ORIENTATION

生成AIプログラミング入門編 オリエンテーション

VSCode × Claude Code で、作る側と直す側を1日で往復する240分

本研修は、生成AIにプログラミングを任せる開発を初めて体験する方のための240分です。前半の課題Aでは、雛形も設定ファイルもない空のフォルダから、自然言語の指示だけで一覧・検索・詳細のある業務アプリを1本組み上げます。後半の課題Bでは、他人が書いた Java・Spring Boot のプロジェクトを読み解き、不具合を Issue に起票してから2件直し、テストを通します。動いた瞬間の手応えと、勢いだけで作ったものが抱える穴を、同じ日に自分の手で確かめる構成です。

形式 会場開催 4時間 [240min]　**対象** 生成AIプログラミング未経験から初心者 20名

環境 VSCode × Claude Code（Amazon Bedrock 経由・Claude Sonnet 4.6）　**講師** 安田 光喜（Givery）

VSCode　Claude Code　Amazon Bedrock

Claude Sonnet 4.6　Spring Boot　Java 17

SECTION 01

本研修のゴール

到達点は「AIを触ってみた」ではありません。240分の終わりに、**自分の指示で作らせ、出てきたものを実行と数字で確かめ、直すところまで一人で回せる状態**になっていることです。プログラミングの文法知識は前提にしません。判断の材料になるのは、画面が動くかどうかと、テストが通るかどうかです。到達したかどうかは気分ではなく、手元に残るファイルで判定します。

ゴール 1 ・ 課題A 前半

空のフォルダから業務アプリを1本作れる

設定ファイルが1つも無い状態から始めます。用意されているのは備品50件・カテゴリ5種の `dummy_data.csv` と、画面要件を書いたお題シートだけです。ここから一覧・検索・詳細ドリルダウンの3つの動きを持つ `kadaiA/index.html` を、自然言語の指示だけで組み上げます。ブラウザで開くだけで動く形が完成の目安です。

1回で全部を頼まず、3〜4回に分けて指示を出します。途中で自分の意図を足せることを体で覚えるためです。

ゴール 2 ・ 課題A 後半

設定を置く前と後で、AIの応答の差を説明できる

作り終えたところで、配布物に同梱してある設定一式（ `CLAUDE.md` と `.claude/` ）を自分の手で配置します。同じAIが、同じ作業をしているのに、応答の末尾に実行サマリーを出すようになり、 `/selfcheck` という1つのコマンドで点検から修正案の作成、再点検までを回すようになります。

回した結果は `.work/` に中間データが4点、 `kadaiA/REPORT.md` に報告書として残ります。前後の差は `kadaiA/COMPARE.md` に自分の言葉で書き出します。

ゴール 3 ・ 課題B

他人のコードの不具合を2件、根拠を書いてから直せる

初見の Java・Spring Boot プロジェクトが題材です。中には意図的に不具合が仕込まれています。まず Plan モードで壁打ちして不具合を見つけ、 `ISSUE_00N_内容がわかる名前.md` の形で起票し、その Issue を根拠に修正します。

直ったかどうかは、ブラウザで画面を見て確かめ、 `./mvnw test` が BUILD SUCCESS になることで確定させます。この2件到達が課題Bの合格ラインです。3件目は発展課題として扱います。

到達したかどうかの判定

できたつもりで終わらせないために、ゴールごとに証拠となるファイルと判定方法を決めてあります。演習中はこの表の右2列を見ながら、自分がどこまで来ているかを確認してください。

ゴール	手元に残る証拠	判定の仕方
1 作れる	`kadaiA/index.html`	ブラウザで開くと備品が50件並び、備品名の一部を入力すると件数が絞り込まれ、1件を選ぶと詳細が出て一覧に戻れる
2 確かめられる	`kadaiA/REPORT.md` ／ `kadaiA/COMPARE.md` ／ `.work/` の4ファイル	`/selfcheck` が完走して報告書が出ている。設定を置く前と後で応答がどう変わったかを、自分の言葉で2点以上書けている
3 直せる	`kadaiB/issues/` の起票ファイル ／ 修正後のソース	不具合2件について Issue が書けている。ブラウザで直っていることを確認できる。`./mvnw test` が BUILD SUCCESS になる

本研修のスタンス

「AIがすごい」で終わらせません。課題Aの前半でAI開発の速さを体験として掴み、後半で「勢いだけで作ったものは何を確かめたか誰も説明できない」ところまで自分の手で確かめていただきます。**動くことと、正しいことは別**です。この線引きを持ち帰っていただくのが本研修の狙いです。

一 今日のゴールに含めないこと

240分でできることには限りがあります。次の4つは今日の範囲外です。範囲外だと先に言っておくほうが、演習中の判断が速くなります。

- **Java や JavaScript の文法を学ぶこと**。課題Bのコードは読みますが、読み解く作業自体もAIに手伝わせます
- **本番環境へのデプロイ**。課題Aの成果物はブラウザでファイルを開けば動く形、課題Bはローカル起動までです
- **チーム全体への展開設計**。設定ファイルをチームで共有する話は、まとめの20分で入り口だけ扱います
- **Git と GitHub の操作**。本研修では使いません。Issue はローカルの Markdown ファイルとして書きます

SECTION 02

アジェンダ

240分のうち140分、全体の58%を演習に充てます。座学は演習の前提を揃えるための最小限です。前半の課題Aは「作る側」、後半の課題Bは「直す側」に軸足を置いています。同じ道具を、向きの違う2つの仕事で使うのが今日の構成です。

図1 240分の時間配分。帯の幅がそのまま時間の比率です。座学2本で60分、演習2本で140分、締めくくりで40分という配分になっています。

No	セッション	時間	やること
S01	オリエンと支援の段階遷移	[20min]	本日のゴール共有。全員でVSCodeを開き、Claude Code のパネルから1文送って応答が返ることを確認します。詰まりをここで全部拾います。そのうえで、コーディング支援の4段階のどこを今日扱うのかを位置づけます
S02	座学とデモ	[40min]	AIコーディングの現在地を数字で確認し、AIが1往復のあいだに何をしているか（読む・計画する・書く・実行する・報告する）をログ画面で実演します。続けてVSCode上の基本操作と、Plan モードと Agent モードの使い分けを扱います
S03	**ハンズオン 課題A**	**[70min]**	設定が1つも無い状態で業務アプリを作り、気になった点を自分の言葉で書き出し、そこで初めて設定一式を配置します。同じAIの応答がどう変わるかを確かめ、最後に `/selfcheck` でパイプラインを1周回します
S04	**ハンズオン 課題Bと発展**	**[70min]**	Java・Spring Boot のプロジェクトを読み解き、Plan モードで不具合を2件見つけ、Issue に起票してから直します。テストを通したあと、自分用の Skill 作成と脅威モデリングまで進みます
—	まとめとAI設計デモ	[20min]	課題Aで自分が書き出した懸念に、設定ファイルがどう答えたのかを振り返ります。コンテキストエンジニアリングの考え方と、コンテキストからハーネス、ループへとつながる3段構えを、講師端末の実演で見ていただきます
S05	振り返りと質疑応答	[20min]	4つの問いに1問ずつ答える形で、今日できるようになったことを言語化します。記入6分、共有6分、質疑8分の配分です

検算：20 + 40 + 70 + 70 + 20 + 20 = 240。合計 [240min] です。

演習14ステップの内訳

課題Aと課題Bは、それぞれ7つのステップに分かれています。1ステップあたり5分から20分です。手順の本体は配布フォルダの `exercises/` にあり、どのファイルにも 目的／操作／自分で考える／生成物の名前と場所／OK基準／追加と考察／発展課題 の順で書いてあります。進みが速い方は、各ステップの発展課題に進んでください。

#	課題A（70分）	#	課題B（70分）
A-0	設定が1つも無い状態を確認する（5分）	B-0	課題Bを起動して現状を掴む（8分）
A-1	素の状態で備品管理アプリを作る（15分）	B-1	自動化の提案を出させる（8分）
A-2	気になる点を自分の言葉で書き出す（8分）	B-2	Plan モードで不具合を2件見つける（12分）
A-3	ハーネスを配置する（5分）	B-3	Issue を起票する（7分）
A-4	実行サマリーの出方を確認する（7分）	B-4	修正して、動かして、テストを通す（20分）
A-5	`/selfcheck` でパイプラインを回す（20分）	B-5	自分の Skill を作る（8分）
A-6	設定の有無を比較する（10分）	B-6	脅威モデリングを回す（7分）

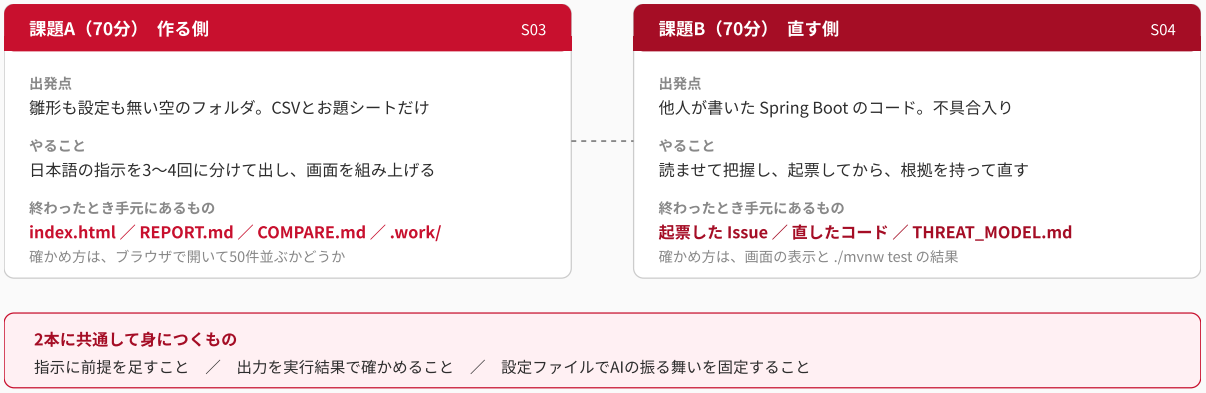

図2 課題Aと課題Bの位置づけ。出発点も、やることも逆向きです。共通しているのは、指示の出し方と、出てきたものの確かめ方の2点で、この2点が明日から使える部分になります。

なぜ作る側と直す側を1日で往復するのか

実際の業務で新規開発だけを担当し続けることは、まずありません。既存のコードを読み、原因を切り分け、影響範囲を見てから直す時間のほうが長いはずです。AIに任せられる範囲も、この2つでは大きく違います。課題Aでは指示するだけで進みますが、課題Bでは**どこを直すかを人が決めてからでないと、AIは的外れな修正を始めます**。この差を体感していただくために、あえて性質の違う課題を2本並べています。

SECTION 03

コーディング支援の段階遷移

AIによるコーディング支援は、書いている行の続きを埋める補完から始まり、チャットでの相談を経て、いまはタスクごと任せる段階に来ています。段階が上がるほど任せられる範囲は広がりますが、そのぶん**出てきたものを確かめる作業が人の側に集まります**。今日どこを扱うのかを先に決めておかないと、演習中に「AIが全部やってくれるはず」と「AIは信用できない」のあいだで判断がぶれます。本研修が越えるのは、段階2（対話）から段階3（委任）への一歩です。

図3　コーディング支援の段階遷移。本研修は段階2（対話）から段階3（委任）へ渡る一歩を扱います。課題A の前半が段階2、ハーネスを置いた後半と課題B が段階3です。段階4（協働）は、まとめの20分で入り口を実演で見ていただくところまでです。

段階が上がると何が変わるか

段階ごとに、人の仕事とAIの仕事の境目が動きます。あわせて、うまくいかないときの症状も変わります。段階3でよく起きるのは「動くけれど、何を根拠にそう書いたのかが誰にも説明できない」という状態です。今日の課題Aは、まさにこの状態を意図的に作ってから対処する構成になっています。

段階	人がやること	AIがやること	うまくいかないときの症状
段階1 補完	設計も実装も ほぼ全部。 提案を採るか どうかだけ決める	いま書いている行の 続きを提案する	提案が的外れで、無視する回数が増える。 実害は小さい
段階2 対話	質問を組み立て、 返ってきたコードを 貼り、動くかどうか 確かめる	聞かれた範囲の 関数やファイルを 書いて返す	断片が集まるが、全体としてつながらない。 貼り付ける手間が残る
段階3 **委任**	何を作るかを言葉で 伝え、出てきた ものを実行して 確かめる	ファイルを作り、 書き換え、 コマンドを実行し、 結果を報告する	速く動くものができる。ただし入力の検証や 例外処理が抜けたまま完成扱いになる
段階4 協働	チームの決まりを ファイルに 書き、全員が 同じ枠で AI を 動かせるようにする	共有された枠の中で 点検と修正を 繰り返し、 中間データと 報告書を残す	枠が曖昧だと、AIが際限なく作業を広げる。 枠の設計そのものが仕事になる

いまの現在地を数字で見る

Stack Overflow が2025年に実施した開発者調査（回答者およそ49,000人、AIに関する設問への回答は33,662人）の数字です。導入は進み、信頼は下がっているという、ねじれた状態が読み取れます。

84%

開発プロセスでAIツールを使っている、または使う予定がある

前年の76%から上昇。プロの開発者に限ると51%が毎日使用

45.7%

AIの出力の正確さを信頼していない

「強く信頼する」と答えたのは3.1%にとどまる

66%

「ほぼ正しいが、微妙に違う」出力に手を焼いている

回答者が挙げた不満の第1位。デバッグに時間がかかるが45.2%

この数字の読み方

使う人は増え、信頼は下がっています。矛盾しているようですが、実際に使い込んだ人ほど「ほぼ正しいが微妙に違う」出力に何度も当たった結果だと考えると筋が通ります。**今日の演習で目指すのは、AIを信頼することではなく、信頼しなくても回せる手順を持つこと**です。課題Aの後半で配る設定ファイルは、そのための道具です。

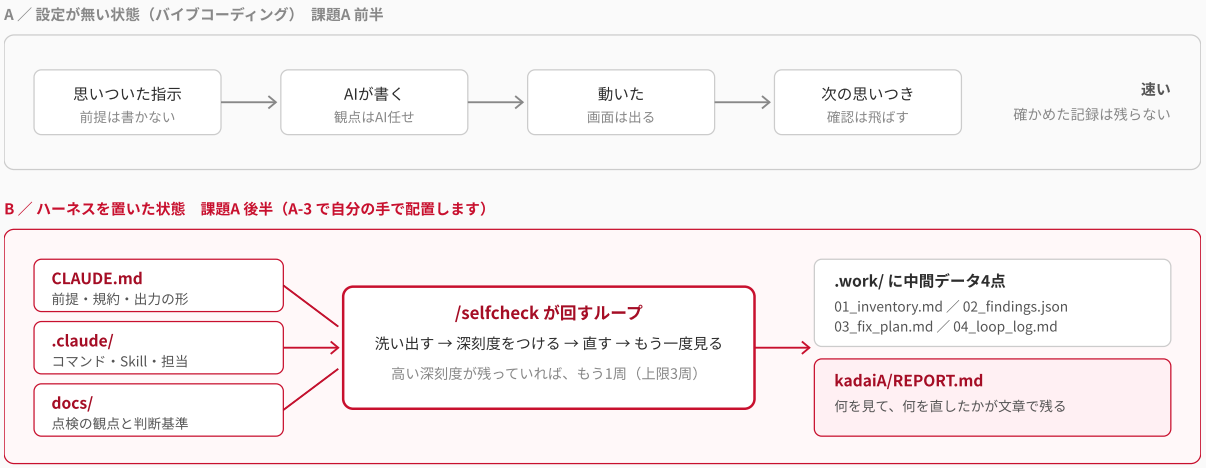

図4 バイブコーディングとハーネスの関係。上下で変わるのはAIの性能ではなく、AIに渡している前提と手順です。同じモデルに同じ作業をさせても、渡す情報を変えると、残るものが変わります。ハーネスという言葉は「AIを毎回同じ手順・同じ観点で動かすための設定一式」を指します。

用語の整理

バイブコーディングは、細かい設計を決めずに、思いついた指示を重ねて作っていく進め方です。速さは本物で、今日の課題A前半でその速さを味わっていただきます。悪者ではありません。ただ、1人で試すぶんには問題にならない性質が、チームで使うと途端に問題になります。誰も何を確かめたか説明できないからです。

ハーネスは、その穴を埋めるための設定一式です。本研修では `CLAUDE.md`（前提と規約）、`.claude/`（コマンド・Skill・サブエージェント）、`docs/`（点検の観点4本）の3点セットを指します。演習A-3で、配布フォルダの `_harness_kit/step1_kadaiA/` から、この3つを自分の手でコピーして配置します。3つとも置かないと、A-5 の `/selfcheck` が観点を読めずに結果が薄くなります。

SOURCES

Stack Overflow Developer Survey 2025 — AI（採用率84%・不信45.7%・「ほぼ正しいが微妙に違う」66%）
Anthropic Engineering — Best practices for Claude Code
Anthropic Engineering — Effective context engineering for AI agents
Anthropic Engineering — Effective harnesses for long-running agents
Claude Code Docs — Overview

SECTION 04

本研修の環境

エディタは VSCode、AIは Claude Code です。モデルへの接続は Amazon Bedrock 経由で、使用するモデルは Claude Sonnet 4.6 です。個人でのアカウント契約は不要で、事前セットアップガイドの手順で接続が済んだ状態から始めます。研修中にモデルを切り替えることはしません。

構成要素	役割	今日の使い方
VSCode	作業の起点になるエディタ。ファイルを見る、書き換える、コマンドを実行する場所です	開くフォルダは最初から最後まで `handson` です。`kadaiA` や `kadaiB` を単体で開くと、途中で配置する設定ファイルが読み込まれません
Claude Code	日本語の指示を受けて、ファイルの作成・編集からコマンド実行までを一続きに進めるコーディングエージェントです	VSCode 左のアクティビティバーにあるアイコンからパネルを開いて使います。作業対象は `@kadaiA/` のようにパスで指定して切り替えます
Amazon Bedrock	AWS 上で Claude のモデルを呼び出すための入口です	接続設定は事前セットアップで完了しています。研修中に設定を触る場面はありません
Claude Sonnet 4.6	本研修で使用するモデルです	接続先のモデルは事務局が指定します。受講者側での切り替えは行いません
Java 17 / Spring Boot 3.4	課題Bの題材プロジェクトの技術スタックです	画面は Thymeleaf、データは H2 のインメモリDBです。ビルドは同梱の Maven Wrapper（`./mvnw`）で行うため、Maven のインストールは不要です

Claude Code の開き方

開き方は1つに統一します。VSCode 左のアクティビティバーにある Claude Code のアイコンを押すと、パネルが開きます。ここに日本語で指示を書きます。開始5分でつまずく原因の多くは、起動方法が資料ごとに違うことなので、本研修ではこの方法だけを使います。

VSCode で handson を開く → **アクティビティバーの Claude Code アイコンを押す** →

パネルに日本語で1文入力 → **応答が返れば準備完了**

Tips ／ 設定ファイルは開いた時点で読み込まれます

`CLAUDE.md` と `.claude/` は、パネルを開いた時点で読み込まれます。演習A-3 でこれらを配置したあとは、**パネルをいったん閉じて開き直してください**。開き直さないと、置いたはずの設定が効かず、実行サマリーも出ません。演習中に「何も変わらない」と感じたら、まずここを疑ってください。

ターミナルで `claude` と入力しても起動できますが、研修中はパネルの操作で統一します。ご自身の環境ではどちらでもかまいません。

Plan モードと Agent モード

同じ指示でも、モードによって起きることが変わります。詳しい使い分けは S02 の40分で扱いますが、位置づけだけ先に共有します。課題Bの不具合探しでは Plan モードを使います。いきなり書き換えさせると、どこが原因だったのかが分からなくなるためです。

Plan モード

調べて、計画を出す

- コードは書き換えず、何をするつもりかを先に出します
- 提示された計画を読んで、承認するか、指摘してから承認します
- 原因の切り分けや、影響範囲を先に知りたいときに向いています

課題Bの B-2（不具合を2件見つける）で使います

Agent モード

その場で調べて、書き換える

- 計画を挟まず、ファイルの作成・編集・実行まで進みます
- 差分が提示されるので、受け入れるかどうかをその場で判断します
- やることが決まっている作業を任せるときに向いています

課題Aの A-1（アプリを作る）で使います

モードはプロンプト入力欄の下にあるモード表示から切り替えられます。`Shift` + `Tab` でも切り替わります。S02 で全員に一度押していただき、表示が変わることを確認します。

冒頭8分でそろえること

S01 の環境確認で、全員の VSCode 上で Claude Code のパネルが開き、日本語の1文に応答が返る状態をそろえてから先へ進みます。`こんにちは。今日の作業フォルダの中身を教えてください` と1文だけ送ってみてください。返ってこない方はその場で挙手をお願いします。演習が始まってからの環境トラブルは全員の進行に響くため、詰まりは冒頭8分に集めます。

ー パネルが開かない、応答が返らないときに見るところ

- **拡張機能が入っているか**：`Ctrl` + `Shift` + `X`（Mac は `Cmd` + `Shift` + `X`）で拡張機能の一覧を開き、Claude Code が有効になっているかを確認します
- **VSCode を再読み込みする**：`Ctrl` + `Shift` + `P`（Mac は `Cmd` + `Shift` + `P`）でコマンドパレットを開き、Developer: Reload Window を実行します
- **開いているフォルダを確認する**：`handson` ではなく親フォルダや別フォルダを開いていないかを見ます
- **それでも返らない場合**：挙手してください。接続設定側の問題であれば、講師と事務局で対応します

SOURCES

Claude Code Docs — Use Claude Code in VS Code（パネルの開き方・差分の受け入れ）
Claude Code Docs — Permission modes（Plan モードほか）
Claude Code Docs — Claude Code on Amazon Bedrock（推論プロファイルIDの指定）
Claude Code Docs — Settings（settings.json で指定できること）
Claude Code Docs — Memory（CLAUDE.md の読み込み）

SECTION 05

進め方とルール

演習は各自のペースで個別に進めます。ペアワークもグループ討議も発表もありません。全員が同じ速さで進む必要もありません。同じ指示を出しても出力は毎回変わるので、隣の画面と違っていても問題ないと考えてください。

進め方

個人演習、詰まったら hints、それでも駄目なら挙手

- 手順の本体は `exercises/` の各ファイルにあります。順番に読んで進めてください
- 詰まったら `hints/` の参考プロンプトと対処を見ます。**先に開かないでください**。自分で考える時間がなくなります
- hints を見ても動かないときは挙手してください。席までうかがいます
- 早く終わった方は `exercises/challenges/` の発展課題5本に進んでください
- 気づいたことは `handson/memo.md` に書き残します。`## A-0` から `## B-6` までの見出しが最初から切ってあるので、該当する見出しの下に書き足してください。S05 の振り返りでそのまま使います

ルール

研修中の約束事

- Git と GitHub は使いません。Issue はローカルの Markdown ファイルとして書きます
- 扱うデータは配布した架空データだけです。業務データ、顧客情報、実在の社名・氏名・メールアドレス・電話番号は入力しないでください
- `kadaiA/dummy_data.csv` と `kadaiB/src/main/resources/data.sql` は書き換えません
- `_harness_kit/` は講師の合図があるまで開きません。中身を先に読むと演習Aの前半が成立しません
- AIの出力は、実行して確かめてから採用します。読んだだけで採用しません

詰まったときの手順

手が止まったときに何をするかを、先に決めておきます。演習中は、この順番で進めてください。1つ目と2つ目で解決することがほとんどです。

1. 指示に前提を足して、もう一度送る → **2. hints/ の該当ファイルを開く** →

3. exercises/ の OK基準に戻って現在地を確認 → **4. 挙手する**

うまくいかなかった指示ほど価値があります

AIの出力は同じ指示でも毎回ばらつきます。思ったとおりに出ないのは普通のことなので、作り直すのではなく**「ここをこう変えてください」と差分で伝え直す**ほうが速く直ります。生成が途中で止まったときは「続けてください」で再開できます。S05 の振り返りでは、うまくいった指示だけでなく、うまくいかなかった指示も歓迎します。原因を一緒に見ます。

同じ指示でも出力が変わるのはなぜか

AIは、次に来る言葉を確率で選びながら文章とコードを書いています。選び方に幅があるため、同じ入力でも出てくるものが毎回少しずつ変わります。これは不具合ではなく、仕様です。

だからこそ、出力を安定させたいときは「AIに何度も同じことを言う」のではなく、**渡す前提のほうを固定します**。今日の課題Aで配る `CLAUDE.md` は、まさにこれを紙一枚で行う仕組みです。プロジェクトの前提、守ってほしい規約、応答の最後に必ず出す項目を書いておくと、指示の言い回しが多少変わっても、返ってくるものの形がそろいます。

SECTION 06

配布物の確認

事前に配布した `genai-nyumon-handson.zip` を展開して使います。展開すると `genai-nyumon-handson/` というフォルダができ、その中に `handson/` が入っています。二重の階層になるので、**VSCode で開くのは内側の `handson` です。**ここを間違えると、演習の途中で配置する設定ファイルが読み込まれません。

```
# 展開後の階層
genai-nyumon-handson.zip
  └ genai-nyumon-handson/
      └ handson/  ← VSCode で開くのはここ
```

handson フォルダの中身

配布直後の状態です。演習を進めると、この中にファイルが増えていきます。

```
handson/                      ← VSCode で開くのはここ
├── README.md                   この配布物の案内。最初に一読してください
├── memo.md                     気づいたことを書き残すファイル。## A-0 〜 ## B-6 の見出しだけ用意し
てあります
├── exercises/                  演習の手順書。README.md に一覧表があります
│   └── challenges/              発展課題5本
├── hints/                      詰まったときの参考プロンプト。詰まってから開きます
├── _harness_kit/               設定ファイルの待機場所。合図があるまで開きません
│   ├── step1_kadaiA/            演習 A-3 で handson 直下へコピーする一式
│   │   ├── CLAUDE.md             A-3 でコピー。プロジェクトの前提
│   │   ├── .claude/              A-3 でコピー。設定・コマンド・Skill・サブエージェント
│   │   ├── docs/                 A-3 でコピー。/selfcheck が読む観点ファイル4本
│   │   └── AGENTS.md ほか        他のAIツール向けの見本。コピーしません
│   └── step2_kadaiB/            演習 B-0 で追加する一式
├── kadaiA/                     課題A。備品50件の dummy_data.csv とお題シート
└── kadaiB/                     課題B。Spring Boot のプロジェクト。issues/ に起票の型があります
```

配布直後の `handson` 直下は、この6つだけです。`CLAUDE.md` ・ `.claude` ・ `docs` はここにありません。演習 A-3 で `_harness_kit/step1_kadaiA/` から自分の手で置きます。

CLAUDE.md ・ .claude ・ docs が無いのは、不備ではありません

配布直後の `handson` 直下には `CLAUDE.md` も `.claude` も `docs` もありません。**これは演習の出発点です。**設定が1つも無い状態でまず作ってみて、そのあと自分の手で `_harness_kit/step1_kadaiA/` から配置します。同じAIの応答が前後でどう変わるかを、自分の目で確かめていただくための順番です。「配布物が壊れている」と思わずに進めてください。

演習を進めると増えていくもの

増えるもの	いつ	何か
`kadaiA/index.html`	A-1	自分の指示で作るアプリ本体
`CLAUDE.md`	A-3	Claude Code が起動時に自動で読む共有文脈
`.claude/`	A-3 と B-0	設定・スラッシュコマンド・Skill・サブエージェント
`docs/` の4ファイル	A-3	コーディング規約・生成物チェックリスト・セキュリティチェックリスト・セルフチェックの観点。`/selfcheck` が点検の入力として読みます
`.work/` の4ファイル	A-5	`/selfcheck` の中間データ。消さずに残してください
`kadaiA/REPORT.md`	A-5	`/selfcheck` が出す点検の報告書
`kadaiA/COMPARE.md`	A-6	設定の有無で何が変わったかを比べた記録
`kadaiB/issues/ISSUE_002_...md`	B-3	自分で起票する Issue
`kadaiB/THREAT_MODEL.md`	B-6	脅威モデリングの出力

開始前チェック

次の4点が確認できれば準備完了です。1つでも欠けている方は、S01 の環境確認の時間に挙手してください。

- [ ] VSCode で `handson` フォルダが開いている（親フォルダではありません）。左のエクスプローラの一番上に `HANDSON` と出ていれば正しい階層です
- [ ] アクティビティバーの Claude Code アイコンからパネルが開き、日本語の1文に応答が返る
- [ ] エクスプローラに `kadaiA/dummy_data.csv` が見えている
- [ ] ターミナルで `java -version` を実行すると 17 と表示される（課題Bで使います）

事前セットアップの段階で `kadaiB` フォルダの `./mvnw -q -DskipTests package` を1回実行しておくと、課題Bの起動待ちが数分から数十秒に縮みます。まだの方は S03 の演習中、手が空いたタイミングで実行しておいてください。

SECTION 07

今日持ち帰るもの

研修が終わったあと、手元には作ったファイルと、自分で書いた記録が残ります。`handson` フォルダはそのまま持ち帰っていただいてかまいません。設定ファイルと演習の手順書も入ったままなので、明日以降ご自身の環境で読み返せます。

残るもの	性質	あとで何に使えるか
`kadaiA/index.html`	自分が作らせた成果物	指示だけでここまで出せるという基準になります。次に何かを作らせるとき、どのくらいの粒度で頼めばよいかの目安になります
`kadaiA/REPORT.md` ／ `.work/` の4ファイル	AIが点検した記録	AIに点検させると何がどこまで見えるのか、逆に何が見えないのかが具体的に分かります。中間データが残るという発想そのものが持ち帰りの中身です
`kadaiA/COMPARE.md` ／ `memo.md`	自分の言葉で書いた記録	職場で「設定ファイルを置くと何が変わるのか」を説明するときの材料になります。他人の言葉より、自分が書いた1行のほうが説明に使えます
`CLAUDE.md` ／ `.claude/`	そのまま流用できる型	プロジェクトの前提、点検の観点、応答の最後に出す項目の書き方の見本です。中身を自分の案件に書き換えれば、そのまま使えます
`kadaiB/issues/` の起票ファイル	書き方の型	現象・期待する動作・発生箇所・再現手順・修正方針の5項目です。AIに修正を頼むときの指示文としても、そのまま機能します

明日から試すこと

研修で覚えたことは、翌週には半分忘れます。忘れる前に手を動かせるよう、規模の小さい順に3つ挙げます。S05 の振り返りでは、この中から1つ選んで書いていただきます。

今週中に

既存コードを読ませてみる

自分の担当しているコードのうち、他人が書いた部分をAIに読ませ、何をしているかを日本語で説明させます。今日の課題B の B-0 と同じことを、自分の題材でやるだけです。合っているかどうかは自分で判断できるので、精度を測る練習にもなります。

今月中に

CLAUDE.md を1枚書いてみる

いま関わっているプロジェクトの前提を10行ほど書きます。使っている言語とバージョン、触ってはいけないファイル、必ず守ってほしい書き方。今日配った `CLAUDE.md` をひな形にして、中身を差し替えるところから始めるのが速いです。

続けて

点検の観点をファイルに出す

レビューで毎回指摘している内容を、頭の中からファイルへ移します。演習 A-3 で置く `docs/` の4本がその見本です。観点がファイルになっていると、AIにも人にも同じ基準で頼めるようになります。

今日繰り返し出てくる言葉

240分のあいだ、次の言葉が何度も出てきます。いま完璧に覚える必要はありません。演習の中で実物を見たときに「これのことか」と結びつけばそれで十分です。

エージェント

agent

指示を受けて、自分で調べて、書いて、実行するところまで進めるAIの使い方です。今日の Claude Code がこれにあたります。

バイブコーディング

vibe coding

細かい設計を決めずに、思いついた指示を重ねて作っていく進め方です。課題A の前半で体験します。

ハーネス

harness

AIを毎回同じ手順・同じ観点で動かすための設定一式です。本研修では `CLAUDE.md` と `.claude/` と `docs/` を指します。

コンテキスト

context

AIがいま参照できる情報の全体です。開いているファイル、これまでの会話、読み込まれた設定のすべてが含まれます。

Plan モード

plan mode

調べて計画を出しますが、コードは書き換えないモードです。課題B で不具合を探すときに使います。

実行サマリー

execution summary

応答の最後にAIが自己申告する4項目です。動作／従った設定／対象ファイル／未実施・保留。設定を置いた瞬間から出るようになります。

スラッシュコマンド

slash command

`/名前` で呼び出す手順書です。中身は日本語の文章で書かれています。今日使うのは `/selfcheck` と `/regression-check` です。

Skill

skill

特定の作業のやり方をまとめたフォルダです。説明文とユーザーの発話が合致すると自動的に選ばれます。課題Bで自分でも1つ作ります。

Issue 駆動

issue driven

直す対象を先に文書にしてから、それを根拠に修正を進める進め方です。課題B の B-3 から B-4 で通しで体験します。

それでは始めます

まず VSCode を開き、`handson` フォルダを開いてください。続いてアクティビティバーの Claude Code アイコンからパネルを開き、`こんにちは。今日の作業フォルダの中身を教えてください` と1文だけ送ってみてください。応答が返ったら、S01 の環境確認は通過です。

演習中に開くファイルは `exercises/` の各手順書です。詰まったら `hints/` 、それでも動かなければ挙手をお願いします。

SOURCES

Claude Code Docs — Common workflows（読ませる・直させるの基本パターン）
Claude Code Docs — Memory（CLAUDE.md の書き方と読み込まれ方）
Claude Code Docs — Agent Skills（Skill の作り方と選ばれ方）
Claude Code Docs — Subagents（別枠で調べさせる仕組み）
Claude Code Docs — Commands（スラッシュコマンドの一覧）
Anthropic Engineering — Best practices for Claude Code

生成AIプログラミング入門編 オリエンテーション

Givery, Inc.

講師: 安田 光喜（Givery）