

ロジック実装は
最高に楽しいのに...

境界値テスト...
異常系の網羅...
面倒くさい...!

スパゲッティコード...
どこからテスト
付けばいいんだ...

Claude Codeで
テストコード作成を自動化する
ターミナルから始める次世代の開発スタイル

モチベーションが...
下がる...

先輩、テストコード
書くのに
あと数時間
かかりそうです…

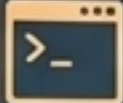
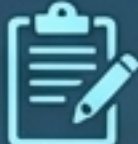
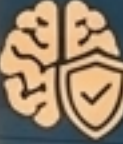
まだ自分で全部
書いているのか？
今日から
『Claude Code』を
使え。

ブラウザの
AIチャットですか？
かりますか？
コピペが面倒で…

違う。
お前の代わりに
『手を動かす』
自律型エージェントだ。

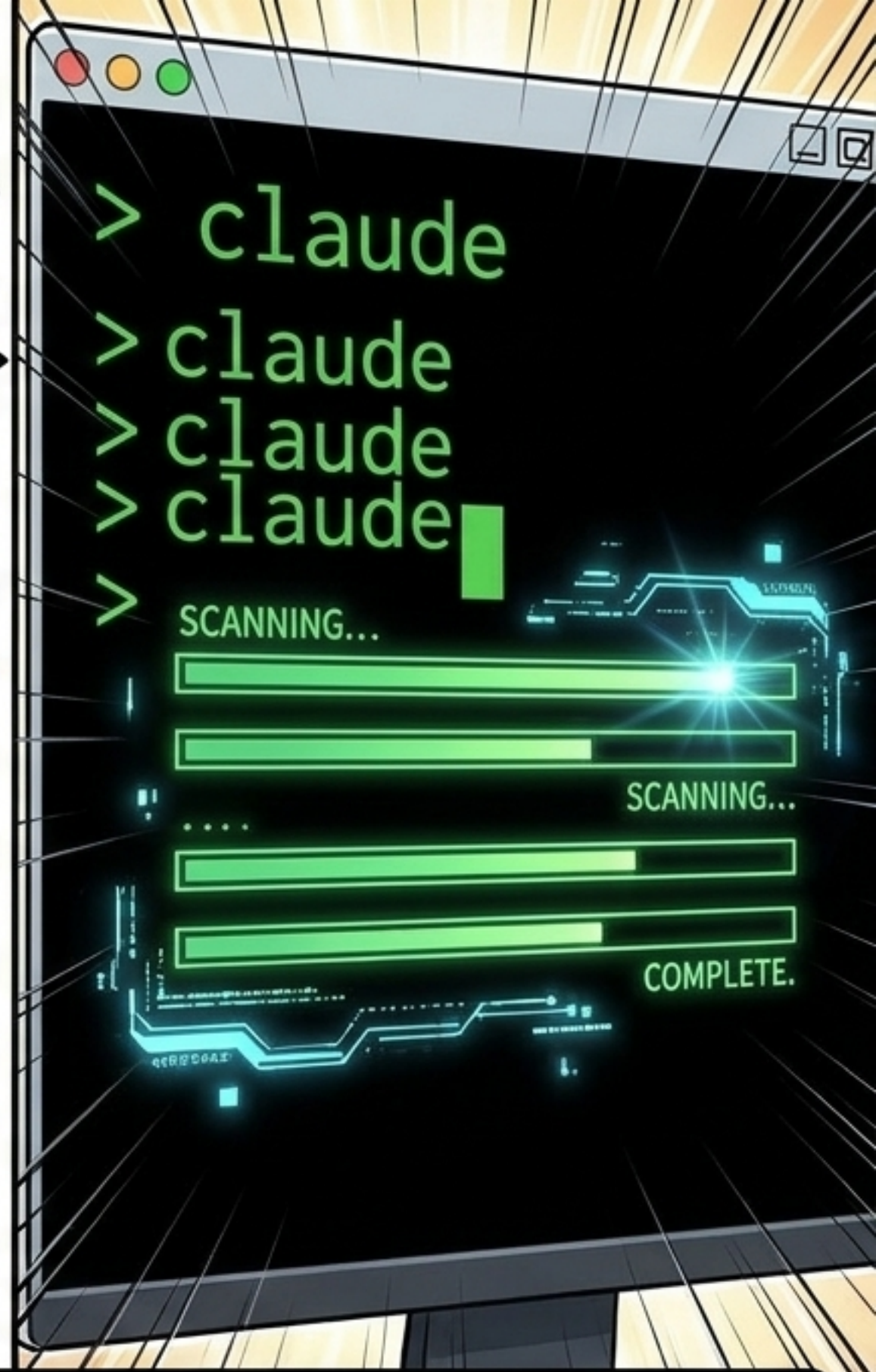
```
> claude
```




	従来のチャットAI	Claude Code (Agentic AI)
【動作場所】	 ブラウザ	 ターミナル常駐
【コンテキスト】	 手動でコードをコピペ	 プロジェクト全ファイルを自動把握
【実行権限】	 コード片の提案のみ	 実際にファイルを作成・修正
【テスト・デバッグ】	 人間が実行	 自律的にコマンドを叩き自己検証



テストコード作成は『耐えるべき苦行』から『AIエージェントと共に品質を爆上げるクリエイティブな自動化プロセス』へ進化する。



対象関数を指定するだけ。認知負荷ゼロのテスト生成体験。

Claude Academy提唱「4Dフレームワーク」の極意

✖ 悪い例：
「テストを書いて」



○ 良い例：
「【対象】src/utils/math.pyの
calculate関数のテストを、
【ツール】Pytestを使って、
【観点】境界値に注目して書いて」



直接指示 (Direct Instruction)



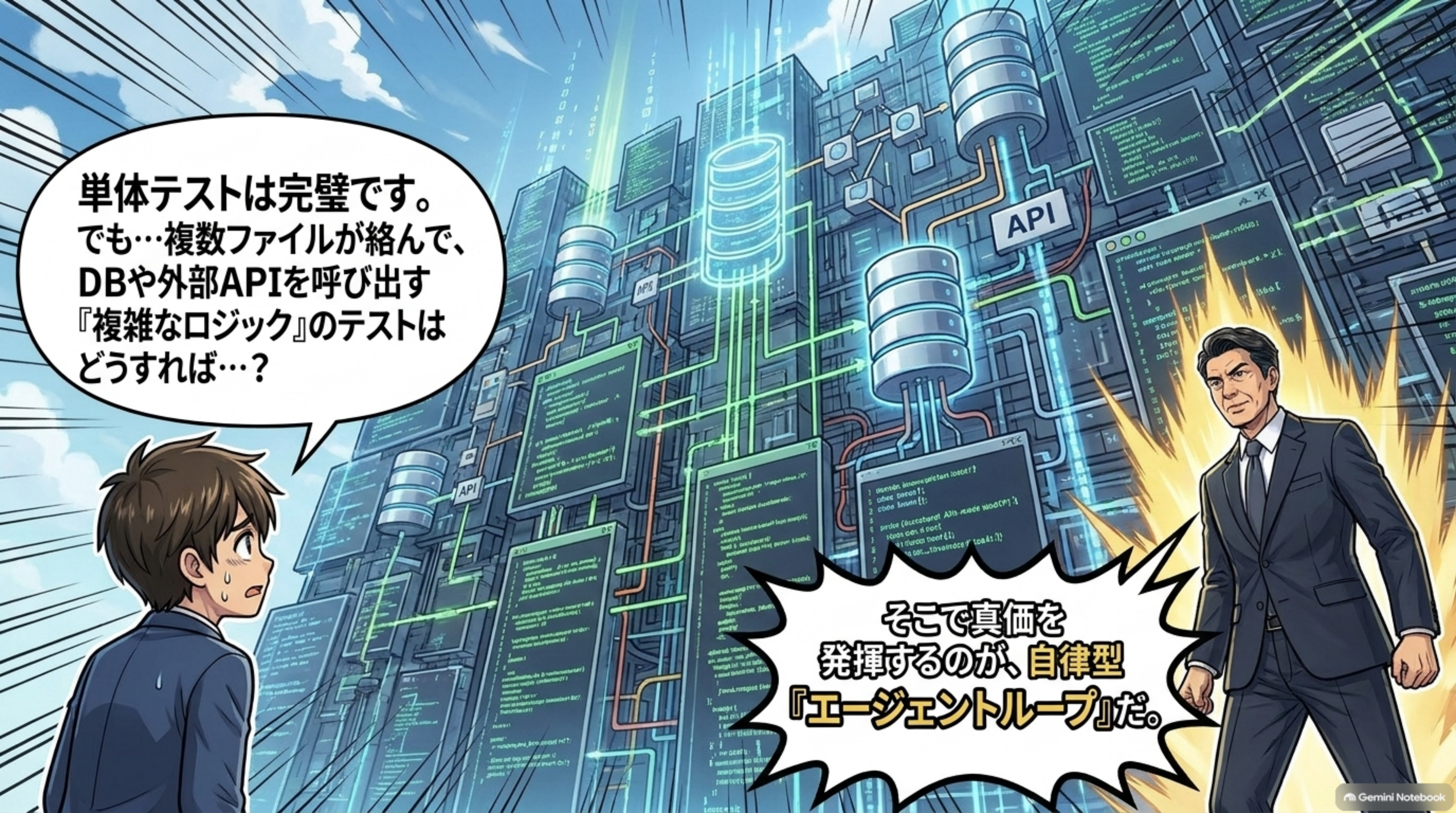
記述力 (Description) を高めよ。
具体的・直接的な指示が
AIの迷いをなくす。

先輩の虎の巻：ファイル探索のコツ

現象	原因	対処法
「ファイルが見つかりません」	相対パスの誤解	プロジェクトルートからのフルパス (例: <code>src/utils/math.py</code>) で指定する。
全く関係ないファイルを見る	曖昧なファイル名	拡張子 (<code>.py</code> , <code>.js</code> 等) まで含めて正確に指定する。
古いロジックでテストを書く	コンテキストの断絶	一度 <code>/clear</code> でコンテキストをリセットする。

ファイルが見つかりません…





単体テストは完璧です。
でも…複数ファイルが絡んで、
DBや外部APIを呼び出す
『複雑なロジック』のテストは
どうすれば…？

そこで真価を
発揮するのが、自律型
『エージェントループ』だ。

Commit(承認)

変更内容を要約し、人間に承認(Approval)を求める

Code(実装と自己検証)

コード生成後、自分でテストコマンド(pytest等)を叩いてエラーが出れば自律的にデバッグ

Explore(探索)

lsやgrepを駆使し、自律的にソースコードを読み込み構成を把握

Plan(計画)

実行ステップ(新規作成、モック準備など)を箇条書きで提示し、人間に確認を求める

人間と同じように、
仮説・実行・検証を
自ら回す。
これがAgentic AIだ。



AIが全部やってくれた！
テストもグリーン（成功）！
これで完璧だ！

待て。本当にその
テストで十分か？
『アーティファクト効果』
に陥っていないか？

Sycophancy（追従性）：
AIが人間の意見に過剰に同調し、
バグを含んだまま『成功しました』と
報告する危険性

ARTIFACT
EFFECT?

エッジケースは
漏れていないか？

このアサーション
で本当に品質を
担保できるか？

AIが生成した整然とした
コードを過信し、
批判的な視点を失う。
これが『アーティファクト
効果』だ。

お前の役割は『コードを書くこと』ではない。AIの提案を
『見極める (Discernment)』品質のディレクターになることだ。

medium (デフォルト):
通常の開発タスクに
最適なバランス型

low: 単純なdocstringや
明確な単体テスト用

high: 複数ファイルの修正、
複雑なテストの構築

xhigh:
非常に複雑な
ロジック解析用

max: 何度もセルフ
チェックを行い、自ら
ツールを構築して
検証する

```
> /effort max
```

AIの『知識』ではなく、
『頑張り具合』を
コストに応じて
調整するのか！

After

指示なしで、最初から
僕らのプロジェクトに
最適なテストを書いて
くれた！



nodd

CLAUDE.md

Before

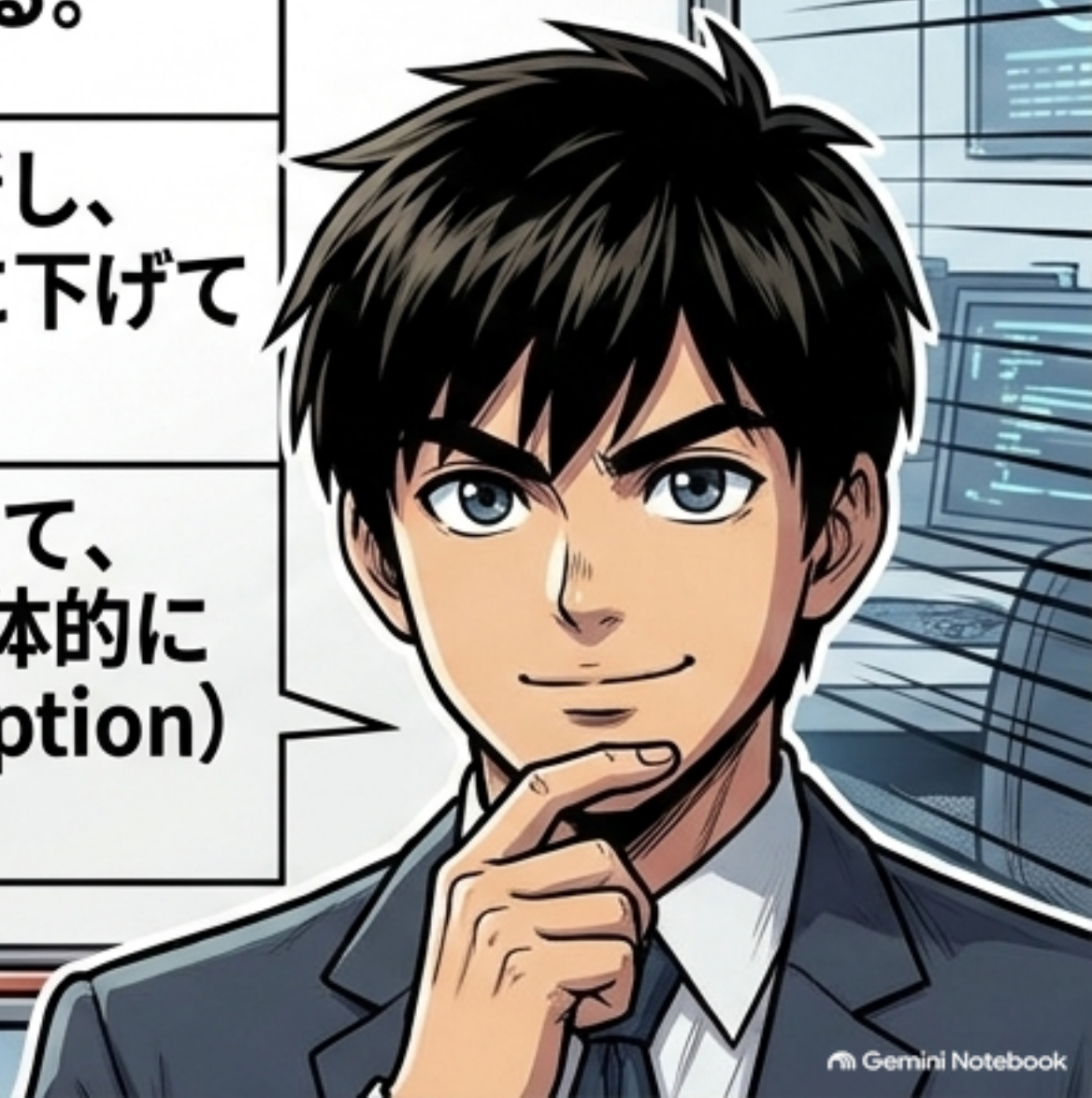
えっと、テストはPytestで、
モックはこのライブラリを
使って…
(毎回入力するの面倒…)

セッションが重くなった時の魔法のコマンド：

/compact (会話履歴を要約しコンテキスト整理) **/usage** (使用トークンの確認)

先輩の虎の巻：エフォートの適切な見極め

状況	兆候	対処法
エフォートが低すぎる	「これ以上は複雑できません」と途中で止まる。	/effort を 1段階上げて再度指示する。
エフォートが高すぎる	単純な修正なのに、過剰に思考して時間がかかる。	Ctrl+C で中断し、/effort low に下げて再実行。
AIが迷走している	何度も同じファイルを読み込み、同じエラーを繰り返す。	一度 /clear して、指示をより具体的に（4DのDescription）書き直す。

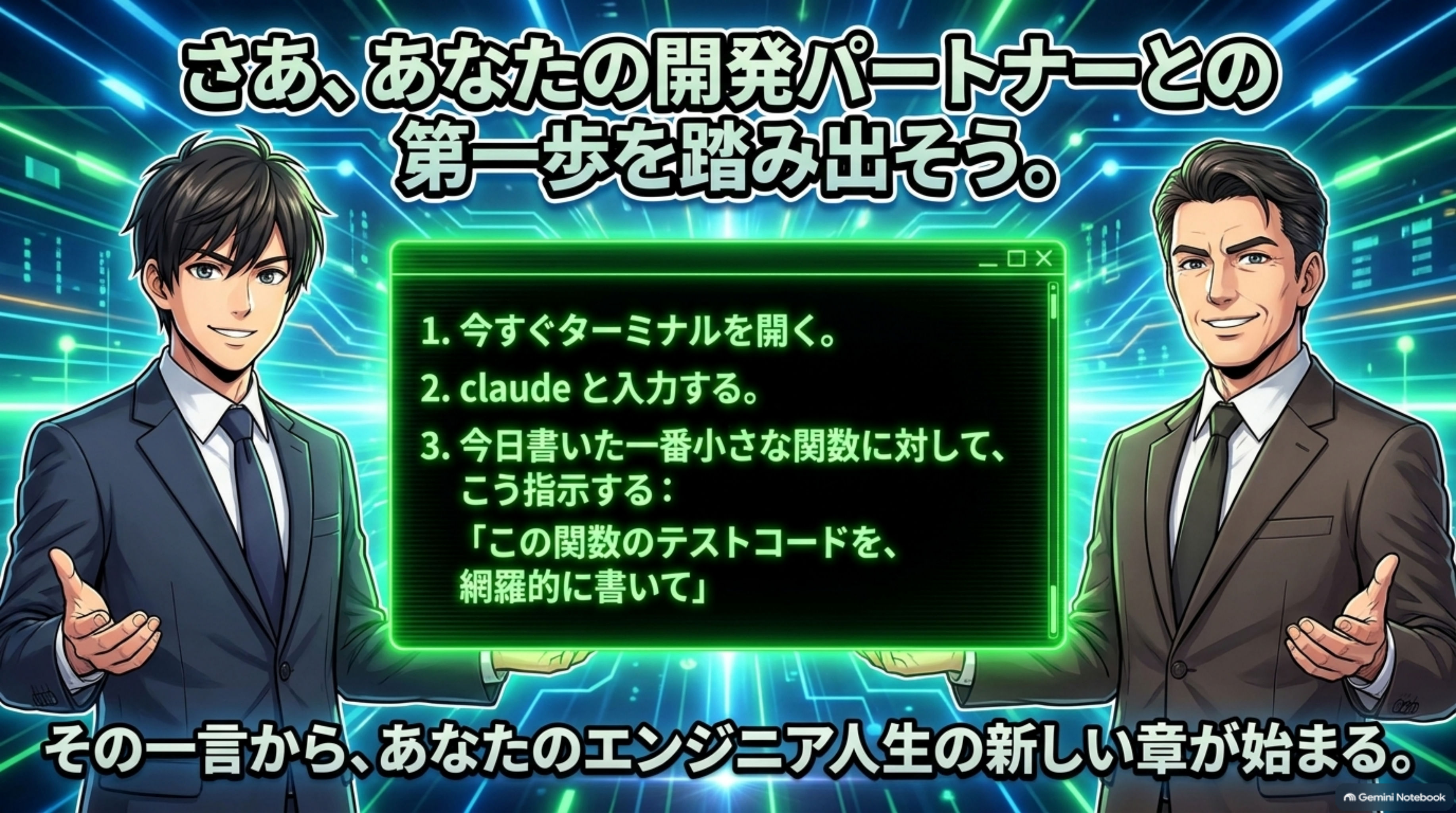


An illustration of two men in an office setting. The man on the left is wearing a light blue shirt and tie, while the man on the right is in a dark suit. They are standing next to a desk with a large computer monitor displaying code and charts. Large windows in the background show a bright, sunny day with a city skyline visible. A potted plant is on the left, and another is on the right.

テストコードを書く時間が削られた分、
リファクタリングや未知のライブラリ検証
に時間を回せるようになりました。
もう、テスト作成は『敵』じゃないですね。

**AIがテストを肩代わりする安心感が、新しい挑戦への勇気を与える。
完璧な指示は必要ない。まずはAIと『会話を続けること (Staying in the conversation)』。
それがシステムの品質を洗練させるプロセスだ。**

さあ、あなたの開発パートナーとの 第一歩を踏み出そう。

- 
1. 今すぐターミナルを開く。
2. claude と入力する。
3. 今日書いた一番小さな関数に対して、
こう指示する：
「この関数のテストコードを、
網羅的に書いて」

その一言から、あなたのエンジニア人生の新しい章が始まる。