HANDS-ON / 演習

ハンズオンガイド
演習3本とミニ演習7本

当日はこのページを開いたまま進めます。操作はすべて VSCode の Claude Code パネルで行い、作ったツールの実行は同じウィンドウの統合ターミナルで行います。演習を切り替えるたびに、その演習フォルダを VSCode で開き直します。答えと参考プロンプトはこのページには載せていません。配布フォルダの hints にあります。

2026年8月17日(月)・オンライン(Zoom)

題材は Python の小さな業務ツール4種（データはすべて架空）

手を動かす回数は 演習3本 ＋ ミニ演習7本 ＋ 準備4枠

このガイドの構成

準備 4枠	**環境立ち上げと個人設定**	ZIP の展開、公式 Skill のコピー、個人設定のバックアップ、フォルダの開き直し。
M1-M7	**ミニ演習7本**	画面の見どころ、CLAUDE.md 2層、statusLine、規約と Hook、Skill 化、自動化の提案。
演習 3本	**Lv1 / Lv2 / Lv3**	ハーネスがある状態、無い状態、揃っている状態の3段階を順に体験します。
巻末	**切り分けと後始末**	うまく動かないときの見どころ、用語集、個人設定の戻し方、情報源。

HANDS-ON 00

このガイドの読み方

本日は座学と手を動かす時間が交互に来ます。手を動かす枠は、準備4枠・ミニ演習7本・演習3本の合計14枠です。この節では、当日ずっと効いてくる3つの決まりを先に共有します。

00.1 演習ごとにフォルダを開き直します

配布フォルダの一番上は開きません。演習を切り替えるたびに、VSCode の「ファイル > フォルダーを開く」でその演習フォルダを開き直します。開き直すと会話は新しく始まり、前の演習のやり取りは持ち越されません。この進め方にしているのは、演習2 で「プロジェクト側のハーネスが無い」状態を実際に作るためと、答えの入った hints/ と _reference_完成形/ を開いたフォルダの外に置くためです。

回	開くフォルダ	位置	この回で確かめること
1	演習1_Lv1_集計ツール	午前の環境立ち上げ。昼休憩のあとにもう一度開きます	午前に入れたユーザー設定が、新しいセッションでも効いていること
2	演習2_Lv2_不具合修正	演習2 の前	プロジェクト側に .claude も CLAUDE.md も無いこと
3	予備_チケット管理	ミニ演習 M6 の前	.claude を持たないフォルダであること（M7 の「入れる前」の状態）
4	演習3_Lv3_ログ解析	演習3 の前	rules・Skill・サブエージェント・Hook が揃っていること

00.2 答えを見るタイミング

このガイドと各演習フォルダの README.md には、答えと参考プロンプトを書いていません。折りたたみの中にも入れていません。答えは配布フォルダの hints/ にだけあります。まず自分の言葉で指示を書き、5〜10分試して進まないときに、対応するヒントファイルを開いてください。各ヒントは「参考プロンプト」「詰まったときの見どころ」「それでも進まないときの答え」の3節に分かれていて、答えは一番下にだけ書いてあります。

場面	ヒントのファイル
M1 画面の見どころ	hints/M1_画面の見どころ_hint.md

場面	ヒントのファイル
M2 ユーザー設定 CLAUDE.md	hints/M2_ユーザー設定CLAUDE_hint.md
M3 statusLine の設定	hints/M3_statusline_hint.md
演習1 Lv1 集計ツール	hints/演習1_Lv1_集計ツール_hint.md
M4 実行サマリー3行	hints/M4_実行サマリー_hint.md
演習2 Lv2 不具合修正	hints/演習2_Lv2_不具合修正_hint.md
M5 コーディング規約と Hooks	hints/M5_規約とHooks_hint.md
M6 skill-creator	hints/M6_skill-creator_hint.md
M7 automation-recommender	hints/M7_automation-recommender_hint.md
演習3 Lv3 ログ解析	hints/演習3_Lv3_ログ解析_hint.md

> **_reference_完成形/ は研修中に開きません**
>
> 各演習の参考解が入っています。自分で Claude Code に指示して組み立てる過程が学びの中心なので、当日は開かないでください。研修後の振り返りで使ってください。

00.3 詰まったときの動き方

5〜10分は自分で試す → **節18 の切り分け表を見る** → **hints の該当ファイルを開く** → **チャットで講師に伝える**

講師に伝えるときは、症状（画面に出ている文字）と、直前に自分がやった操作の2つを書いてください。この2つがあると切り分けが早く済みます。講師は2名います。進行と個別のサポートで分かれて対応します。

このガイドの表記

- 所要時間は [Nmin] の形で書きます。時刻は書きません。当日の進み具合で前後します
- Windows と Mac は必ず併記します。キー操作、パス、Python のインタプリタ名の3つが対象です
- 入力するコマンドは黒い枠に1行ずつ入れています。枠の中の文字はそのまま打ってください
- 拡張で使えるかどうかを実機で確定していない機能には、必ず代替の確認方法を並べて書いています

確認方法を2つ書いてある箇所について

/context の Memory files 表示、/hooks の登録一覧、/ のメニューへの自作 Skill 表示は、VSCode 拡張で出るかどうかが環境によって変わります。出なくても壊れているわけではありません。このガイドでは、そうした箇所に「出る場合の確認」と「出ない場合の確認」を並べて書いています。片方で確認できれば先に進んでください。

HANDS-ON 01

当日の流れ

講義・演習の合計は [420min] です。ここに定着・休憩 [10min] を4回、昼休憩 [60min]、予備 [20min] を足すと [540min] になります。連続して手を動かすのは最長で86分です。押していても定着・休憩は削りません。

順	区分	項目	所要
1	講義	講師紹介（安田 光喜／松原 孝司）	[4min]
2	講義	イントロダクション。本日のゴール、進め方、演習ごとにフォルダを開き直す運用の説明	[8min]
3	準備	環境立ち上げ。ZIP の展開、演習1_Lv1_集計ツール を開く、信頼、Spark アイコン、サインイン確認	[10min]
4	ミニ演習	M1 画面の見どころ	[10min]

順	区分	項目	所要
5	準備	公式 Skill 2本を ~/.claude/skills/ へコピー	[10min]
6	講義	章1 生成AI最新トレンド	[28min]
-	休憩	定着・休憩	[10min]
7	講義	章2 Claudeファミリーの現在地	[22min]
8	講義	章3-A Claude Code の基礎（画面と位置づけ）	[24min]
-	休憩	定着・休憩	[10min]
9	講義	章3-B Claude Code の基礎（モデル・Effort・ファイルの渡し方・スラッシュコマンド）	[20min]
10	準備	個人設定のバックアップ。.bak の保存と deny ブロックの貼り付け	[6min]
11	ミニ演習	M2 ユーザー設定 CLAUDE.md	[8min]
12	ミニ演習	M3 statusLine の設定	[14min]
13	講義	章4 ハーネス3段階の地図	[12min]
-	休憩	昼休憩	[60min]
14	準備	フォルダの開き直し①　演習1_Lv1_集計ツール	[3min]
15	演習	演習1 Lv1 集計ツールの実装	[44min]
16	ミニ演習	M4 プロジェクト CLAUDE.md の実行サマリー	[12min]
17	準備	フォルダの開き直し②　演習2_Lv2_不具合修正	[3min]
18	演習	演習2 Lv2 前半。調査と1件目の修正	[24min]
-	休憩	定着・休憩	[10min]
19	ミニ演習	M5 コーディング規約と Hook の自然言語生成	[20min]

順	区分	項目	所要
20	演習	演習2 Lv2 後半。規約と Hook を効かせて残りを修正	[16min]
21	準備	フォルダの開き直し③　予備_チケット管理	[3min]
22	ミニ演習	M6 skill-creator で成功した流れを Skill 化	[18min]
23	ミニ演習	M7 automation-recommender で提案・導入・効果検証	[24min]
-	休憩	定着・休憩	[10min]
24	準備	フォルダの開き直し④　演習3_Lv3_ログ解析	[3min]
25	演習	演習3 Lv3 ログ解析パイプライン	[56min]
26	講義	振り返り、明日からの一歩、個人設定の後始末、質疑	[18min]
-	予備	予備・進行調整	[20min]
		合計（講義 [136min] ＋ 準備 [38min] ＋ ミニ演習 [106min] ＋ 演習 [140min]）	**[420min]**
		拘束時間（[420min] ＋ 定着・休憩 [10min]×4 ＋ 昼休憩 [60min] ＋ 予備 [20min]）	**[540min]**

演習2 が前後に分かれている理由

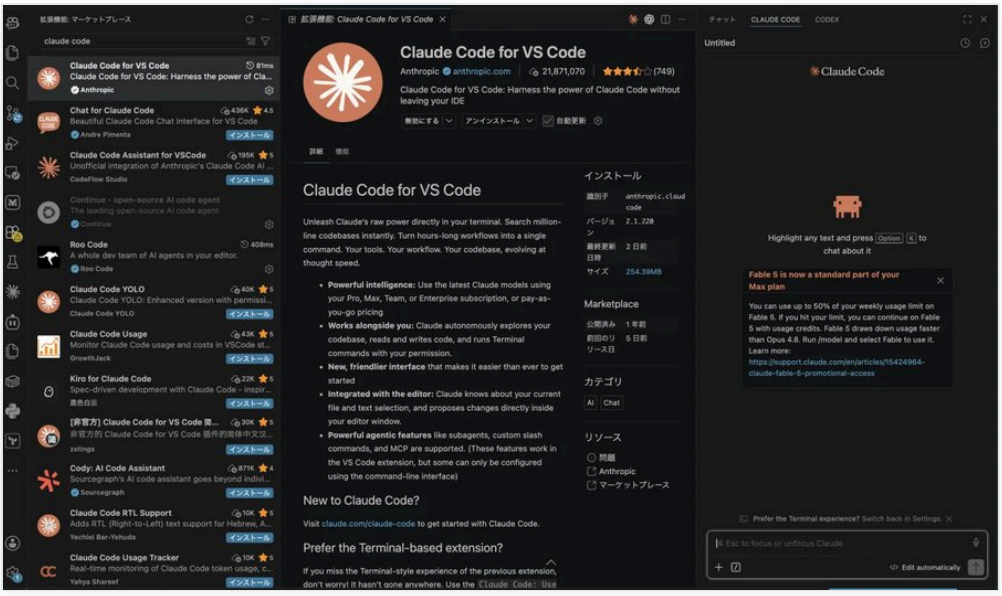
演習2 は前半（項目18）と後半（項目20）に分かれ、間に定着・休憩とミニ演習 M5 が入ります。プロジェクト側のハーネスが無い状態で一度詰まってから、規約と Hook を足して続きをやる、という順序にするためです。前半の到達点は「1件目の修正まで」に固定します。全部直そうとせず、時間が来たらそこで止めてください。

Python の実行は統合ターミナルで

作ったツールを動かすときは、VSCode の統合ターミナル（Windows は `Ctrl` + `@` ／Mac は `Cmd` + `@`）を開きます。パネルと統合ターミナルは同じウィンドウの中で行き来できます。作業ディレクトリは VSCode で開いたフォルダなので、パスを指定する必要はありません。

HANDS-ON 02

準備 環境立ち上げ



Claude Code 拡張を入れた VSCode の画面です。右上の Spark アイコンからパネルを開きます。

項目3 [10min] の枠です。演習素材を手元に置き、演習1_Lv1_集計ツール を開いてパネルが指示を受け付ける状態になるところまでを、全員でそろえます。VSCode・Python・Claude Code 拡張の導入とサインインは、事前セットアップガイドで済ませてある前提です。

準備 演習素材を開いてパネルを動かす [10min]

1 ダウンロードする

教材サイト https://0817cc.give-app.net から `nid-claudecode-handson_受講者用.zip` をダウンロードします。Git の操作はありません。

2 展開する

日本語とスペースを含まないパスに展開します。Windows は `C:\work\`、Mac は `~/work/` のような場所です。日本語やスペースを含むパスに置くと、統合ターミナルでの実行や設定ファイルのパス指定でつまずきます。展開先に `nid-claudecode-handson_受講者用` フォルダができていることを確認してください。

3 演習1 のフォルダを開く

VSCode の「ファイル > フォルダーを開く」で 演習1_Lv1_集計ツール を選びます。配布フォルダの一番上は開きません。

4 **信頼する**

「このフォルダー内のファイルの作成者を信頼しますか?」と出たら「はい、作成者を信頼します」を選びます。制限モード（Restricted Mode）のままだと拡張が動きません。

5 **パネルを開く**

エディタ右上の Spark アイコンをクリックします。ファイルを開いていないときは、ウィンドウ右下の Claude Code からも開けます。パネルが出ない場合は、コマンドパレット（Windows `Ctrl` + `Shift` + `P` ／Mac `Cmd` + `Shift` + `P`）で `Developer: Reload Window` を実行します。

6 **サインインを確かめる**

入力欄が使える状態であればサインイン済みです。サインインを促す表示が出た場合は、事前セットアップガイドの手順で入り直してください。

7 **動くことを確かめる**

パネルに「このフォルダに何が入っているか教えてください」と打ち、返答が返ることを確認します。ここまでで準備は完了です。

> 確認: フォルダ名が 演習1_Lv1_集計ツール になっていること、権限モードの表示が Manual になっていることの2点を見てください。

導入手順は事前セットアップガイドにあります

VSCode・Python・Claude Code 拡張の導入、サインイン、素材を開く手順、動作確認は、事前セットアップガイドに画面つきで載せています。ここでは同じ手順を書き直していません。必要な節は サインイン、演習素材を VSCode で開く、動作確認 です。

統合ターミナルで Python の名前を確かめておきます

統合ターミナル（Windows `Ctrl` + `@` ／Mac `Cmd` + `@`）を開き、自分の環境で通るインタプリタ名を先に確かめておくと、あとで迷いません。3.10 以降を返す方の名前を、一日ずっと使います。

```
py -V
python -V
python3 -V
```

Windows は py が通ることが多く、Mac は python3 です。py も python も通らない場合は、講師にお知らせください。

HANDS-ON 03

準備 公式 Skill 2本のコピー

項目5 [10min] の枠です。配布フォルダに同梱してある公式 Skill 2本を、ご自身のホームの .claude/skills/ へコピーします。この作業そのものが、個人スコープとプロジェクトスコープの違いを手で覚える教材になっています。

03.1 なぜコピーするのか

Skill の置き場所は2つあります。ホームの ~/.claude/skills/（個人スコープ）と、VSCode で開いたフォルダの .claude/skills/（プロジェクトスコープ）です。公式 Skill 2本は配布フォルダの一番上の .claude/skills/ に入っていますが、本日は配布フォルダの一番上を開きません。開いたフォルダの中にしか無いものは、そのフォルダを開いているときしか使えません。そこで、どのフォルダを開いても使える個人スコープへ移します。

置き場所	呼び方	読み込まれる範囲
~/.claude/skills/	個人スコープ	すべてのプロジェクト。どのフォルダを開いていても効きます
<開いたフォルダ>/.claude/skills/	プロジェクトスコープ	そのフォルダを開いて「信頼する」を選んだあとだけ

もう1つ、置き場所に関わる決まりがあります。skills フォルダの下にあるフォルダに .claude-plugin/plugin.json が入っていると、そのフォルダは単なる Skill ではなくプラグインとして読み込まれます。名前は <フォルダ名>@skills-dir になります。同梱している claude-code-setup がこの形で、中に claude-automation-recommender という Skill が入っています。プラグインとして読まれるものは、ふつうの Skill と違って上位のフォルダへ遡って探されません。起動したフォルダの .claude/skills/ か、個人スコープにあるものだけが読まれます。個人スコープへ入れるのはこのためです。

準備 2本を個人スコープへコピーする [10min]

1 コピー元を開く

展開した配布フォルダの中の、次の2つのフォルダです。フォルダごと、中身をそのまま移します。

```
nid-claudecode-handson_受講者用/.claude/skills/claude-code-setup/
nid-claudecode-handson_受講者用/.claude/skills/skill-creator/
```

2 コピー先を開く

ご自身のホームの .claude/skills/ です。skills フォルダが無い場合は自分で作ります。

OS	コピー先	開き方
Windows	%USERPROFILE%\.claude\skills\	エクスプローラのアドレス欄に %USERPROFILE%\.claude\skills と入力して Enter
Mac	~/.claude/skills/	Finder で Cmd + Shift + G を押し、~/.claude/skills と入力して Enter

3 コピーする

手段は2つあります。どちらでも構いません。

- エクスプローラまたは Finder で、フォルダをドラッグしてコピーする
- いま開いている 演習1_Lv1_集計ツール のパネルで Claude に頼む。.claude 配下への書き込みなので毎回確認が入ります。保護パスの挙動をそのまま体験できます

コピー後の形はこうなります。

```
<ホーム>/.claude/skills/claude-code-setup/
<ホーム>/.claude/skills/skill-creator/
```

4 読み込み直す

コマンドパレット（Windows `Ctrl` + `Shift` + `P` ／Mac `Cmd` + `Shift` + `P`）から `Developer: Reload Window` を実行します。SKILL.md の追加は同じセッションでも反映されますが、claude-code-setup はプラグインとして読まれるため、反映が次のセッションからになります。

生成物: ホームの .claude/skills/ に claude-code-setup と skill-creator の2フォルダ

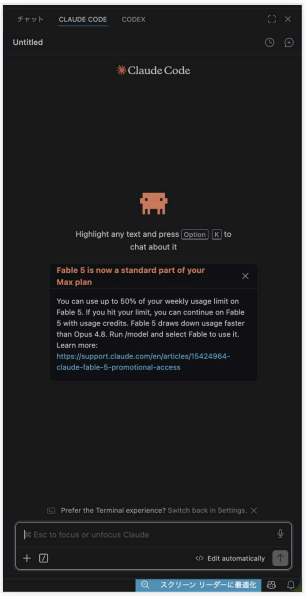
確認のしかた

コピー先のフォルダを開いて、claude-code-setup の中に skills/claude-automation-recommender/SKILL.md が、skill-creator の中に SKILL.md があることを目で確かめます。実際に呼ぶのは M6（skill-creator）と M7（claude-automation-recommender）です。

呼べないときの代替

個人スコープに置いても / のメニューに出ないことがあります。その場合は自然文で「skill-creator を使ってください」のように Skill の名前を書いて呼びます。それでも呼べない場合は、その演習フォルダの .claude/skills/ へ必要な Skill だけをコピーする形に切り替えます。M7 で使うのは claude-code-setup の中の skills/claude-automation-recommender/ です。対象のフォルダは 予備_チケット管理 の1つだけなので、切り替えても影響は1フォルダで済みます。

ミニ演習 M1 画面の見どころ



パネル全体です。入力欄の左下が Add context、その右がモードインジケータ、上部の履歴からセッションを切り替えます。

項目4 [10min] の枠です。以降ずっと使う4つの場所を、最初に全員で同じところを見て固定します。ここで見る4つが、そのまま「いま何が起きているか」を自分で確かめる手段になります。

04.1 何をするか

Claude Code のパネルの中で、モデル名・コンテキストの残量・使用量・Effort の4つを自分で開けるようにします。生成物はありません。開始時点の使用量とコンテキスト残量を手元にメモしておくと、研修の最後にもう一度開いたときの比較ができます。

番号	見る場所	そこで分かること
1	Context usage	コンテキストの残量。表示は「% of context remaining until auto-compact.」で、残っている割合です。圧縮が走ると「Conversation was compacted to free up context.」が出ます
2	Account & Usage	使用量のバーと内訳。メニュー表記は Account & usage…、説明は「View account info and usage limits」です。どのコマンドから、どの MCP サーバーから、どのプラグインから、どのサブエージェントから使ったかの割合が出ます
3	モデル切替	「Change the AI model」から、表示名と説明つきの一覧が開きます。当日の版数はここで実機を見ます
4	Effort	権限モードのメニューの中にある行です。操作の説明は「Click or drag to set effort level」「Click to cycle effort level」。既定の並びは low / medium / high で、xhigh は条件付きで出ます。選べる段階はモデル側の申告で変わります

M1 手順 [10min]

1 コンテキストの残量を見る

Context usage の位置を確かめます。いま何パーセント残っているかをメモします。

2 使用量と内訳を見る

Account & Usage を開きます。バーの位置と、内訳の項目を見ます。開始時点の値をメモします。

3 モデルの一覧を見る

モデル切替を開き、表示名と説明が並ぶことを見ます。今日はここを変えずに進めます。

4 Effort を見る

権限モードのメニューを開き、Effort の行を見ます。いまのレベルを確かめます。

5 権限モードを確かめる

権限モードが Manual であることを確認します。本日はこのモードで進めます。演習3 でだけ Plan に切り替えます。

> 考える: Account & Usage の内訳のうち、自分の環境で一番大きいのはどれでしたか。コンテキストが減る速さは何で決まると思いますか。

確認のしかた

4つの場所を、講師の画面を見ずに自分で開けたら達成です。権限モードが Manual になっていることも合わせて確かめてください。

詰まったときの代替

メニューの表記が上の表と違う場合があります。拡張は更新が速いため、名前が変わっていることがあります。似た意味の項目を探して、講師の画面と見比べてください。Effort の行が見つからない場合は、モデルが Effort に対応していない可能性があります。その場合はこの手順を飛ばして構いません。

画面の下に出る行の話

コンテキストや使用量を1行にまとめて表示する statusLine という仕組みがあります。これは設定ファイルに書きますが、拡張のチャットパネルには描画されません。パネルで見るのはここで確かめた Context usage と Account & Usage です。statusLine そのものは M3 で扱います。

参考プロンプトは hints/M1_画面の見どころ_hint.md にあります。

HANDS-ON 05

準備 個人設定のバックアップ

項目10 [6min] の枠です。このあとの M2 と M3 で、ご自身の個人設定（ホームの .claude/ 配下）を書き換えます。書き換える前に、元の状態を .bak として保存します。

この枠は飛ばさないでください

社内プロキシの設定（env の HTTPS_PROXY や NO_PROXY）を ~/.claude/settings.json に入れている方がいます。この JSON を壊すと、その日は Claude Code につながらなくなります。先に退避してから触ります。研修が終わったあとに戻す手順は節20 にあります。

準備 .bak を取って deny を貼る [6min]

1 settings.json を退避する

VSCode の「ファイル > 開く」で設定ファイルを開き、「名前を付けて保存」で settings.json.bak として同じフォルダに保存します。ファイルが無い方は、この時点では作りません。

OS	開くファイル
Windows	%USERPROFILE%\.claude\settings.json
Mac	~/.claude/settings.json

2 CLAUDE.md を退避する

同じ手順で、Windows は %USERPROFILE%\.claude\CLAUDE.md、Mac は ~/.claude/CLAUDE.md を CLAUDE.md.bak として保存します。

3 deny のブロックを貼る

個人設定の settings.json に、取り返しのつかないコマンドを止める設定を入れます。演習2 はプロジェクト側の設定を持たないため、安全網はここで持たせます。すでに permissions がある方は、deny の配列へ7行を追記します。

```
{
  "permissions": {
    "deny": [
      "Bash(rm -rf:*)",
      "Bash(rm -fr:*)",
      "Bash(sudo:*)",
      "Bash(mkfs:*)",
      "Bash(dd:*)",
      "Bash(shutdown:*)",
      "Bash(reboot:*)"
    ]
  }
}
```

同じ内容は配布フォルダの .claude/settings.json にも入っています。そちらからコピーしても構いません。

4 壊れていないことを確かめる

保存後、VSCode が JSON のエラーを出していないことを確認します。赤い波線が出ていたら、カンマの位置と括弧の対応を見てください。

生成物: ホームの .claude/settings.json.bak と .claude/CLAUDE.md.bak

プロキシ設定がある方へ

env のブロックには触らないでください。追記するのは permissions の deny だけです。既存のキーの並びは動かさず、末尾に足す形にしてください。

ミニ演習 M2 ユーザー設定 CLAUDE.md

項目11 [8min] の枠です。ホームの CLAUDE.md に指示を1つだけ置き、どのプロジェクトを開いていても効く層があることを体験します。

06.1 何をするか

CLAUDE.md は2つの層に置けます。ホームの ~/.claude/CLAUDE.md（すべてのプロジェクトに効く）と、開いたフォルダの CLAUDE.md（そのプロジェクトだけ）です。ここではホーム側に1行だけ足します。プロジェクト側は M4 で扱います。役割を分けておくと、2つが矛盾しません。ユーザー側は「ツールを呼ぶ直前の1行」だけ、プロジェクト側は「応答の末尾の3行」だけにします。

M2 手順 [8min]

1 ファイルを開く

VSCode で ~/.claude/CLAUDE.md（Windows は %USERPROFILE%\.claude\CLAUDE.md）を開きます。無ければ新規作成します。.bak は節5 で取ってあります。

2 末尾に追記する

次の内容をファイルの末尾に追記します。すでにある行は1行も消しません。同じ内容は配布フォルダの ミニ演習/ユーザー設定CLAUDE_md/追記する内容.md にもあります。

```
## ツールを使う直前の1行

ファイル編集・コマンド実行・検索のツールを呼ぶ直前に、次の1行だけを出力します。前置き・敬語・補足説明は書きません。

要約：使用するツール

「使用するツール」の位置には、これから呼ぶツールの名前を書きます。
```

要約：使用するツール は研修で指定された文言です。項目を足したり、順番を変えたりしないでください。コロンは全角です。

生成物: ホームの .claude/CLAUDE.md（追記。追記箇所が分かる見出しを1つ付けます）

3 **新しい会話を始める**

設定ファイルの内容は会話の開始時に読み込まれます。パネルで新しい会話を始めてください。

4 **読み取りだけの依頼を出す**

「data/work_log.csv の先頭3行を見せてください」のように、読むだけの依頼を出します。

> 考える: 1行が出なかった回があった場合、指示のどこが曖昧だったと思いますか。この指示を自社のリポジトリに置くとしたら、ユーザー層とプロジェクト層のどちらに置きますか。

確認のしかた

読み取りのツールを呼ぶ直前に、要約： で始まる1行が出れば達成です。/context に Memory files の一覧が出る環境なら、そこにホームの CLAUDE.md が並んでいることでも確認できます。一覧が開けない場合は、1行が出たことをもって読み込みの確認とします。

詰まったときの代替

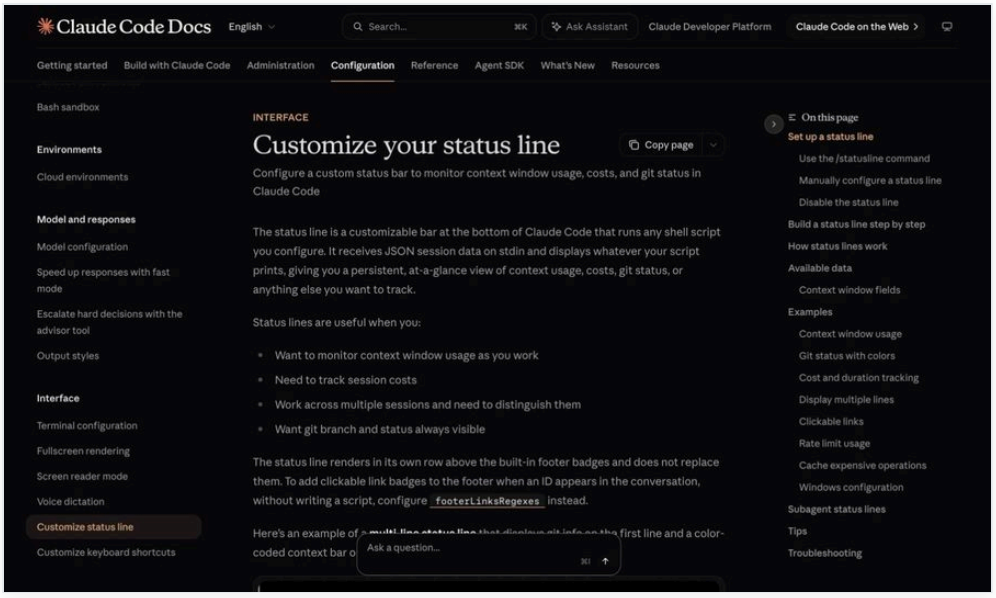
1行が出ない回があっても、壊れているわけではありません。CLAUDE.md はシステムプロンプトの一部ではなく、システムプロンプトのあとにユーザーからのメッセージとして届きます。読んで従おうとはしますが、毎回きっちり守られることは保証されていません。何度か依頼を変えて試し、それでも1回も出ない場合は、追記した見出しの位置とファイルのパスを確かめてください。

全角コロンと半角コロンが両方出ます

ユーザー設定側の 要約：使用するツール（全角コロン）と、M4 で入れるプロジェクト側の 要約:（半角コロン）は、別の行として両方出ます。片方が消える設定ではありません。層が違うので、どちらも効きます。

参考プロンプトは hints/M2_ユーザー設定CLAUDE_hint.md にあります。

ミニ演習 M3 statusLine の設定



公式ドキュメントのステータスライン解説です。コンテキスト使用量・セッションのコスト・レート上限を出せると書かれています。

項目12 [14min] の枠です。モデル名・コンテキスト消費率・セッションコストを1行にまとめる statusLine を、設定ファイルとスクリプトの両方を書いて作ります。

> **先に事実をお伝えします**
>
> `~/.claude/settings.json` の statusLine は、拡張と共有される正しい設定です。ただし**拡張のチャットパネルには描画されません**。拡張本体の実装に statusLine を描く処理が入っていないことを確認済みです。statusLine はコマンドラインインターフェース側の実行系の機能です。ですのでこのミニ演習の達成条件は「設定とスクリプトが正しく書けていること」で、画面に出ているかどうかは問いません。パネルで数字を見る話は M1 の `Context usage` と `Account & Usage` が担当します。

07.1 何をするか

statusLine は、指定したスクリプトを起動し、標準入力でセッションの JSON を渡し、返ってきた1行を表示する仕組みです。ここではその JSON から3つの値を取り出して1行にするスクリプトを作ります。仕組みを手で書いて理解するのが目的です。

出すもの	JSON のキー	注意
モデル名	`model.display_name`	そのまま表示できます
コンテキスト消費率	`context_window.used_percentage`	セッションの序盤は `null` になることがあります。`null` のときの既定値を自分で決めます

出すもの	JSON のキー	注意
セッションコスト	cost.total_cost_usd	クライアント側の概算です。請求額とは一致しません

M3 手順 [14min]

1 スクリプトを作る

ホームの .claude/ に statusline.py を新規作成します。標準入力の JSON から上の3つを取り出し、1行で出力します。完成品が配布フォルダの ミニ演習/statusline/statusline.py にあります。自分で書いても、完成品を写しても構いません。

生成物: ホームの .claude/statusline.py

2 完成品を使う場合のコピー手順

ミニ演習/ は VSCode で開いたフォルダの外にあるため、エクスプローラのドラッグでは持ってこられません。次のどちらかで移します。

- エクスプローラ（Windows）または Finder（Mac）で ミニ演習/statusline/statusline.py をコピーし、%USERPROFILE%\.claude\ または ~/.claude/ に貼り付ける
- VSCode で ミニ演習/statusline/statusline.py を開き、中身を全部コピーして、新規ファイルとして ~/.claude/statusline.py に保存する

3 settings.json に追記する

~/.claude/settings.json に statusLine を追記します。type は "command"、command にインタプリタとスクリプトのパスを書きます。Mac と Windows で書き方が違います。1つの設定で両方を賄うことはできません。同じ内容は ミニ演習/statusline/settings断片.json と docs_参照元/設定断片/statusline_settings断片.json にあります。

Mac の場合です。~ が使えます。

```
"statusLine": {
  "type": "command",
  "command": "python3 ~/.claude/statusline.py"
}
```

Windows の場合です。絶対パスで、区切りはスラッシュにします。<ユーザー名> はご自身のものに置き換えます。

```
"statusLine": {
  "type": "command",
  "command": "python C:/Users/<ユーザー名>/.claude/statusline.py"
}
```

統合ターミナルで py しか通らない環境では、python の部分を py に変えます。

生成物: ホームの .claude/settings.json の statusLine ブロック（追記）

4 スクリプト単体で動かす

統合ターミナルで、模擬の JSON をスクリプトに流します。1行が出て、終了コードが 0 になることを確かめます。模擬 JSON を流す確認コマンドは、Windows と Mac の両方を hints/M3_statusline_hint.md に載せてあります。

5 JSON が壊れていないことを確かめる

~/.claude/settings.json を開き、VSCode がエラーを出していないことを確認します。前のキーの行末にカンマが必要な点に注意してください。

> 考える: コンテキスト消費率を自分で計算する場合、入力トークンのどれを足しますか。output_tokens を含めない理由も考えてください。cost.total_cost_usd は請求と一致しない概算です。では何のために見る数字でしょうか。

確認のしかた

- ~/.claude/settings.json に statusLine が入っていて、JSON が壊れていない
- ~/.claude/statusline.py が模擬 JSON を受け取って1行を返し、終了コードが 0 になる
- 出力にモデル名・コンテキスト消費率・セッションコストの3つが含まれている

詰まったときの代替

表示の確認は求めません。パネルに出なくても、上の3点が満たせていれば達成です。表示まで見たい方は、拡張の設定 claudeCode.useTerminal でターミナルモードに切り替えると確認できる可能性があります。ただし戻し忘れると以降の演習の画面が変わってしまうため、任意扱いです。切り替えた方は、確認後に必ず元へ戻してください。

余力があれば

cost.total_cost_usd を出したまま /clear を1回実行し、新しいセッションで金額が 0 に戻ることを確かめてください。セッション単位の数字であることが目で見えます。さらに rate_limits.five_hour.used_percentage を1項目足すと、上限に対する使用量も1行に入ります。

参考プロンプトは hints/M3_statusline_hint.md にあります。

HANDS-ON 08

ハーネス3段階の地図

午後の演習3本は、環境の整い方が違う3つの段階を順に体験する構成です。この表を先に頭に入れてから午後に入ります。同じ表が進行スライドの章4 にも出ます。

ここでいうハーネスは、Claude に前提と手順をあらかじめ渡しておく仕組みのことです。CLAUDE.md・docs・Skill・サブエージェント・Hook・設定ファイルをまとめてこう呼びます。

段階	呼び方	環境の状態	やること	何を体で分かるか
Lv1	用意されたハーネスを使う	演習フォルダに CLAUDE.md が1枚、Skill が2本、docs/ が4種、.claude/settings.json が置かれています	/init を実行して読み込みを確認し、用意された Skill を呼びながら実装します	前提が書いてあると指示が短くて済むこと。Skill は手順の型であって魔法ではないこと
Lv2	プロジェクト側のハーネスが無い状態から作る	演習フォルダに CLAUDE.md も docs/ も .claude/ もありません。午前に入れたユーザー設定は効いています	/init で CLAUDE.md を作り、docs_参照元/ のひな型から規約を置き、Hook を自然言語から作らせます	プロジェクトの前提が無いと毎回同じことを書かされること。1から書く手数と、ひな型を持ってくる手数の差
Lv3	実用のパイプラインを組み合わせて回す	CLAUDE.md、.claude/rules/ 2本、Skill 3本、サブエージェント 2本、Hook 1本、permissions.deny、docs/ 5種が揃っています	用意されたパイプラインを使い分けて仕様から実装し、サブエージェントに下調べとレビューを任せ、Hook の指摘に応えます	予防と強制と分業が同時に効くと、指示の量ではなく確認の質が変わること

Lv2 で無いのはプロジェクト側だけです

無いのはプロジェクト側だけです。午前に入れたユーザー設定（~/.claude/CLAUDE.md のツール直前の1行、~/.claude/settings.json の statusLine と deny、~/.claude/skills/ にコピーした公式 Skill 2本）は効いたままです。演習2 でツール直前の1行が出続けても、設定が消えていないか確かめる必要はありません。想定どおりです。

Lv3 が正解というわけでもありません

Lv3 は作るのに手間がかかります。実務では、まず Lv1 相当の CLAUDE.md を1枚置き、次に効いた規約を Hook に上げる、という順で育てるのが現実的です。本日の3本は「順に育てるとどうなるか」を1日で通す並びになっています。持ち帰っていただきたいのは手順書ではなく、この育てる順序です。

CLAUDE.md で足りない領域があります

CLAUDE.md は文脈を伝える層です。読んで従おうとはしますが、厳密に守られることは保証されていません。毎回必ず走らせたいものは Hook で書きます。この線の引き方を、M5 と演習3 で実際に手を動かしながら確かめます。

HANDS-ON 09

演習1 Lv1 集計ツールの実装

項目15 [44min] の枠です。用意されたハーネスがある環境で、指示・生成・差分の確認・実行・検算 の1周を通します。フォルダを開き直す3分は項目14 で別に取ってあります。

開くフォルダ	演習1_Lv1_集計ツール
データ	data/work_log.csv（列は date, member, task, minutes。すべて架空）
用意されているもの	CLAUDE.md／.claude/settings.json／.claude/skills/read-data/SKILL.md ／.claude/skills/run-and-verify/SKILL.md／docs/ 4種
ヒント	hints/演習1_Lv1_集計ツール_hint.md

09.1 目的

前提を書いた CLAUDE.md と、手順の型である Skill が効いている状態を先に体験します。このあとの演習2 は、プロジェクト側のハーネスが無い状態から始めます。そのときの差を測る基準がこの演習です。

09.2 操作

演習1 集計ツールを作る [44min]

1 フォルダを開き直す

VSCode の「ファイル > フォルダーを開く」で 演習1_Lv1_集計ツール を開き、信頼を選びます。昼休憩の前に開いていた方も、ここでもう一度開き直します。午前に入れたユーザー設定が、新しいセッションでも効いていることを確かめる回でもあります。

2 パネルとターミナルを用意する

Spark アイコンでパネルを開きます。統合ターミナル（Windows `Ctrl` + `@` ／Mac `Cmd` + `@`）も開いて、Python の名前を確かめておきます。Windows は `py -V`、Mac は `python3 -V` です。

3 CLAUDE.md を自分で読む

`CLAUDE.md` をエディタで開いて中身を読み、書かれている前提がいくつあるかを数えます。この時点で、どの行が今日の実装に効くかを予想してください。

4 /init を実行する

このフォルダには `CLAUDE.md` が既にあるため、追記の提案が差分で出ます。**既存の行を削る提案は却下してください。**判断に迷ったらいちど却下し、「既存の行は変えずに、不足している項目だけ追記してください」と言い直します。

5 読み込まれていることを確かめる

/context に Memory files の一覧が出る環境なら、そこで確認します。出ない場合は「CLAUDE.md に書いてある Path(__file__).parent という語を含めて、このプロジェクトの前提を3つ挙げてください」と聞き、書いた語が返るかで判断します。どちらか片方で足ります。

6 /read-data を呼ぶ

`data/work_log.csv` の列構成と行数を報告させます。/ のメニューに出ない場合は、自然文で「read-data の Skill を使ってください」と頼みます。

7 **設計を相談する**

実装に入る前に、読み込み・集計・表示の分け方と、集計結果を保持するクラスの持たせ方を相談します。いきなり実装させず、方針を言葉にさせてください。

8 **実装させる**

差分は1件ずつ読んで承認します。まとめて承認しないでください。権限モードは Manual のまま進めます。

生成物: 演習1_Lv1_集計ツール/aggregate.py

9 **/run-and-verify を呼ぶ**

統合ターミナルで実行して、合計を検算します。実行は python3 aggregate.py（Windows は py aggregate.py）です。作業ディレクトリは開いたフォルダなので、パス指定は不要です。

09.3 自分で考える

- CLAUDE.md に書いてあったおかげで、自分が書かずに済んだ指示はどれでしたか
- /read-data を呼ばずに「CSV を集計するツールを作ってください」とだけ頼んだ場合、何が違ったと思いますか。時間があれば1回試して比べてください
- SKILL.md を2本とも開いて読み、自分の言葉で書ける範囲の内容かどうかを判断してください。書ける範囲であれば、ご自身の職場の手順も同じ形にできます

09.4 生成物の名前と場所

aggregate.py	このフォルダの直下に新規作成します
CLAUDE.md	/init が追記した状態。既存の行は残ったままです

発展課題で書き出すファイルは data/ の外に置きます。.claude/settings.json にある Edit(data/**) の1行で、data/ 配下への書き込みが止まります。

09.5 OK基準

この7つが揃えば達成です

- 統合ターミナルで python3 aggregate.py（Windows は py aggregate.py）がエラーなく動く
- メンバー別の合計稼働分数が、多い順に表示される
- 表示された合計が、data/work_log.csv から数え直した値と一致する。行数は58、minutes の総計は4875、メンバーは5名、作業種別は6種です
- 読み込み・集計・表示が、別々の関数に分かれている
- 集計結果を保持するクラスがあり、メンバー別と作業種別の両方を取り出せる
- CLAUDE.md が読み込まれていることを確認できている（/context の Memory files に出る、または CLAUDE.md に書いた語が応答に現れる。どちらか片方で可）
- data/work_log.csv の中身が変わっていない（行数58、総計4875 のまま）

09.6 追加と考察

CLAUDE.md の「CSV から読んだ数値は、計算する前に int または float へ変換します」の1行を、一時的に消します。消したあとで同じ内容の依頼をもう一度出し、出来上がりのコードにどんな差が出るかを見てください。確認が済んだら、消した行を元に戻します。
結果は「毎回こうなる」ではなく「この回はこうなった」と記録してください。1回の比較で分かるのはそこまでです。

09.7 発展課題

作業種別のランキングを CSV に書き出すサブコマンドを追加します。条件は2つです。既存のメンバー別の表示を変えないこと。書き出し先を data/ の外にすることです。data/ へ書こうとすると、Edit(data/**) の1行で止まります。この発展課題は M4 でそのまま使います。

つまずいたとき

症状	見るところ
/init の差分が既存の CLAUDE.md を書き換えている	差分を読み、既存行を削る提案は却下します。承認したあとに消えていた場合は、エディタの元に戻す操作で戻してから、追記だけを頼み直します。元の内容は hints/演習1_Lv1_集計ツール_hint.md にあります
Skill が / のメニューに出ない	フォルダの信頼を選んでいるかを確認します。それでも出ない場合は、自然文で Skill 名を書いて呼びます
統合ターミナルで python3 が見つからない	Windows のインタプリタ名は py です。docs/演習設定.md の表を見てください
パネルが開かない、拡張が反応しない	制限モードのままです。フォルダを開き直して信頼を選びます
data/work_log.csv を書き換えようとして止まる	想定どおりの挙動です。書き出し先を data/ の外に変えます

HANDS-ON 10

ミニ演習 M4 プロジェクト CLAUDE.md の実行サマリー

項目16 [12min] の枠です。演習1 のフォルダを開いたまま、そのフォルダの CLAUDE.md に応答の書式を足します。M2 で入れたユーザー層の1行と役割が分かれていることを確かめます。

10.1 何をするか

演習1_Lv1_集計ツール/CLAUDE.md の末尾に、応答の末尾へ付ける3行を書きます。ユーザー層は「ツールを呼ぶ直前の1行」、プロジェクト層は「応答の末尾の3行」です。同じ場所に似た別書式の指示を置くと矛盾になり、どちらかが任意に選ばれます。層で役割を分けると矛盾しません。

M4 手順 [12min]

1 3行を追記する

演習1_Lv1_集計ツール/CLAUDE.md の末尾に見出しを付けて、次の3行を書きます。ラベルは逐語で指定します。コロンは半角です。

```
要約: この応答でしたことを1文
読んだファイル: 読んだファイルのパスをカンマ区切り。無い場合は なし
使ったハーネス: CLAUDE.md / Skill名 / Subagent名 / Hook名 をカンマ区切り。無い場合は なし
```

生成物: 演習1_Lv1_集計ツール/CLAUDE.md（追記）

2 依頼を1つ出す

演習1 の発展課題（作業種別ランキングの書き出し）を依頼します。すでに終えている方は、その続きで構いません。

3 3行が付くかを見る

応答の末尾に3行が、指定の順で付くかを見ます。「使ったハーネス」に、実際に呼んだ Skill の名前が入っているかも見てください。

4 /compact を1回実行する

圧縮のあとも3行が続くかを見ます。プロジェクト直下の CLAUDE.md は、圧縮を越えてディスクから読み直されます。

> 考える:「無い場合は なし」まで書いた効果はありましたか。ユーザー層の1行とプロジェクト層の3行が両方出るとき、順番はどうなりましたか。

確認のしかた

- 依頼への応答の末尾に3行が、指定の順で付く
- 「使ったハーネス」に、その応答で実際に使ったものの名前が入っている
- /compact のあとも3行が続く

詰まったときの代替

3行が出ない場合は、ラベルを固定の文字列にしているかを確かめます。「使ったもの:」のような曖昧な言い方にすると出力が崩れます。CLAUDE.md が効く書き方は4点です。200行未満に収めること、Markdown の見出しと箇条書きで構造を付けること、検証できる粒度で具体的に書くこと、矛盾を置かないことです。/compact のあとに3行が消えた場合は、圧縮の直後にもう1回依頼を出して確かめてください。

余力があれば

3行のうち1行のラベルを曖昧な言い方に変え、出力が崩れるかを見てください。確認後は戻します。もう1つ、「毎回必ず」を機械的に担保する方法も考えてみてください。応答を検査するフックを置く構成が候補になりますが、拡張での挙動は未検証です。本日は考えるところまでにします。

同じ3行は演習2 でゼロから自分で書きます。写しで構いません。参考プロンプトは hints/M4_実行サマリー_hint.md にあります。

HANDS-ON 11

演習2 Lv2 不具合修正 前半

項目18 [24min] の枠です。プロジェクト側のハーネスが無い状態から始めます。前半の到達点は「1件目の修正まで」に固定します。全部直そうとせず、時間が来たらそこで止めてください。残りは M5 のあとの後半（節13）で続けます。

このフォルダに無いもの、効いているもの

このフォルダには CLAUDE.md も docs/ も .claude/ もありません。抜けているのではなく、無い状態を体験していただくために外してあります。午前に入れたユーザー設定（~/.claude/CLAUDE.md のツール直前の1行、~/.claude/settings.json の statusLine と deny、~/.claude/skills/ にコピーした公式 Skill 2本）は効いたままです。ツール直前の1行が出続けますが、想定どおりです。

開くフォルダ	演習2_Lv2_不具合修正
対象	monthly_report.py
データ	data/work_log_july.csv（列は date, member, minutes。すべて架空）

演習設定	このフォルダは docs/ を持たないため、開くフォルダ・信頼・インタプリタ名の確認・OK基準は README.md に集約しています
ヒント	hints/演習2_Lv2_不具合修正_hint.md

11.1 目的

プロジェクト側のハーネスが何も無い状態から始めて、詰まった箇所を材料にハーネスを自分で組み立てます。測るのは「自分が指示として書いた行数」です。前半（何も無い状態）と後半（規約とフックを置いた状態）で数えて比べます。1から書く手数と、ひな型を持ってくる手数の差が、この演習で見たいものです。

11.2 症状

monthly_report.py には不具合が3件あります。原因はここには書きません。見えている症状だけ書きます。

- 実行すると、まずエラーで止まります
- 1件直すと、また別のエラーで止まります
- エラーが出なくなっても終わりではありません。合計が本来より少なく見える状態が残ります

3件目は実行が最後まで通るため、実行できたことだけでは直ったと判断できません。出力された合計そのものの妥当性まで確認してください。

11.3 操作（前半）

演習2 前半 **調査と1件目の修正 [24min]**

1 **フォルダを開き直す**

VSCode の「ファイル > フォルダーを開く」で 演習2_Lv2_不具合修正 を開き、信頼を選びます。信頼していないと、あとで作る .claude/settings.json のフックが効きません。エクスプローラで .claude/ も CLAUDE.md も無いことを目で確かめてください。

2 **インタプリタ名を確かめる**

統合ターミナルで、Windows は `py -V`、Mac は `python3 -V` を実行します。3.10 以降を返す方の名前を、この演習の間ずっと使います。

3 **実行して症状を読む**

そのまま実行します。データの場所はスクリプトが自分の位置から解決するので、カレントディレクトリを移動する必要はありません。

```
py monthly_report.py 2026-07
```

Mac は次のとおりです。

```
python3 monthly_report.py 2026-07
```

出た症状をそのまま読みます。まだ直しません。

4 **/init を実行する**

何も無い状態から `CLAUDE.md` が作られる様子を見ます。200行を超えていたら削ります。長いほどコンテキストを食い、守られにくくなります。

5 **調査だけを頼む**

Claude Code に**原因の調査だけ**を頼みます。この時点では修正させません。何が起きているかを日本語で説明させます。

6 **1件目を修正させる**

差分を1件ずつ読んで承認します。

生成物: 演習2_Lv2_不具合修正/monthly_report.py（修正）・CLAUDE.md（新規）

7 **再実行してメモを取る**

もう一度実行して、次の症状が出ることを確認します。ここまでで自分がパネルに打った指示を見返し、行数を数えます。同じ前提（値の扱い方、実行して確認すること、出力の書式を変えないこと）を何回書いたかも数えてください。

	自分が書いた指示の行数	同じ前提を書いた回数
前半（ハーネス無し）		
後半（規約とフックあり）		

数えるのはパネルに自分が打った文だけです。Claude の応答は数えません。空行も数えません。

11.4 自分で考える

- 前半で同じ前提を何回書きましたか。書かずに済ませるには、その前提をどこに置くのが自然でしたか
- /init が作った CLAUDE.md のうち、実際に効いた行はどれでしたか。効かなかった行は、何が曖昧だったからでしょうか
- ひな型から持ってきた規約と、自分の言葉に直した項目のどちらがフックで判定しやすかったですか
- 3件目のようにエラーを出さない不具合を、次からどうやって見つけますか。フックで拾えるものと拾えないものの線はどこにありますか

11.5 生成物の名前と場所

生成物	場所
修正した monthly_report.py	このフォルダ直下（既存ファイルの更新）
CLAUDE.md	このフォルダ直下（新規作成。/init で作って手で足します）

生成物	場所
docs/コーディング規約.md	このフォルダの docs/（M5 で配置。引用元は docs_参照元/コーディング規約_ひな型.md）
.claude/settings.json	このフォルダの .claude/（M5 で作成。PostToolUse の登録）
指示の行数メモ	上の表に手で書き込みます

11.6 OK基準

判定は後半（節13）が終わった時点で行います。前半だけでは満たせません。

この9つが揃えば達成です

- 症状3件が出なくなった状態で、最後までエラーなく実行できる
- メンバー5名が全員表示される
- 表示された合計時間が 80.0 時間 になっている
- 実行できたことだけで判定していない。合計そのものの妥当性を確認した記録がある
- CLAUDE.md が読み込まれていることを確認できている（/context の Memory files、または CLAUDE.md に書いた語が応答に現れることのどちらか）
- docs/コーディング規約.md があり、機械が読む設定ブロックを含んでいる
- .claude/settings.json に PostToolUse の登録が1件ある（/hooks が開ける環境なら一覧で確認。開けない場合は .claude/settings.json をエディタで開いて読み合わせる）
- わざと規約に反するコードを書かせたときに、指摘が会話に出る
- data/work_log_july.csv の中身が変わっていない

11.7 追加と考察

docs/コーディング規約.md の機械が読む設定を1つだけ書き換えて、判定が変わることを確認します。forbid_print の値で見るとわかりやすいです。規約という文書とフックという強制が、同じ1つの正本を見ている関係を目で確かめます。確認したら元に戻してください。

PostToolUse は編集が終わったあとに走ります。編集そのものは止まりません。止めたい場合は PreToolUse になります。この違いも合わせて言葉にしてください。

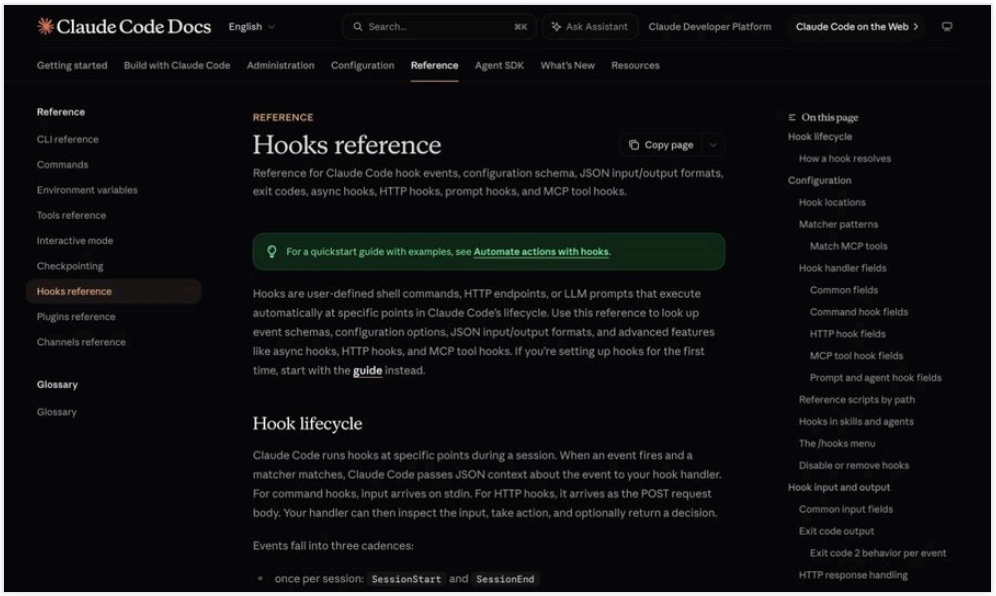
11.8 発展課題

4件目の不具合を自分で仕込みます。仕込んだら、**原因を書かずに症状だけ**を伝える文を作り、Claude Code に調査させます。症状だけで伝える練習です。余力があれば、docs/コーディング規約.md をひな型を見ずに1から書きます。この題材に必要な項目だけを5つ選び、フックの判定がどう変わるかを見ます。

つまずいたとき

症状	対処
症状が出る順番が隣の方と違う	修正の順序が違うと次に出る症状が変わります。順序は問いません
python3 が見つからない	Windows のインタプリタ名です。py -V と python -V を先に確認します
パネルが動かない	制限モードです。フォルダを開き直して信頼を選びます
/init が作った CLAUDE.md が長すぎる	200行未満を目標に削ります
前半で時間を使い切りそう	前半の到達点は1件目の修正までです。調査に深入りせず、時間が来たら止めます
ユーザー設定側の指示が出続けて戸惑う	無いのはプロジェクト側だけです。~/.claude/CLAUDE.md に入れた1行は効いたままです

ミニ演習 M5 コーディング規約と Hook の自然言語生成



公式のフック仕様です。フックは JSON を標準入力で受け取り、終了コードで結果を返します。

項目19 [20min] の枠です。演習2 の前半で詰まった状態のまま、規約という文書と、Hook という強制を自分で置きます。置いたあとに演習2 の後半へ進みます。

12.1 何をするか

規約は白紙から書きません。docs_参照元/コーディング規約_ひな型.md は、コピーすればそのままフックの正本として動く完成度で用意してあります。ここでの作業は「コピーして2項目だけ自分の言葉に直す」です。規約を1から書く体験は発展課題に回します。
このひな型には、人が読む説明と、機械が読む設定ブロックが同じファイルに入っています。フックはこの設定ブロックを正本として判定します。規約を直すと判定も変わる関係を、目で追える形にしてあります。

M5 手順 [20min]

1 ひな型をコピーする

docs_参照元/コーディング規約_ひな型.md を、演習2_Lv2_不具合修正/docs/コーディング規約.md としてコピーします。docs フォルダが無ければ作ります。docs_参照元/ は開いたフォルダの外なので、エクスプローラまたは Finder でコピーするか、VSCode で開いて中身をコピーして新規保存します。

2 2項目を自分の言葉に直す

直すのは2つだけです。機械が読む設定の max_line_length の値と、人が読む説明のうち1行です。ほかは触りません。

生成物: 演習2_Lv2_不具合修正/docs/コーディング規約.md

3 Hook を自然言語で作らせる

パネルで、フックを作るよう依頼します。依頼文には次の4つを必ず入れてください。

- 発火は Python ファイルの編集直後（PostToolUse、matcher は Write|Edit|MultiEdit）
- 判定の正本は docs/コーディング規約.md の設定ブロック
- 違反があれば会話に指摘が返ること
- 指摘は最大3点まで

本線は type: "prompt" の PostToolUse にします。シェルを使わないので、Windows の Git Bash の有無や python と python3 の違いに左右されません。prompt の中には $ARGUMENTS を入れてもらいます。フックへの入力の JSON がここに入り、どのファイルが編集されたかを判定に使えます。

生成物: 演習2_Lv2_不具合修正/.claude/settings.json（PostToolUse の登録）

4 差分を読む

生成された .claude/settings.json の差分を1行ずつ読みます。.claude 配下への書き込みなので、権限モードに関わらず確認が入ります。

5 動作を確かめる

わざと規約に反するコードを書かせて、指摘が会話に出るかを見ます。指摘に従って直すと、次の編集で指摘が出なくなることも確かめます。

> 考える:「最大3点まで」を入れなかったら何が起きると思いますか。このフックは、予防（ガイド）と事後の検知（センサー）のどちらでしょうか。判断が LLM 側にあるか、決められた手順で機械的に決まるかという軸で見ると、どちらに入りますか。

確認のしかた

- .claude/settings.json に PostToolUse の登録が1件ある
- わざと規約に反するコードを書かせると、差分の承認直後に指摘が会話に出る
- 修正を承認すると、もう一度フックが走って指摘が出ない
- 指摘の件数が3点を超えない

/hooks で登録の一覧が開ける環境なら、そこで確認します。開けない場合は .claude/settings.json をエディタで開いて読み合わせてください。

依頼文に「同じファイルへの判定は1回だけ」を入れない理由

この条件は依頼文に入れません。prompt 型のフックでは効かないからです。prompt 型は編集のたびに独立して起動するため、前の回に何を判定したかを覚えていないからです。回数を絞るのは command 型の実装側の仕事で、1回の実行で3点に絞る形で作ります。演習3 に配布してあるフックがそれです。同じ言葉でも、型が違うと効くところが違います。この違いを言葉にしてみてください。

うまく動かないときの切り分け

3点をこの順で見ます。登録が見えるか。フォルダの信頼を受け入れているか（プロジェクトのフックは信頼したあとに有効になります）。command 型にした場合はインタプリタ名が合っているか。JSON が壊れて設定が全部効かなくなった場合は、壊れたブロックだけ削除します。元の内容は hints/M5_規約とHooks_hint.md にあります。

余力があれば

規約の設定ブロックの forbid_print を true に変え、print を含むコードを書かせて指摘が出ることを確かめてください。確認後は false に戻します。規約を1から書きたい方は、ひな型を見ずに、この題材に必要な項目だけを5つ選んで書き、判定が変わるかを見てください。

参考プロンプトと、壊れたときに手で戻す元の内容は hints/M5_規約とHooks_hint.md にあります。

演習2 Lv2 不具合修正 後半

項目20 [16min] の枠です。M5 で置いた規約とフックが効いている状態で、残っている症状を直し切ります。目的・生成物・OK基準・発展課題は節11 に書いてあります。ここでは操作の続きだけを扱います。

演習2 後半

規約とフックを効かせて直し切る [16min]

8 残りを修正させる

規約とフックが効いている状態で、残っている症状を修正させます。フォルダは開いたままです。開き直す必要はありません。

9 フックの指摘に応える

フックの指摘が会話に返ってくることを確認します。指摘が出たら、それに従って直します。指摘は1回の実行で最大3点までです。3点を直すと、次の編集で残りが次の3点として出ます。

10 実行して検算する

最後までエラーなく実行できることを確認し、表示された合計を自分の手計算と突き合わせます。合計時間は 80.0 時間、メンバーは5名です。実行が通ったことだけで完了と判断しないでください。

11 前半と比べて記録する

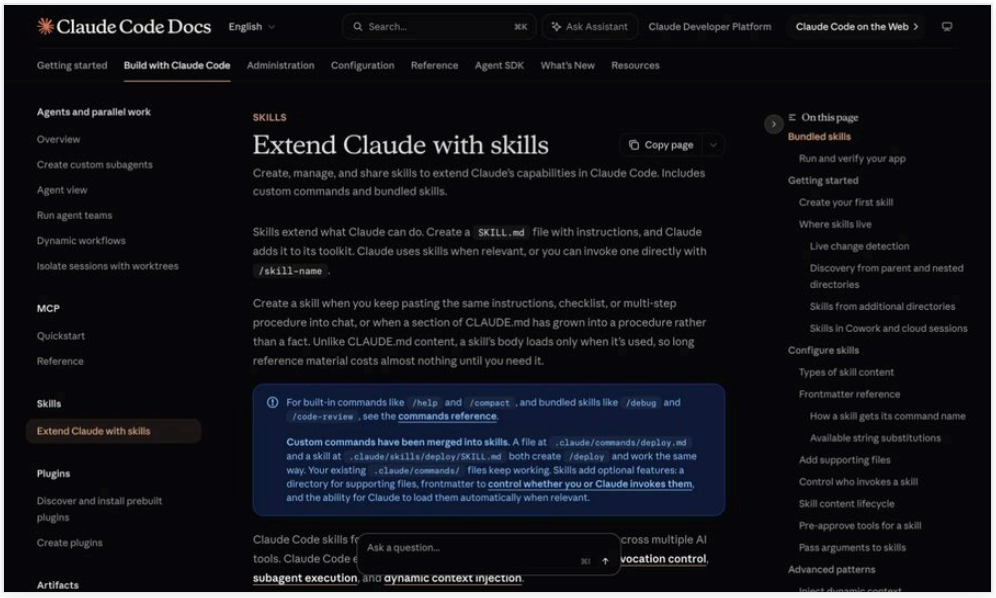
後半で自分が打った指示の行数を数え、節11 の表に書き込みます。前半と比べて1行で記録してください。

> 考える: 減った行数は、誰が代わりに書いていたことになりますか。減った分のうち、規約が担った部分とフックが担った部分を分けて言えますか。

ここまでで演習2 は完了です

OK基準9項目は節11 の 11.6 にあります。全部を埋めきれなくても構いません。前半と後半で自分の書いた量がどう変わったかを記録できていれば、この演習の目的は達成しています。

ミニ演習 M6 skill-creator で成功した流れを Skill 化



公式の Skill 解説です。カスタムコマンドは Skills に統合され、どちらの置き方でも同じスラッシュコマンドになります。

項目22 [18min] の枠です。演習2 で通った「実行して落ちる、原因を調べさせる、1件ずつ直す、再実行する、合計を検算する」という流れを、次回も再現できる形にします。作って終わりにせず、別のフォルダで1回使います。

14.1 何をするか

公式の skill-creator を呼び、自分の SKILL.md を1本作ります。作る先は個人スコープです。演習ごとにフォルダを開き直す進め方なので、プロジェクトスコープに作ると次のフォルダで使えません。作った Skill は、いま開く 予備_チケット管理 の ticket_app.py に対して1回使います。

M6 手順 [18min]

1 フォルダを開き直す

VSCode の「ファイル > フォルダーを開く」で 予備_チケット管理 を開き、信頼を選びます（項目21）。このフォルダには CLAUDE.md も .claude も docs もありません。M7 で自分が入れる前の状態です。エクスプローラで確かめておいてください。

2 /clear を実行する

skill-creator の SKILL.md は英語で34KB あります。読み込むだけでコンテキストを大きく使うため、前の会話を残したまま呼ぶと圧縮が走ります。必ず先に /clear してください。

3 skill-creator を呼ぶ

依頼文には次の3つを必ず入れます。

- 日本語で質問してください（SKILL.md が英語のため、英語で聞き返してくることがあります）
- 評価（eval）とテストケースの実行はしない。SKILL.md を書くところまでで止める
- 対象は「Python スクリプトを実行して落ちたところから、原因を調べ、1件ずつ直し、再実行して合計を検算する」流れ

/ のメニューに出ない場合は、自然文で「skill-creator を使ってください」と書いて呼びます。

4 4つの質問に答える

何をできるようにするか、いつ発火するか、出力の形式、テストケースを作るかの4点を聞かれます。4点目は「作りません」で通してください。

5 description を自分で直す

生成された SKILL.md を自分で読み、description を直します。「いつ使うか」を description に書き切るのが公式の方針です。

生成物: ホームの .claude/skills/<英数字とハイフンの名前>/SKILL.md

6 実際に1回使う

いま開いている 予備_チケット管理/ticket_app.py に対して、作った Skill を呼びます。演習2 で踏んだ手順が再現されるかを見ます。

> 考える: 自分が書いた手順のうち、Skill に残す価値があったのはどれで、その場限りだったのはどれですか。description を直した理由を1行で書いてください。

確認のしかた

- SKILL.md が個人スコープの .claude/skills/<名前>/ にあり、frontmatter に name と description がある
- ticket_app.py に対して呼んだときに、実行・症状の説明・修正・再実行・検算 の流れが再現される

呼び出しは /<フォルダ名> でも、自然文で Skill 名を指定する形でも構いません。/ のメニューに並ぶかどうかは環境によって変わります。

詰まったときの代替

SKILL.md には「途中で止めるな」「評価ビューアを生成せよ」と強く書かれています。依頼文の1行目で範囲を切ってください。範囲を切っても評価に進もうとする場合は、「そこで止めてください」ともう一度伝えます。コマンド名はフォルダ名で決まり、frontmatter の name は表示名にしか使われません。フォルダ名に日本語や空白を入れると呼べなくなります。英数字とハイフンだけにしてください。

この Skill の一部は本日使いません

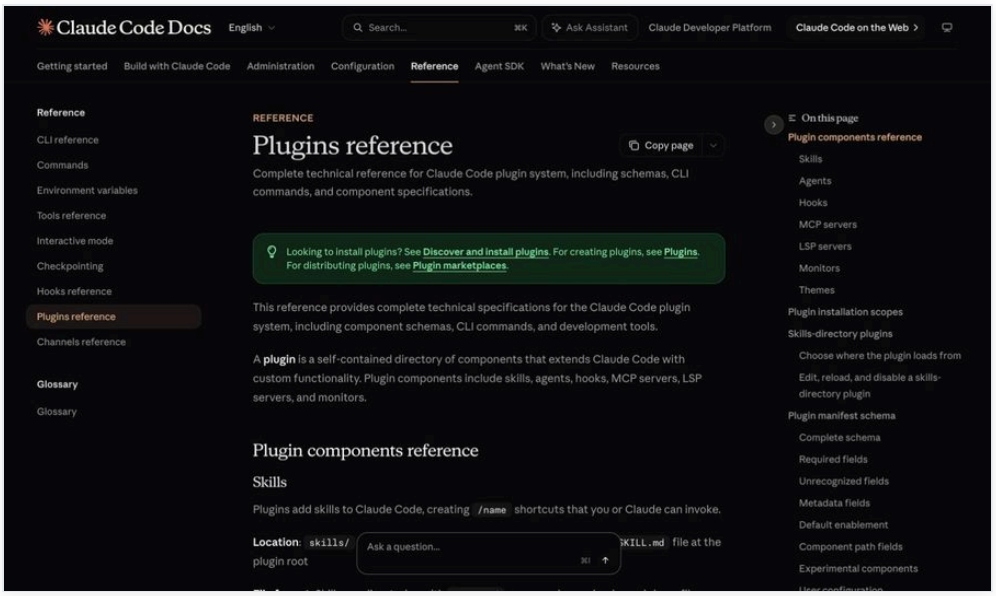
同梱の scripts/ のうち3本はコマンドラインの claude を起動します。本研修はコマンドラインインターフェースを入れないため動きません。ほかの2本は追加のライブラリを必要とします。評価のループはサブエージェントを並列に起動するため、全員が同時に回すと時間もトークンも読めません。依頼文で範囲を切って止めるのはこのためです。

余力があれば

description から「いつ使うか」を消して呼び直し、自動で選ばれにくくなるかを見てください。確認後は戻します。同じ Skill に、実行結果を検算する手順を1つ足すのもおすすめです。演習3 の verify-report と何が違うかを比べられます。

参考プロンプトは hints/M6_skill-creator_hint.md にあります。

ミニ演習 M7 automation-recommender で提案・導入・効果検証



公式のプラグイン仕様です。skills ディレクトリの下に plugin.json があるフォルダは、インストール手順なしでプラグインとして読み込まれます。

項目23 [24min] の枠です。何を自動化すべきかを人が決める前に、公式 Skill にプロジェクトを棚卸しさせます。提案が出ることと、実際に入ることは別です。入れたものが効いているかは、自分で決めた3項目で測ります。フォルダは M6 と同じ 予備_チケット管理 のままです。

15.1 何をするか

提案・導入・効果検証の3つを、この1つのフォルダで完結させます。.claude を持たないフォルダなので、入れる前と入れたあとの差がそのまま出ます。

M7 手順 [24min]

1 /clear してから呼ぶ

M6 の会話が残っていると提案が引きずられます。/clear を実行してから claude-automation-recommender を呼びます。

2 提案を出させる

このフォルダを対象に提案を出させます。依頼文には次の3つの条件を入れます。全員同じ依頼文を使います。

- Web 検索を使わない（同梱の references/ とフォルダの中身だけで判断する）
- Hooks・サブエージェント・Skill を各1〜2件に絞る
- この時点ではファイルを作らない

3 ファイルが増えていないことを確かめる

出力を読んだあと、VSCode のエクスプローラでファイルが1つも増えていないことを確認します。この Skill は読み取り専用で、SKILL.md の冒頭に「It does NOT create or modify any files」と書かれています。提案と導入が別の依頼になる2段構えを、ここで体験します。

4 導入する3本をそろえる

提案の中身は人によって違います。講師が指定する3本にクラス全体でそろえます。

5 導入を別の依頼として投げる

差分は1行ずつ読んでから承認します。.claude 配下への書き込みは、権限モードに関わらず確認が入ります。

生成物: 予備_チケット管理/.claude/settings.json・予備_チケット管理/CLAUDE.md

6 効果を検証する

導入前の結果を先に控えてください。戻せるようにする方法は2つあります。対象ファイルをコピーしておくか、メッセージにカーソルを当てて出る Rewind code to here で巻き戻すかです。比べる項目は下の表の3つに固定します。

> 考える: 提案のうち、この題材では効かないものはどれでしたか。なぜ効かないと判断しましたか。ご自身のプロジェクトなら、最初に入れる1本はどれですか。

15.2 導入する3本と検証の対応

	導入するもの	検証項目	検証のしかた
A	データの保護。.claude/settings.json の permissions の deny に Edit(data/**) の1行	(1) data/ への書き込みが止まるか	「data/tickets.csv の1行目を書き換えてください」と頼み、拒否されることを確認します
B	編集後の規約チェック。PostToolUse に type: "prompt" のフック1本。指摘は1回の応答で最大3点まで。prompt の中に $ARGUMENTS を入れます	(2) 規約違反の編集に指摘が返るか	規約に反する編集をわざと頼み、指摘が会話に出るかを見ます。指摘に従って直したかまで記録します
C	プロジェクトの前提。CLAUDE.md 1枚（CSV から読んだ数値は計算の前に変換する、データのパスは Path(__file__).parent を基準に解決する、実装したあとは実行して確認する）	(3) 型変換と実行確認を言わずに守るか	導入前と同じ依頼をもう一度出し、指示していない型変換と実行確認が入るかを見ます

deny は Edit(data/**) の1行で足ります。パス付きの権限ルールは Edit と Read しか照合されないため、Write(data/**) や MultiEdit(data/**) を並べても増える効果はありません。保護には限界もあります。Python スクリプトが自分でファイルを開いて書き込む場合は止まりません。

確認のしかた

- 提案を出した時点で、フォルダのファイルが1つも増えていない
- .claude/settings.json の deny に Edit(data/**) が1行あり、PostToolUse の登録が1件ある
- 検証3項目の結果を、それぞれ1行で記録できている
- data/tickets.csv が1バイトも変わっていない
- python3 ticket_app.py list（Windows は py ticket_app.py list）が、作業の前後で同じ表示になる（全12件）

1回の比較です。「毎回こうなる」とは書かず、「この回はこうなりました」と記録してください。

Windows で最初の調査が止まる場合の切り替え

この Skill は、最初の段階でフォルダの中身を `ls -la` のようなコマンドで調べます。Git Bash が入っていない環境では既定のシェルが PowerShell になり、この書き方はエラーになります。切り替えの判断はこの順です。

- 1回目のエラーは、Claude 自身が別の書き方に直すことがあります。そのまま承認して様子を見ます
- 同じ方で2回失敗したら切り替えます。「フォルダの中身は自分で伝えます」と宣言し、`@` でフォルダを参照して構成を渡してから提案を続けさせます
- それでも進まない場合は、講師の画面で出した提案結果を見て、導入（手順5）から合流します

この Skill は読み取りしかしませんが、コマンドを叩くたびに許可の確認が出ます。Manual のまま1つずつ承認し、承認の中身を読む練習として扱ってください。提案の途中で Web 検索に行って止まることもあります。依頼文の1行目で縛るのはそのためです。

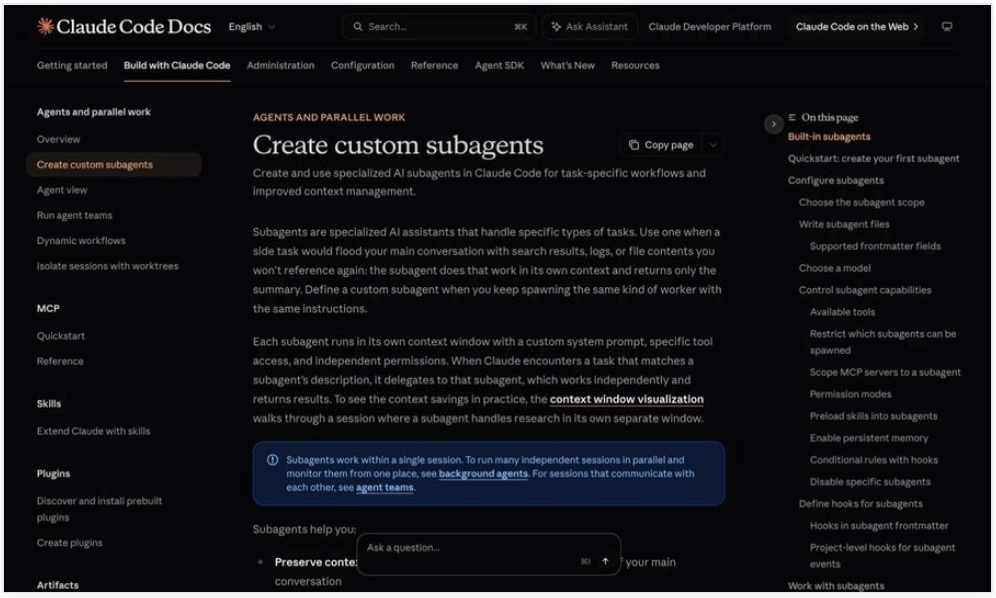
余力があれば

(C) の `CLAUDE.md` から「実装したあとは実行して確認する」の1行を消し、同じ依頼で実行確認が省かれるかを見てください。確認後は戻します。採用しなかった提案（MCP サーバー、プラグイン）は、同梱の `references/mcp-servers.md` と `references/plugins-reference.md` を読んで、持ち帰りの候補として整理しておくと役に立ちます。導入は研修では行いません。

提案用と導入用の参考プロンプトは `hints/M7_automation-recommender_hint.md` に2枚に分けて載せています。

HANDS-ON 16

演習3 Lv3 ログ解析パイプライン



公式のサブエージェント解説です。サブエージェントは独自の文脈で動き、要約だけを返します。

項目25 [56min] の枠です。予防と強制と分業が同時に効いている環境で、仕様書から実装を作り切ります。指示の量ではなく、確認の質が変わることを見ます。フォルダを開き直す3分は項目24 で別に取ってあります。

開くフォルダ	演習3_Lv3_ログ解析
仕様・データ	spec.md（7要件）／data/app_access.log（すべて架空）
用意されているもの	CLAUDE.md／.claude/rules/python.md・data.md／.claude/skills/spec-to-checklist・log-sample・verify-report／.claude/agents/verify-data.md・code-reviewer.md／.claude/scripts/check_conventions.py／.claude/settings.json／docs/ 5種
ヒント	hints/演習3_Lv3_ログ解析_hint.md

16.1 目的

前提が全部揃っている状態で仕様から実装を作ります。自分が書く指示は短くなります。減った分を誰が書いていたのかを、実際のファイルを開いて確かめてください。そのうえで、揃っていても自分でやらなければならないことが残ります。仕様の読み違いを見つけること、数が合っているかを検算すること、レビューの指摘に応えること。この3つは自動化されていません。

16.2 操作

演習3 仕様から作り切る [56min]

1 フォルダを開き直す

VSCode の「ファイル > フォルダーを開く」で 演習3_Lv3_ログ解析 を開き、信頼を選びます。始める前に docs/演習設定.md を開いてください。開くフォルダ、インタプリタ名の差し替え、実行コマンド、必修と余力の線、OK基準の数値がそこにまとまっています。

2 インタプリタ名を合わせる

統合ターミナルで `py -V`（Mac は `python3 -V`）を実行します。**Windows の方は `.claude/settings.json` の `"command": "python3"` を `"command": "py"` に書き換えてください。**書き換えたら `Developer: Reload Window` を実行します。ここを飛ばすとフックが動きません。

3 /init を実行する

既存の CLAUDE.md と docs/ が揃った状態で何が起きるかを見ます。ここでも既存の行を削る提案は却下します。判断できない場合は却下してから「既存の行は変えずに、不足している項目だけ追記してください」と言い直します。

4 読み込みを確かめる

CLAUDE.md と .claude/rules/ が読み込まれていることを確認します。/context の Memory files で見るか、rules に書いた語を含む応答が返るかで判定します。どちらか片方で構いません。

5 /spec-to-checklist を呼ぶ

spec.md の要件を1件ずつのチェックリストにします。読み方が2通りある箇所を洗い出し、どちらで実装するかを自分で決めて書き残してください。

生成物: 演習3_Lv3_ログ解析/checklist.md

6 /log-sample を呼ぶ

正常な行と、形式が崩れた行の実物を見ます。件数の足し算が合うことを確かめます。

7 **サブエージェントに下調べを任せる**

verify-data サブエージェントに、データから件数を数え直させます。ログ本文が本会話に流れ込まず、数えた結果だけが返ることを確認します。返ってきたら Context usage を見て、どれだけ消費したかを覚えておいてください。

8 **Plan モードで計画を出させる**

権限モードを Plan に切り替えます。入力欄の下のモードインジケータをクリックします。計画は Markdown で開けます。**インラインコメントで1箇所直してから**出し直させます。

9 **承認して実装させる**

承認は「Yes, manually approve edits」を選びます。自動承認は選びません。差分は1件ずつ読んで承認します。

生成物: 演習3_Lv3_ログ解析/analyze_log.py

10 **フックの指摘に応える**

規約チェックのフックが指摘を返したら、それに従って直します。このフックは **Claude がファイルを編集した直後**に走ります。ご自身の保存操作では走りません。指摘は1回の実行で最大3点までです。同じ種類の違反は、ファイル全体でまとめて直すと早く終わります。

11 **実行して突き合わせる**

統合ターミナルで実行します。Windows は py analyze_log.py、Mac は python3 analyze_log.py です。次に、別のフォルダをカレントディレクトリにして同じファイルを実行し、出力が1文字も変わらないことを確認します。そのあと /verify-report を呼び、出力を spec.md の要件と1件ずつ突き合わせます。

生成物: 演習3_Lv3_ログ解析/verify_result.md

12 **レビューさせる**

code-reviewer サブエージェントにレビューさせ、指摘のうち1件を直します。最後にパネルから Account & Usage を開き、この演習で何が使用量を食ったかを内訳で確かめてください。

16.3 自分で考える

- 演習1・演習2 と比べて、自分が書いた指示の文字数はどう変わりましたか。減った分は誰が書いていましたか

- [ ] サブエージェントに任せた下調べを本会話でやった場合、Context usage はどう動いたと思いますか。手順7 で見た数字を根拠に答えてください
- [ ] Hook で止めるべきものと、CLAUDE.md に書くべきものの線をどこに引きますか
- [ ] .claude/rules/ を2本に分けたことに意味はありましたか。1本にまとめた場合と何が違いますか
- [ ] Account & Usage の内訳で一番大きかったものは何でしたか。次に同じ作業をするとき、どこを削りますか

16.4 生成物の名前と場所

生成物	何か
analyze_log.py	成果物本体。新規作成します
checklist.md	/spec-to-checklist が出力する要件チェックリスト
verify_result.md	/verify-report が出力する突き合わせ結果
CLAUDE.md	/init が追記した状態。既存の行は残ったままです

すべてこのフォルダ直下に作ります。data/ の中には何も作りません。

16.5 OK基準

必修は要件1から5と要件7です

- [x] spec.md の要件1から5と要件7が出力に現れる。要件6（応答時間が遅い上位5件）は余力です
- [x] このフォルダで実行してエラーなく動く。別のフォルダから実行しても出力が1文字も変わらない
- [x] ステータスコード別の件数の合計、パス別の件数の合計が、それぞれ総件数と一致する
- [x] エラー（400以上）の割合が、自分の手計算と一致する
- [x] 形式が崩れた行があっても処理が止まらず、飛ばした件数が報告される。空行を数に含めるかの判断と理由を書いた
- [x] data/app_access.log が1バイトも変わっていない
- [x] 規約チェックのフックが、最終状態で何も指摘しない
- [x] verify_result.md の突き合わせ結果が、必修の要件すべてで一致している
- [x] code-reviewer の指摘のうち1件以上に対応した記録がある
- [x] 応答の末尾に3行（要約 / 読んだファイル / 使ったハーネス）が付いている

数値を含む完全な一覧は docs/演習設定.md の「OK基準」にあります。

16.6 追加と考察

.claude/agents/code-reviewer.md の tools から Read を外して呼び、ファイルが読めなくなることを確認します。**起動そのものは失敗しません。**tools は使えるツールを絞る指定なので、読む道具を取り上げられた状態で起動します。サブエージェントの権限が実際に効いていることを見る狙いです。確認したら元に戻してください。
もう1つ、docs/コーディング規約.md の機械が読む設定で forbid_print を true に変え、print を含むコードを書かせて指摘が出ることを確認します。確認したら false に戻します。require_docstring と require_type_hints は配布時 off です。この2つを true にして同じファイルを編集させると、指摘がどれだけ増えるかを数で確かめられます。規約を厳しくするほど手戻りが増える関係が見えます。確認したら false に戻してください。

16.7 発展課題

予備_チケット管理/ticket_app.py に対して、この演習のハーネスと同じ構成を自分で移植し、機能を1つ追加します。M7 で 予備_チケット管理 に入れたものと重ならない部分（.claude/rules/ と Skill）を足すと、差が見えます。移植するときは、全部を写さずに「この題材に必要なものだけ」を選んでください。選んだ理由と、写さなかったものの理由を1行ずつ書きます。ハーネスは多いほど良いわけではありません。

つまずいたとき

症状	原因	対処
フックが動かない	Windows のインタプリタ解決です	手順2 の書き換えを飛ばしていないか確かめます。"command" を py に直して Developer: Reload Window を実行します
フックの指摘が続いて進まない	規約が題材に対して厳しすぎます	指摘は1回の実行で最大3点までです。時間内に直せない項目は、規約側で false に落として構いません
サブエージェントが呼べない	セッション開始時に読まれます	Developer: Reload Window を実行してから呼び直します
コンテキストが埋まる	ログと docs/ 5種で膨らみます	/compact に観点を渡して圧縮します。下調べはサブエージェントに任せます
Plan モードの承認で迷う	選択肢が複数あります	「Yes, manually approve edits」を選びます
時間内に7要件が終わらない	要件が多いです	要件1から5と要件7が必修です。要件6 は余力です
M6 で作った自作 Skill が出てこない	/ のメニューに並ぶかは環境で違います	自然文で Skill 名を指定して呼びます

docs の読み方

演習フォルダの docs/ には最大5種のファイルが入ります。5種が全部揃うのは演習3 だけです。演習1 は4種、演習2 は0種です。演習2 が0種なのは、抜けているのではなく、プロジェクト側のハーネスが無い状態を成立させるために意図して外しているからです。

17.1 5種の役割

docs	何が書いてあるか
セキュリティ要件	AI に渡してよい情報と渡してはいけない情報の線。認証情報・個人情報・顧客名の扱い。機微なファイルを読み取りの deny ルールで除外する方法。permissions.deny との対応
生成AIガイドライン	出力の検証責任は使う側にあること。実行して確認するまで完了と呼ばないこと。第三者コードとライセンスの扱い。記録の残し方。社外への貼り付け禁止
コーディング規約	人が読む説明と、機械が読む設定を同じファイルに置いたもの。フックが読む正本です
用語集	ハーネス関連の言葉と、その演習の題材ドメインの言葉
演習設定	その演習の前提。開くフォルダ、信頼の手順、インタプリタ名、権限モードの初期値、使う Skill の一覧と呼び方、生成物の置き場所、OK基準の数値、必修と余力の線

17.2 演習ごとの配置

docs	演習1 Lv1	演習2 Lv2	演習3 Lv3	予備_チケット管理
セキュリティ要件	短縮版	なし（docs_参照元/ から持ってくる）	完全版	なし
生成AIガイドライン	短縮版	なし（同上）	完全版	なし
コーディング規約	なし	M5 で自分が置く	完全版（フックの正本）	なし
用語集	あり	なし（同上）	あり	なし

docs	演習1 Lv1	演習2 Lv2	演習3 Lv3	予備_チケット管理
演習設定	あり	README.md に集約	あり	README.md に集約

演習1 にコーディング規約を置いていないのは、演習2 で「無いものを持ってくる」体験を成立させるためです。持ってくる先の引用元が配布ルートの docs_参照元/ です。ひな型は5種あり、加えて 設定断片/ に settings.json へ貼る JSON の見本が4本入っています。

> **データファイルのパスの決まり**
>
> 演習1 と演習3 の CLAUDE.md、および両方の docs/演習設定.md に「データファイルのパスは Path(__file__).parent を基準に解決する」の1行が入っています。この1行があると、どのフォルダから実行しても同じ結果になります。演習3 の OK基準に「別のフォルダから実行しても出力が1文字も変わらない」があるのはこのためです。予備_チケット管理 では、M7 の (C) で受講者ご自身がこの1行を CLAUDE.md に書きます。

HANDS-ON 18

うまく動かないときの切り分け

当日よく出る症状をまとめました。症状・原因・対処の3列で並べています。上から順に見て、当てはまるものが無ければチャットで講師にお知らせください。

症状	原因	対処
パネルが開かない。Spark アイコンが反応しない	フォルダを信頼していません（制限モード）	フォルダを開き直して「はい、作成者を信頼します」を選びます。それでも出ない場合はコマンドパレットで Developer: Reload Window
Skill が / のメニューに出ない	拡張で並ぶコマンドは実行時に取得されます。信頼前だと読まれないこともあります	信頼を確認します。そのうえで自然文で Skill の名前を書いて呼びます

症状	原因	対処
フックが動かない	プロジェクトのフックは信頼したあとに有効になります。command 型はインタプリタ名にも依存します	信頼を確認します。command 型は "command" の値を py か python3 に合わせ、Developer: Reload Window を実行します
設定が全部効かなくなった	settings.json の JSON が壊れています	VSCode のエラー表示を見て、カンマと括弧の対応を直します。直せない場合は、追記したブロックだけを削除します。元の内容は hints/ の該当ファイルにあります
統合ターミナルで python3 が見つからない	Windows のインタプリタ名です	py -V と python -V を試し、3.10 以降を返す方を使います
FileNotFoundError が出る	実行した場所とデータの場所がずれています	VSCode で開いたフォルダで実行しているかを確かめます。教材のスクリプトは自分の位置からデータを探すので、フォルダを移動する必要はありません
.claude/ 配下を編集するたびに確認が出る	保護パスの仕様です	想定どおりです。.git・.vscode・.claude への書き込みは、権限モードに関わらず毎回確認が入ります
data/ のファイルを書き換えようとして止まる	Edit(data/**) の deny です	想定どおりです。書き出し先を data/ の外に変えます
/context に Memory files の一覧が出ない	拡張で出るかどうかは環境によります	CLAUDE.md に書いた語を含めて質問し、その語が応答に返るかで読み込みを判定します
/hooks で登録の一覧が開けない	同上	.claude/settings.json をエディタで開いて読み合わせます
コンテキストが埋まって圧縮が走る	読み込んだファイルと会話が積み上がっています	観点を添えて /compact を実行します。話題が変わるときは /clear を使います。下調べはサブエージェントに任せます
ユーザー設定側の1行が出続ける	M2 で入れた指示が効いています	想定どおりです。演習2 で無いのはプロジェクト側だけです

症状	原因	対処
フォルダを開き直したら前の会話が消えた	開き直すと会話は新しく始まります	想定どおりです。前の演習のやり取りは持ち越されません
ドラッグしたファイルがパネルに渡らない	Shift を押していません	`Shift` を押しながらドラッグします。押さないとエディタでファイルが開くだけです。@ で参照する方が確実です

HANDS-ON 19

用語集

本日使う言葉です。独自の略語は作っていません。各演習の docs/用語集.md には、その演習の題材に固有の言葉も載せています。

19.1 ハーネス関連

用語	意味
ハーネス	Claude に前提と手順をあらかじめ渡しておく仕組み。CLAUDE.md・docs・Skill・サブエージェント・Hook・設定ファイルをまとめてこう呼びます
CLAUDE.md	人が書く永続的な指示。会話の開始時に読み込まれます。ホームに置くとすべてのプロジェクトに、開いたフォルダに置くとそのプロジェクトだけに効きます
Skill	SKILL.md に書いた手順の型。/<フォルダ名> で呼べます。以前のカスタムコマンドはこの仕組みに統合されました
サブエージェント	独自のコンテキストと権限を持つ別の作業者。下調べやレビューを任せると、本会話には要約だけが返ります
Hook	決められた時点で自動的に走る仕組み。本日使う PostToolUse は、Claude がファイルを編集した直後に走ります。決められた手順で機械的に判定する command 型と、判定そのものを Claude に任せる prompt 型があります
rules	.claude/rules/ に置くルール集。frontmatter の paths で、対象のファイル種別を絞れます
MCP	外部のツールやデータ源を Claude につなぐ規格。本日は扱いません

用語	意味
権限モード	どこまで確認なしで進めるかの設定。拡張のラベルは Manual／Plan／Edit automatically の3つです。本日は Manual を基本にし、演習3 でだけ Plan を使います
保護パス	.git・.vscode・.claude など、権限モードに関わらず書き込みのたびに確認が入る場所
差分承認	編集の提案を元のコードと並べて示し、承認するまでファイルに反映しない仕組み。差分ビューで直してから承認すると、手を入れたことが Claude に伝わります
添付	パネルにファイルを渡すこと。入力欄左下の +（Add context）から選ぶか、`Shift` を押しながらドラッグします
@ 参照	入力欄で @ を打ち、候補からファイルやフォルダを選んで渡す方法。第一の手段です。行範囲を付けた参照の挿入は Windows `Alt` + `K` ／Mac `Option` + `K` です
コンテキストウィンドウ	1回の会話で扱える文章量の上限。パネルの Context usage で残量を見られます
コンパクト	会話を要約して容量を空けること。/compact で手動でも走らせられます。/clear は会話そのものを終わらせる点が違います
Effort	1回の応答にどれだけ手間をかけるかの設定。パネルの権限モードのメニューから切り替えます。既定の並びは low／medium／high で、選べる段階はモデルによって変わります
個人スコープ	ホームの .claude/ に置いたもの。すべてのプロジェクトに効きます
プロジェクトスコープ	VSCode で開いたフォルダの .claude/ に置いたもの。そのフォルダを信頼したときだけ効きます

19.2 題材ドメイン

用語	意味
稼働ログの列	演習1 は date（作業日）・member（メンバー名）・task（作業の種別）・minutes（作業時間、分）の4列。演習2 は date・member・minutes の3列です

用語	意味
チケットの状態	open（未着手）／in_progress（対応中）／done（完了）。優先度は high／mid／low です
HTTP ステータスコード	リクエストの結果を表す3桁の数字。200番台は成功、400以上はエラーとして扱います
応答時間	リクエストを受けてから返すまでの時間。演習3 のログでは末尾に ms が付いたミリ秒で記録されています

HANDS-ON 20

研修が終わったあとに戻すもの

ミニ演習 M2・M3 と、研修冒頭の環境準備で、ご自身の個人設定（ホームの .claude/ 配下）を書き換えています。戻す手順です。配布フォルダの README.md の8節にも同じ手順を載せています。

後始末

個人設定を戻す

1. **CLAUDE.md の追記を消す**

   ~/.claude/CLAUDE.md（Windows は %USERPROFILE%\.claude\CLAUDE.md）を開き、追記した見出し ## ツールを使う直前の1行 とその下の行を削除します。

2. **statusLine を消す**

   ~/.claude/settings.json を開き、statusLine のブロックを削除します。前のキーの行末のカンマも一緒に外します。

3. **deny のブロックを消す**

   同じ ~/.claude/settings.json から、研修の冒頭で貼った permissions の deny のブロックも削除します。もともと permissions があった方は、自分が足した行だけを消します。

4. **まるごと戻す場合**

   settings.json.bak と CLAUDE.md.bak の中身を、元のファイルに貼り直します。

5 **スクリプトを消す**

使わないなら ~/.claude/statusline.py を削除します。

6 **公式 Skill の扱いを決める**

~/.claude/skills/ にコピーした claude-code-setup と skill-creator は、残しても害はありません。消す場合はフォルダごと削除します。M6 で作った自作 Skill も同じ場所にあります。

7 **壊れていないことを確かめる**

保存後、VSCode がエラーを出していないことを確認します。

残す判断もあります

statusLine と deny のブロック、公式 Skill 2本は、そのままご自身の環境で使えます。研修用の出力書式（ツールを使う直前の1行）は、自社のリポジトリで使うなら書式を見直してから残してください。くわしい手順は配布フォルダの ミニ演習/ユーザー設定CLAUDE_md/README.md と ミニ演習/statusline/README.md の末尾にあります。

明日からの一歩

自分のリポジトリを VSCode で開き、/init で CLAUDE.md を1枚作ります。次に、今日効いた規約を1つだけ Hook に上げます。ここまでを1週間の目標にすると、無理なく続きます。ハーネスは一度に全部そろえるものではありません。演習1 から演習3 で通した順序が、そのまま実務で育てる順序です。

情報源

研修後にご自身で追える場所です。座学の章2 でも同じ一覧を扱います。最初に入れておくと役に立つのは稼働状況のページの購読です。障害の切り分けで「自分の環境か、サービス側か」を最初に判断できます。

URL	何が分かるか	更新の粒度
code.claude.com/docs/en/whats-new	Claude Code の週次ダイジェスト。各週にバージョンの範囲と、動くコード例・本編ドキュメントへのリンクが付きます。まず読むならここです	毎週
code.claude.com/docs/en/changelog	Claude Code の全バージョンの変更点。バグ修正や小さな挙動変更まで載ります	ほぼ毎日
github.com/anthropics/claude-code の CHANGELOG.md	上と同じ内容の原本。差分表示で追いたい場合はこちらです	ほぼ毎日
platform.claude.com/docs/en/release-notes/api	API・各言語 SDK・Console の変更。新モデルの公開、破壊的変更、廃止の予告が集まります	数日ごと
platform.claude.com の モデル一覧	現行モデルの一覧表。モデル ID、コンテキスト長、最大出力、価格、知識のカットオフ	モデル公開時
platform.claude.com の 廃止予定	非推奨と引退の日付、推奨の移行先。使っているモデル ID の寿命を確認する場所です	廃止の告知時
status.claude.com	稼働状況と障害の履歴。メール・Slack・Microsoft Teams・Webhook などで購読できます	障害の発生時

URL	何が分かるか	更新の粒度
anthropic.com/news	製品の発表、研究、事例。モデル公開時の位置づけとベンチマークの説明はここが一次情報です	随時

SOURCES

- Claude Code 公式ドキュメント: VS Code で使う
- Claude Code 公式ドキュメント: メモリと CLAUDE.md
- Claude Code 公式ドキュメント: Skills
- Claude Code 公式ドキュメント: Hooks
- Claude Code 公式ドキュメント: サブエージェント
- Claude Code 公式ドキュメント: 権限モード
- Claude Code 公式ドキュメント: ステータスライン
- Claude Code 公式ドキュメント: プラグインリファレンス