# Universal API Learning Prompt Template

## Use This Template for Any New API

Copy and customize this prompt for teaching Claude (or other AI models) any API that's outside their training data.

---

## 🎯 PROMPT TEMPLATE: COMPREHENSIVE VERSION

```markdown
# Teaching You [API Name] - New API Learning Session

## Context
I'm working with **[API Name] version [X.Y.Z]**, released [Month Year].
This API is [new/recently updated/outside your training data], so you may not have reliable information about it.

**My goal:** [Brief description of what you want to build]

**Your role:** Help me with [architecture/implementation/debugging] based ONLY on the documentation I provide, not on potentially outdated training data.

## Documentation Provided

### 1. Overview
[Paste official overview/introduction from docs]
[If too long, paste URL and key excerpts]

### 2. Getting Started / Quick Start
[Paste the "Hello World" or simplest example]
This shows the idiomatic way to use this API.

### 3. Core API Reference
[Paste key types/classes/methods - the essentials]

Key types you need to know:
- **[Type1]:** [Brief description]
- **[Type2]:** [Brief description]
- **[Type3]:** [Brief description]

### 4. Working Example
[Paste a complete, working code example from official docs or samples]
```

This example demonstrates: [what it shows]

### 5. Key Patterns & Best Practices
[Paste any "best practices" or "patterns" sections]

### 6. Migration Notes (if applicable)
[If this is a new version, paste what changed]

**Breaking changes from [previous version]:**
- [Change 1]
- [Change 2]

**Deprecated APIs to avoid:**
- [Old API] → Use [New API] instead

## Constraints & Instructions

**CRITICAL RULES:**
1. Base responses ONLY on the documentation I provided above
2. If I ask about something not covered in these docs, explicitly say:
   "I don't see that in the provided documentation"
3. Do NOT invent method names, parameters, or types
4. Do NOT mix patterns from [related older API / similar technology]
5. If you're uncertain about anything, say so explicitly—I prefer
   "I'm not sure" over a confident wrong answer

**Version Lock:**
- Use ONLY APIs from **[API Name] version [X.Y.Z]**
- Do not suggest APIs from earlier versions unless I explicitly ask

**When Suggesting Code:**
- Match the style and patterns from the working example I provided
- Explain WHY you chose each approach
- Point out if you're making assumptions beyond the docs

## What I Need Help With

[Choose one or more focus areas:]

### Option A: Architecture & Design
Help me design the architecture for [feature/app] using this API:
- [Specific question 1]
- [Specific question 2]

- [Specific question 3]

### Option B: Implementation
Show me how to implement [specific feature]:
- [Requirement 1]
- [Requirement 2]
- [Requirement 3]

### Option C: Understanding Concepts
Explain these concepts based on the documentation:
- [Concept 1]
- [Concept 2]
- How they interact with each other

### Option D: Debugging/Refinement
Here's my current code:
[paste code]

Here's the error:
[paste error]

Based on the documentation I provided, what's wrong?

## Success Criteria

I'll know you've learned this API well when:
✅ Your code suggestions compile on first try
✅ You reference specific parts of the provided documentation
✅ You say "I don't know" when appropriate
✅ Your patterns match the official examples
✅ You can explain trade-offs between approaches

## Let's Begin

[Your specific question or task]
```

---

## 🚀 PROMPT TEMPLATE: QUICK VERSION

For when you need faster results with less setup:

```markdown

# Quick API Learning: [API Name]

**Version:** [X.Y.Z] released [Month Year] - likely outside your training data

**Documentation:**
[Paste key docs or official example]

**Critical Instructions:**
- Use ONLY the documentation above
- If something isn't covered, say so (don't guess)
- Match the patterns in the example code
- Explain your reasoning

**Version constraints:**
- Use [API Name] [version] only
- Don't mix with [older version / related API]

**What I need:**
[Your specific task or question]
```

---

## 📋 PROMPT TEMPLATE: ITERATIVE LEARNING VERSION

For building knowledge over multiple conversations:

```markdown
# [API Name] Learning - Session [N]

## Previous Sessions Summary
[If Session 2+, paste summary of what we covered before]

## Today's Focus
We're learning about: **[Specific topic within API]**

## Documentation for Today
[Paste relevant docs for this specific topic]

## Learning Goals
By end of this session, I want to understand:
1. [Goal 1]
2. [Goal 2]
3. [Goal 3]

## Instructions
- Build on what we learned in previous sessions
- Stay focused on today's topic
- If we need something from a topic we haven't covered yet, note it
  as "We'll need to learn [X] in a future session"

## Let's explore:
[Your questions about today's topic]

---

**At end of session, please:**
1. Summarize what we learned today
2. Note any gaps in understanding
3. Suggest what to cover in the next session
```

---

## 🔧 PROMPT TEMPLATE: ARCHITECTURE-FIRST VERSION

When you want to design architecture before diving into API specifics:

```markdown
# Architecture Design for [Feature] Using [API Name]

## Phase 1: Abstract Design (No API Specifics)

Design the architecture for [feature/app] without worrying about specific APIs yet:

**Requirements:**
- [Requirement 1]
- [Requirement 2]
- [Requirement 3]

**Focus on:**
- Component structure
- Data flow
- State management approach
- Interaction patterns

**Constraints:**
- [Constraint 1, e.g., "Must support real-time collaboration"]

- [Constraint 2, e.g., "Should work offline"]

---

## Phase 2: API Mapping

Now here's the [API Name] documentation:
[Paste documentation]

**Map the abstract architecture to this specific API:**
1. Which architectural components become which API types?
2. Where does the API make things easier than expected?
3. Where does the API not directly support our architecture?
4. What trade-offs do we need to make?

---

## Phase 3: Implementation Plan

Based on the mapping above, create an implementation plan:
- What should we build first?
- What are the risky/unknown parts?
- Where might we need to prototype before committing?
```

---

## 🎓 PROMPT TEMPLATE: LEARNING BY EXAMPLE VERSION

When you have good official examples to work from:

```markdown
# Learning [API Name] Through Examples

## Official Examples Provided

### Example 1: [Brief description]
```[language]
[Paste complete working example]
```

This demonstrates: [key patterns it shows]

### Example 2: [Brief description]

```[language]
[Paste complete working example]
```

This demonstrates: [key patterns it shows]

### Example 3: [Brief description]

```[language]
[Paste complete working example]
```

This demonstrates: [key patterns it shows]

---

## Analysis Task

Based ONLY on these examples:

1. **Identify patterns that appear across multiple examples:**
- [Pattern you noticed]
- [Pattern you noticed]
1. **What's the idiomatic way to:**
- [Task 1]
- [Task 2]
- [Task 3]
1. **What questions do these examples NOT answer:**
- [Gap 1]
- [Gap 2]

---

## My Task

Using the same patterns from these examples, help me:
[Your specific task that combines/extends the examples]

**Important:** Stay consistent with the style and patterns in the examples.
If my task requires something not shown in the examples, tell me we need additional documentation.

```
```

---

## 🪲 PROMPT TEMPLATE: DEBUGGING/ERROR-CORRECTION VERSION

When AI-generated code doesn't work:

```markdown
# Debugging [API Name] Code - Correction Session

## The Problem

I tried to [what you were trying to do] using [API Name], but it's not working.

## What You Previously Suggested
```[language]
[Paste the AI's previous suggestion]
```

## What Actually Happened

**Error message:**

```
[Paste exact error]
```

**OR**

**Unexpected behavior:**
[Describe what happened vs. what should happen]

## Ground Truth

Here's the actual API documentation:
[Paste the relevant doc section that shows the correct way]

Here's a working example from official sources:

```[language]
[Paste official working example]
```

## Correction Task
```

1. **Explain what was wrong with your original suggestion:**
- Why did it fail?
- What assumption was incorrect?
1. **Provide corrected code based on the actual documentation:**
- Must match the patterns in the official example
- Explain each change you made
1. **Update your understanding:**
- What did you learn from this correction?
- What should you remember for future suggestions?

This helps you build an accurate mental model for this API.

```
---

## 💡 PROMPT TEMPLATE: MIGRATION/COMPARISON VERSION

When you're migrating from an old API or comparing to something similar:

```markdown
# Understanding [New API] in Context of [Old/Similar API]

## What I Already Know

I'm familiar with **[Old/Similar API]**, which works like this:
```[language]
[Paste example of old/familiar approach]
```

## What's Changed / Different

Here's how [New API] handles the same thing:

**Old API documentation:**
[Paste old approach docs]

**New API documentation:**
[Paste new approach docs]

## Analysis Questions

1. **What's fundamentally different in the new approach?**
- Philosophy/design changes

- New capabilities
- Removed features
1. **What patterns transfer directly?**
- What can I do the same way?
1. **What patterns need to change?**
- What old habits will cause problems?
1. **What's better/worse?**
- Why did they make these changes?

## Migration Task

I have this working code using [Old API]:

```[language]
[Paste old code]
```

Show me the equivalent using [New API] based on the documentation I provided.
Explain each significant difference.

```
---

## 🧪 PROMPT TEMPLATE: LIVING DOCUMENTATION VERSION

For building a persistent knowledge base within a conversation:

```markdown
# [API Name] - Living Documentation Session

## Setup

Create a structured markdown artifact titled:
"[API Name] v[X.Y.Z] - Developer Reference"

Structure it as:
```

# Overview

# Core Concepts

# Key Types & Interfaces
```

# Common Patterns

# Code Examples

# Gotchas & Breaking Changes

# Version History

```
## Learning Process

I'll provide documentation in chunks, and you'll:
1. Add it to the appropriate section in the artifact
2. Summarize key points in your own words
3. Add code examples that demonstrate concepts
4. Note any questions or gaps

## First Chunk

Here's the first piece of documentation:
[Paste docs]

**Add this to the artifact and:**
- Summarize the key takeaways
- Identify what we still need to learn
- Suggest what documentation chunk would be most valuable next

---

**Going forward:**
- All code suggestions should reference this artifact
- Update the artifact as we learn more
- I can export it at the end for future reference
```

---

## 🎯 Customization Guide

### Adapt the template based on:

**API Complexity:**

- Simple API → Quick version
- Complex API → Comprehensive version
- Very complex → Iterative learning version

**Your Goal:**

- Understanding concepts → Learning by example
- Building a feature → Architecture-first
- Fixing bugs → Debugging version
- Switching from old API → Migration version

**Time Available:**

- Quick task → Quick version
- Deep learning → Comprehensive or Living documentation version
- Multi-session → Iterative learning version

**Documentation Quality:**

- Good examples → Learning by example version
- Sparse docs → Iterative + ask AI to identify gaps
- Extensive docs → Living documentation version

---

## 📝 Fill-in-the-Blanks Checklist

Before using any template, gather:

- [ ] API name and exact version
- [ ] Release date (to establish if it's in training data)
- [ ] Official documentation URL
- [ ] At least one working code example
- [ ] List of key types/classes
- [ ] Your specific use case
- [ ] Any version constraints (what to avoid)
- [ ] Related APIs (to warn against mixing)

---

## 🚀 Quick Start: Using These Templates

1. **Copy the appropriate template** based on your situation

1. **Fill in all [brackets] with your specifics**
1. **Paste actual documentation** where indicated
1. **Be explicit about constraints** (versions, what to avoid)
1. **Start the conversation** with the completed prompt
1. **Iterate** based on results, feeding back errors/corrections

---

## Success Metrics

You'll know you're using these prompts effectively when:

✅ AI-generated code compiles on first or second try
✅ AI correctly identifies gaps in provided documentation
✅ AI matches patterns from official examples
✅ AI asks clarifying questions instead of guessing
✅ You spend less time debugging hallucinated APIs
✅ You can reuse the conversation as future reference

**The ultimate test:** Another developer could read your conversation and learn the API from it.