generation
- Show test generation

Take-Home Message: Claude Code is powerful, but it's a tool that enhances developer skills, not replaces them. The goal is to become a better developer faster, not to avoid learning development.

Resources to Mention:

- Claude Code documentation: claude.ai/code
- Official GitHub: github.com/anthropics/claude-code
- Community forums for questions
- Keep learning fundamentals alongside using Claude

Common Student Questions:

- Q: "Is using Claude cheating?" A: No more than using Stack Overflow or documentation. It's a tool. Understanding the code you use is what matters.
- Q: "Will Claude make developers obsolete?" A: No. Claude helps with implementation, but YOU make the decisions about what to build and how to architect it.
- Q: "How much should I rely on Claude?" A: Use the 80/20 rule. 80% thinking and learning, 20% getting assistance.

Final Encouragement: Remind students that everyone makes mistakes learning to use AI tools effectively. It's a new skill that takes practice. Encourage experimentation and learning from failures.