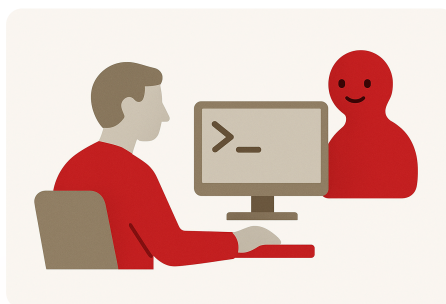


生成AIプログラミング入門編

VSCode と Claude Code で、動くアプリを自分の手で作る4時間

Session	内容	時間
S01	オリエンと支援の段階遷移	[20min]
S02	座学とデモ	[40min]
S03	ハンズオン 課題A (CRUD業務アプリ)	[70min]
S04	ハンズオン 課題B (Java / Spring Boot) と発展	[70min]
-	まとめとAI設計デモ	[20min]
S05	振り返りと質疑応答	[20min]

実施環境は VSCode × Claude Code (Amazon Bedrock 経由・Claude Sonnet 4.6) です。前半で AI 開発の威力を体験し、後半でその限界と、チーム開発に持ち込むための考え方まで進みます。



本研修のゴール

生成AIに日本語で指示して、動くアプリを自分の手で完成させる体験を持ち帰ります。あわせて、勢いだけで作ったときに何が起きるかまで自分の目で確かめます。

今日できるようになること

VSCode 上で Claude Code に指示を出し、テンプレートなしで CRUD 業務アプリを作る。バグ入りの Java / Spring Boot コードを AI と一緒に読み解き、Issue 駆動で修正する。この2つを 240 分で通します。

Claude Code クロードコード

Anthropic 社が提供する開発支援 AI です。本研修では VSCode の拡張として使い、日本語の文章で指示すると、関連ファイルを自分で探し、複数ファイルをまたいで編集し、動作確認まで進めます。接続先は Amazon Bedrock 上の Claude Sonnet 4.6 です。

S01 オリエンテーション

進め方と最初の確認

つまずきは冒頭で解消します。まず全員の VSCode と Claude Code が起動するところまで、この場で確認します。

いま確認すること

- ☐ VSCode が起動する
- ☐ 配布フォルダ（ZIP を展開したもの）が開ける
- ☐ Claude Code のパネルが表示される
- ☐ 短い指示に応答が返る（接続確認）

接続確認の指示（そのまま入力）

このプロジェクトにどんなファイルがあるか、一覧で教えてください。

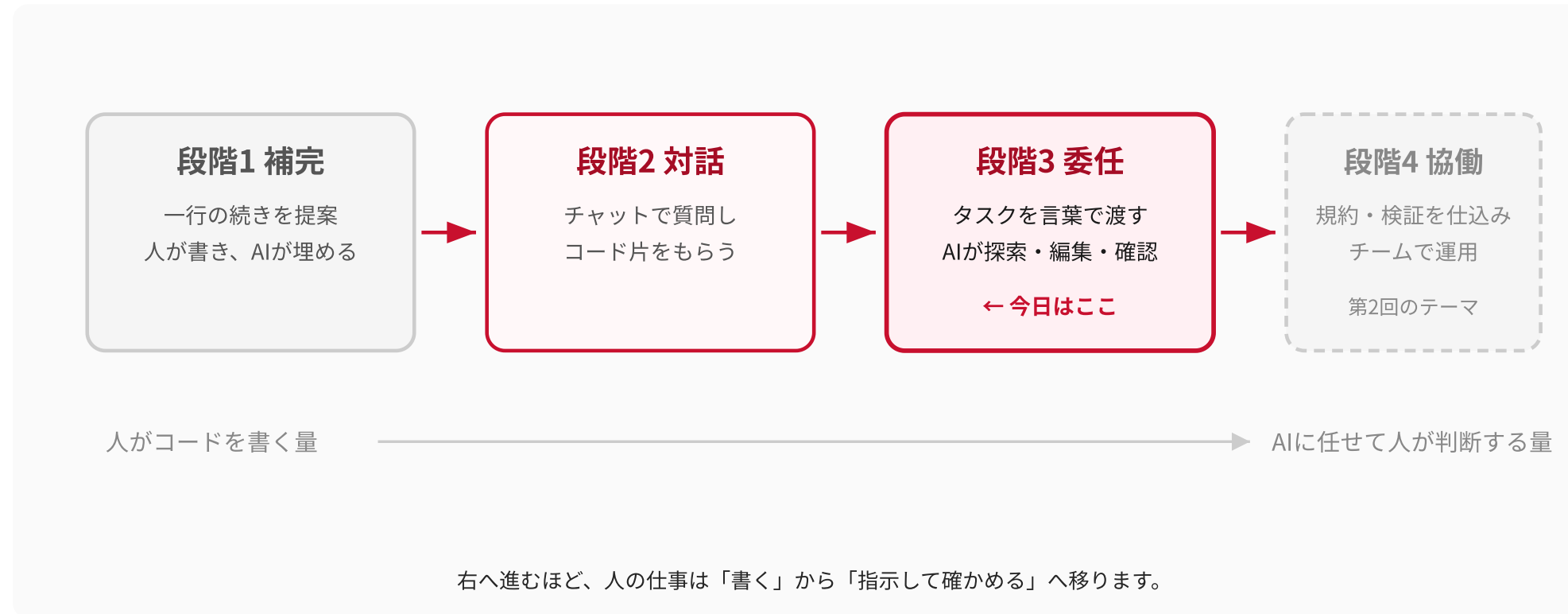
今日のルール

- 1 手を止めない。** エラーが出たらず AI に貼り付けて聞きます。それでも進まなければ挙手してください。講師が回ります。
- 2 完璧を目指さない。** まず動かす、それから直す。今日はこの順番で進めます。
- 3 配布物の外に出ない。** 題材はすべて配布フォルダ内で完結します。外部サービスへの登録は不要です。

S01 支援の段階遷移

AI支援の段階遷移 — 補完から対話委任へ

コードを書く AI の支援は、この数年で「一行の続きを埋める」から「タスクごと任せる」へ段階的に移ってきました。今日扱うのは右側の段階です。



今日の位置づけ

本研修は段階3の体験に集中します。段階4（規約や自動検証を仕込んだチーム運用）は、最後の AI 設計デモで入り口だけ見せ、第2回で本格的に扱います。

AIコーディングの現在地

生成AIでコードを書くことは、もう珍しい取り組みではありません。一方で、出てきたコードをそのまま信じてよいわけでもありません。

広がっている使い方

- 新規画面やちょっとしたツールの下書きを AI に任せる
- 既存コードの読解・説明・コメント付けを頼む
- エラーメッセージを貼って原因を探させる
- テストデータやドキュメントの生成

開発者の大多数が AI ツールを日常的に使う一方、出力を無条件に信頼する開発者は少数にとどまります。「使うが、確かめる」が現場の標準です。

Stack Overflow Developer Survey 2025 survey.stackoverflow.co

AIが得意なこと・任せ方

- ◎ **形が決まっているもの。**一覧画面、検索、入力フォームのような定型的な構造は非常に速く正確です。
- ◎ **既存コードの説明。**読み慣れないコードの要約やコメント挿入は得意分野です。
- △ **曖昧な指示。**「いい感じにして」では AI が勝手に仕様を決めます。何を・なぜを言葉にするほど結果が安定します。
- × **正しさの保証。**動いて見えても、セキュリティや境界条件の穴は残ります。今日の後半で実際に確かめます。

ツールの地図 — 本研修は Claude Code を使う

コードを書く AI ツールは形態別に整理すると見通しがよくなります。比較はここで一望し、以後は Claude Code に集中します。

比較軸	Claude Code	GitHub Copilot	Cursor
形態	VSCode 拡張 / ターミナルで動くエージェント	VSCode 拡張（補完＋チャット）	エディター体型
主な使い方	タスクを言葉で渡して自走させる	入力中の補完と対話	補完と対話
規約ファイル	CLAUDE.md	copilot-instructions.md	.cursor/rules
本研修での扱い	主役。全セッションで使う	名称のみ紹介	名称のみ紹介

エージェント agent

一文の指示から、関連ファイルを自分で探し、複数の場所を編集し、実行して結果を確かめ、エラーが出たら直す、という一連の作業を続けて進める働き方です。補完のように一行ずつ提案するのとは役割が違います。

Amazon Bedrock ベッドロック

AWS 上で各社の AI モデルを利用できるサービスです。本研修の Claude Code は Bedrock 経由で Claude Sonnet 4.6 に接続します。操作は通常の Claude Code と同じで、接続先だけが違います。

同じ目的でも、指示で結果が変わる

AI への指示は「注文」です。何を・なぜ・どう確かめるかが入っているほど、意図に近い結果が返ります。

曖昧な指示

BEFORE

検索機能を作って。

- どの項目で検索するかを AI が勝手に決める
- 部分一致か完全一致かが運任せになる
- 結果が0件のときの表示が考慮されない

意図が伝わる指示

AFTER

商品一覧に検索機能を追加してください。

- 検索対象は商品名で、部分一致
- 検索ボックスは一覧の上に置く
- 0件のときは「該当する商品がありません」と表示
- 既存の一覧表示の作りは変えないでください

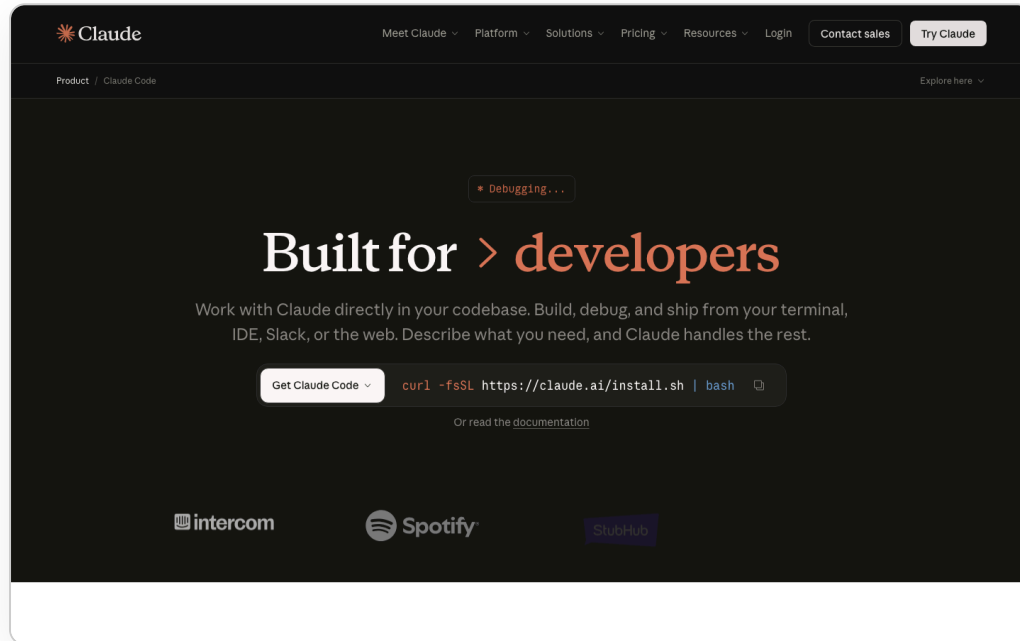
- 仕様のブレが減り、やり直しが減る
- 「変えないでほしい部分」を先に守れる

指示の型

「何を（対象と完成条件）＋ どう（制約と例外時の挙動）＋ 触らないでほしい部分」。この3点を意識すると、初心者でも指示の質が安定します。ハンズオン中もこの型に戻ってください。

VSCode での Claude Code 基本操作

操作はシンプルです。パネルを開き、日本語で話しかけ、提案された変更を確認して承認する。このサイクルの繰り返しです。



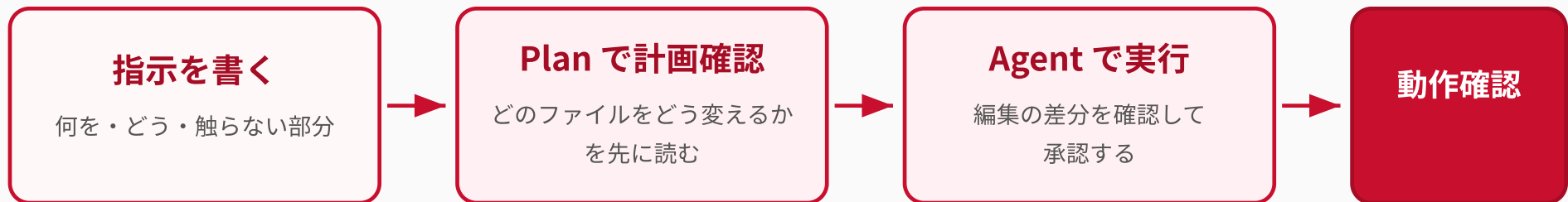
VSCode のサイドバーから Claude Code パネルを開き、下部の入力欄に日本語で指示を書きます。

- 1 起動。** VSCode 左のアイコンから Claude Code パネルを開きます。配布フォルダを開いた状態で起動すると、そのプロジェクトが文脈になります。
- 2 内容把握。** 「このプロジェクトの構成を説明して」と頼むと、AI がファイルを読み、全体像を返します。作業前の地図になります。
- 3 編集サイクル。** 指示 → AI がファイルを読み・編集案を提示 → 差分を確認して承認 → 動作確認。うまくいかなければ文脈を足して言い直します。
- 4 ファイル指定。** @ファイル名 で対象ファイルを明示できます。「どこを直すか」を人が示すと精度が上がります。

Plan モードと Agent モード

Claude Code には「先に計画だけ立てる」モードと「編集まで進める」モードがあります。使い分けが今日いちばん大事な操作です。

	Plan モード	Agent モード
動き	ファイルを読み、変更の計画だけを提示する。ファイルは編集しない	計画にもとづいてファイルの編集・実行まで進める
向く場面	変更範囲が広いとき、何をされるか先に知りたいとき	やることが明確で、小さく進めたいとき
切り替え	入力欄のモード表示から切り替えます（キーボードでは Shift+Tab）	
今日の使い方	課題Bのバグ修正は Plan で計画を見てから直す	課題Aの CRUD 構築は Agent でテンポよく作る



大きな変更ほど Plan を挟みます。計画が意図とずれていたら、実行前に指示を直せます。

課題A — CRUD業務アプリをテンプレートなしで作る

一覧が表示され、検索で絞り込み、1件選ぶと詳細が見える。この業務アプリを、雛形コードなしのまっさらな状態から Claude Code と作ります。

完成イメージ

一覧画面 → 検索で絞り込み → 行をクリックして詳細表示、というドリルダウンが一通り動くこと。見た目の作り込みは不要です。まず全員が「動くもの」を持つことを優先します。

CRUD クラッド

Create（作成）・Read（参照）・Update（更新）・Delete（削除）の頭文字で、業務アプリの基本操作を指します。今日はこのうち参照系（一覧・検索・詳細）を中心に作ります。

環境確認 — 最初の1指示を全員で

いきなり作り始める前に、短い指示を1つ全員で実行し、操作のリズムとつまずきをここで解消します。

手順

- 1 配布フォルダの `kadai-a` を VSCode で開きます。中は空に近い状態です（題材の説明ファイルだけです）。
- 2 Claude Code パネルを開き、下の指示をそのまま入力します。

最初の1指示

「Hello」とだけ表示する簡単な HTML ファイルを `index.html` という名前で作ってください。

- 3 作成された `index.html` をブラウザで開き、表示を確認します。

OK基準

- ☐ Claude Code が応答を返した
- ☐ ファイルが作成され、差分の承認ができた
- ☐ ブラウザで表示を確認できた

ここで止まったら挙手

この3つが通れば、以降の課題はすべて同じ操作の延長です。逆にここで詰まったまま先へ進むと後で必ず止まるので、遠慮なく挙手してください。

課題Aの進め方 — 3段階で積み上げる

一気に全部を頼まず、一覧 → 検索 → 詳細の順に1段ずつ作って動作確認します。小さく作って確かめるリズムが、そのまま実務の型になります。

段階	作るもの	指示のポイント	OK基準
Step 1	一覧画面	題材のデータ項目を伝え、サンプルデータ込みで一覧表を作らせる	ブラウザで一覧が表として表示される
Step 2	検索	検索対象の項目・部分一致・0件時の表示を指定する	入力に応じて一覧が絞り込まれる。0件表示も出る
Step 3	詳細ドリルダウン	行を選ぶと全項目が見える詳細表示。一覧へ戻る導線も指定する	一覧 → 詳細 → 一覧の往復ができる

STEP 1 の指示例（題材に合わせて書き換えて使う）

お題の説明ファイル @README.md を読んで、その業務データの一覧画面を index.html として作ってください。

- ・サンプルデータを10件ほど含めてください
- ・表形式で、項目名のヘッダー行を付けてください
- ・後で検索と詳細表示を足すので、シンプルな作りにしてください

時間が余ったら

登録フォーム（Create）や削除（Delete）の追加、並び替え、件数表示など、自分の業務で欲しい機能を足してみてください。指示の型は同じです。

完成したアプリを、AI自身に点検させる

動いた。ここで終わらず、いま作ったアプリの危険な箇所・曖昧な箇所・Warning を、作った本人である AI に指摘させます。

点検の指示（そのまま入力）

いま作ったこのアプリを、セキュリティと品質の観点でレビューしてください。

- ・危険な実装や脆弱性になりうる箇所
- ・仕様が曖昧なまま実装した箇所（あなたが勝手に決めた仕様）
- ・エラー処理が足りない箇所や Warning

を、重要な順に一覧で指摘してください。修正はまだしないでください。

よく挙がる指摘の例

- 入力値をそのまま画面に出している（XSS の温床）
- 検索の大文字小文字や空白の扱いを勝手に決めていた
- データが増えたときの表示や性能を考慮していない
- エラー時にユーザーへ何も知らせない

この体験の意味

動いた ≠ 正しい

数十分で動くものが作れた一方、その中には AI が勝手に決めた仕様と、指摘されるまで気づかない穴が残っています。速さの裏で何が起きているかを、自分の生成物で確かめるのがこの時間の目的です。

Warning 警告

エラーではないものの、放置すると不具合や脆弱性につながりうる状態を知らせるメッセージです。動いていても Warning が残っていれば、内容を理解してから先へ進むのが安全です。

バイブコーディングの限界と、ハーネスが要る理由

いま体験した「勢いで指示して動くものを作る」進め方をバイブコーディングと呼びます。一人の試作では最強ですが、チーム開発に持ち込むと破綻します。

一人なら強い

- アイデアを数十分で動く形にできる
- 仕様の細部は動かしながら決められる
- 作り直しのコストが低い

チームだと破綻する

- AI が勝手に決めた仕様が人によってバラバラになる
- 命名・構成・書き方に一貫性がなく、他人が読めない
- 点検を人の善意に任せると、穴がそのまま本番に届く

バイブコーディング

速いが、品質は毎回の運と本人の注意力に依存



ハーネス（枠組み）を仕込む

規約・チェック・手順をあらかじめ AI に効かせ、
誰がやっても一定の品質に着地させる

速さはそのままに、品質を運任せにしない仕組みがハーネスです。

ハーネス harness

AI の出力を一定の品質へ導くための枠組みの総称です。規約ファイル、決まった手順、自動チェックなどを組み合わせます。詳しくは今日の最後のデモと、第2回で扱います。

課題B — Java / Spring Boot のバグを Issue 駆動で直す

今度は他人が書いた（バグ入りの）既存コードが相手です。AI でコードを理解し、直すべき点を Issue として言葉にしてから修正する、実務に近い流れを体験します。

フェーズ	やること
1. 理解	配布された Spring Boot プロジェクトを AI に読ませ、コメント挿入と構成説明でソースを把握する
2. 起票	見つけたバグ・修正点を Issue テンプレートに沿って言葉にする（起票練習）
3. 修正	自分の Issue を2〜3個選び、Issue 駆動で約20分かけて修正・動作確認する

Spring Boot スプリングブート

Java で Web アプリや API を作るときに広く使われるフレームワークです。今日は書けなくて構いません。読んで直す側に回るための道具として AI を使います。CI/CD などの自動化は本研修では扱いません。

まず読む — AIにコメントを入れさせて構造を掴む

知らないコードをいきなり直すのは危険です。最初の仕事は理解で、ここがAIのいちばん得意な領域です。

手順

- 1 配布フォルダの `kadai-b` を VSCode で開き、Claude Code に全体構成を説明させます。

構成把握の指示

この Spring Boot プロジェクトの構成を説明してください。

- 主要なクラスの役割
 - 画面や API の一覧
 - データの流れ（リクエストからレスポンスまで）
- を、Java に詳しくない人にも分かる言葉でお願いします。

- 2 気になるファイルに日本語コメントを挿入させ、処理を追える状態にします。

コメント挿入の指示

@対象ファイル名 に、処理の意図が分かる日本語コメントを挿入してください。コードのロジック自体は変更しないでください。

OK基準

- ☐ アプリが何をするものか、自分の言葉で言える
- ☐ 主要クラスの役割分担が説明できる
- ☐ 怪しい・引かかる箇所を1つ以上見つけた

リバースエンジニアリングの手前まで

目的は完全な解析ではなく、直すために必要な範囲の理解です。「どこに何があるか」と「怪しい箇所の当たり」が付けば十分です。読解に時間を使いすぎないでください。

Issue 駆動開発 — 直す前に、言葉にする

見つけた問題をいきなり直さず、まず Issue（課題票）として言葉にします。本研修では配布フォルダ内の Markdown ファイルとして起票します。

Issue テンプレート（配布済み）

```
## 概要
（何が起きているか、1~2行）

## 再現手順
1. ○○画面を開く
2. △△を入力して実行する

## 期待する動作
（本来どうなるべきか）

## 実際の動作
（いま何が起きるか）

## 修正方針（分かる範囲で）
（どのファイルのどこが原因か、仮説でよい）
```

なぜ言葉にしてから直すのか

- 1 指示の質が上がる。期待と実際の差が言葉になっていれば、それがそのまま AI への良い指示になります。
- 2 直し過ぎを防ぐ。Issue の範囲だけ直すと決めておくと、AI が関係ない場所まで書き換えるのを止められます。
- 3 チームで分担できる。実務では Issue が担当割りとレビューの単位になります。今日はその型だけ先に体に入れます。

起票練習

まず1件、見つけた問題をテンプレートに沿って `issues/issue-01.md` として書いてみてください。再現手順が書けない場合は、AI に「この問題の再現手順を整理して」と頼んで構いません。

Issue 駆動で修正する — 2～3件を約20分で

自分が起票した Issue から2～3件を選び、1件ずつ「Plan で計画 → 修正 → 動作確認」を回します。

1件あたりの進め方

- 1 Plan モードで Issue ファイルを渡し、修正計画を出させます。

修正指示の型

@issues/issue-01.md の内容を修正してください。

- この Issue に書いた範囲だけを直してください
- 関係ないリファクタリングはしないでください
- 修正後に、どこをどう直したか説明してください

- 2 計画が Issue の範囲に収まっているか確認し、Agent モードで実行します。
- 3 アプリを動かして、Issue の「期待する動作」になったか確かめます。

OK基準

- ☐ 2件以上の Issue を起票できた
- ☐ 2件以上を修正し、動作確認まで通した
- ☐ 修正内容を自分の言葉で説明できる

直ったように見えて直っていないことがある

AI の「修正しました」は完了の宣言であって証明ではありません。必ず再現手順をなぞり直して、期待する動作に変わったことを自分の目で確認してください。確認までがこの演習です。

発展 — エージェントの進化とハルシネーション

今日体験した「委任」の先で何が起きているか、そして任せるときに必ず知っておくべき AI の弱点を押さえます。

AIエージェントの発展

- 1指示で計画 → 実装 → テスト → 修正まで自走する範囲が広がっている
- 複数のエージェントが役割分担して並行作業する構成も実用段階
- それでも「何を作るべきか」「出力を採用するか」の判断は人に残る

任せる範囲が広がるほど、仕込みが効く

自走する範囲が広がるほど、事前に渡す規約・知識・手順の質が結果を左右します。これが次のコンテキストエンジニアリングの話につながります。

ハルシネーションの具体例

存在しないライブラリやメソッドを、実在するかのような自然さで使ってくる。

例: 実在しない便利メソッドを import して「動くはず」のコードを提示する

エラーの原因を、もっともらしいが誤った理由で断定する。

例: 実際は設定ミスなのに「ライブラリのバグです」と説明する

- 対策1: 出力は仮説として扱い、実行・ログ・公式ドキュメントで裏を取る
- 対策2: 「分からなければ分からないと言って」と指示に添える
- 対策3: 重要な判断ほど、根拠となるコードや出典を出させる

まとめとAI設計デモ

今日の総括 — バイブコーディングの強みと弱み

課題Aで威力を、課題Bで規律を体験しました。両方を一枚で振り返ります。

	体験したこと	そこから言えること
課題A（威力）	テンプレートなしから数十分で一覧・検索・詳細が動いた	定型的な業務アプリの立ち上げは、もう人が一から書く仕事ではない
課題A（限界）	AI 自身に点検させたら、勝手に決めた仕様と穴が出てきた	速さの裏で品質は運任せ。動いた ≠ 正しい
課題B（規律）	Issue に言葉で固定してから直すと、AI の修正が範囲内に収まった	先に言葉と枠を与えるほど、AI は安定して働く

次の問い

ではチームとして、この「先に与える言葉と枠」をどう用意するか。それがコンテキストエンジニアリングと、これから見せるハーネスの設計です。

コンテキストエンジニアリング — AIに渡す知識を整える

AI の出力の質は、モデルの賢さより「何を渡したか」で決まる場面が多くあります。会社のナレッジや個人の暗黙知を、AI が読める形に整えておく仕事がコンテキストエンジニアリングです。

整えて渡すものの例

- コーディング規約・命名ルール（例: `CLAUDE.md` に記載）
- 業務用語集とデータ項目の定義
- 「このシステムではこうする」という過去の判断とその理由
- やってはいけないことのリスト（触ってはいけないファイル、禁止パターン）

コンテキスト context

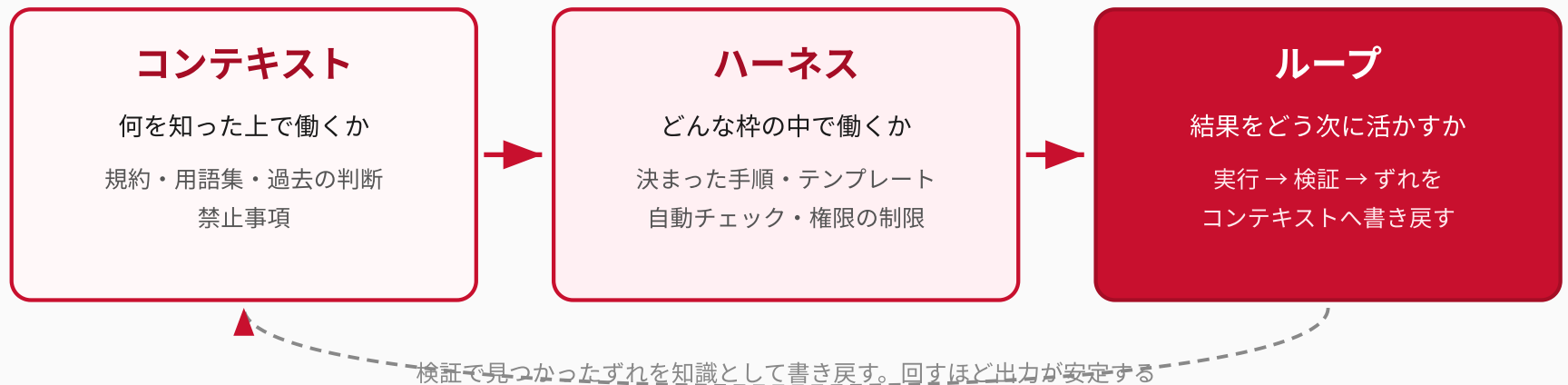
AI が応答を作るときに参照する文脈情報のことです。指示文だけでなく、開いているプロジェクトのファイルや規約ファイルもコンテキストになります。

チームでの運用

- 1 **個人の頭の中を文書へ。**ベテランの暗黙知（レビューでいつも指摘すること）を規約ファイルに落とすと、全員の AI 出力に効きます。
- 2 **育てる前提で置く。**AI の出力が規約とずれたら、その都度ファイルに追記します。使うほど賢くなる資産になります。
- 3 **入力可否のルールも文書に。**顧客情報や本番データを AI に入れてよいかは、各社のセキュリティポリシーに従って先に決めておきます。

設計思想 — コンテキスト → ハーネス → ループ

チームでAIを安定して働かせる設計は、この3層で考えると整理できます。今日の体験はすべてこの図の上に載ります。



知識を渡し、枠の中で走らせ、結果を知識へ返す。この循環が「使うほど良くなる」AI 活用の骨格です。

今日の体験との対応

課題Aの点検指示は簡易的なハーネス、課題Bの Issue テンプレートはコンテキストとハーネスの中間にあたります。第2回では、これをチームの仕組みとして本格的に組み上げます。

デモ — 1コマンドから Skills / Subagents を自動選択

仕込みが整った環境で AI がどう働くか、講師の環境で実演します。人が出すのは1つのコマンドだけです。

デモで見せる流れ

- 1 コマンド投入。** あらかじめ定義したコマンドを1つ実行します。
指示文はごく短いものです。
- 2 AI が道具を選ぶ。** タスクの内容から、必要な Skill（手順書付きの専門能力）と Subagent（役割分担した子エージェント）を AI 自身が選んで呼び出します。
- 3 検証込みで着地。** 生成 → 自動チェック → 修正のループが回り、人が細かく指示しなくても効果の高い出力に到達します。

Skill スキル

特定の作業の手順・品質基準・雛形をまとめたパッケージです。AI はタスクに合う Skill を選んで読み込み、その手順に沿って作業します。ベテランの仕事の型を配る仕組みと考えてください。

Subagent サブエージェント

役割を限定した子の AI です。調査係・実装係・レビュー係のように分担させると、1体で全部やるより速く、間違いも見つかりやすくなります。

デモの見どころ

注目してほしいのは出力の見事さではなく、人が介入していない区間の長さです。介入せずに任せられる範囲は、事前の仕込みの量で決まります。

振り返り — 今日できるようになったこと

最後に、今日の体験を自分の言葉にします。次の4つの問いに、手元のメモで答えてみてください。書けたものが、明日から現場で使える持ち帰りです。

- 1 **何が作れたか。** 課題A・課題Bで自分が完成させたものを一言で。
- 2 **どんな指示が効いたか。** 結果が良くなった指示の書き方を1つ。
- 3 **何に気をつけるか。** 点検・ハルシネーションの体験から、任せるときの注意を1つ。
- 4 **明日、何から始めるか。** 自分の業務で最初に AI に頼んでみる小さな仕事を1つ。

明日からの最初の一步の例

- 読み慣れないコードやファイルの説明を頼む
- 手元の繰り返し作業を小さなツールにしてもらう
- エラーメッセージを貼って原因の仮説を出させる

実業務での入力可否は必ず確認

研修では自由に入力できましたが、実務でのソースコード・本番データ・顧客情報の入力可否は、所属組織のセキュリティポリシーと AI 利用ガイドラインに従って判断してください。

この後は質疑応答の時間です。今日の操作でも、実務への持ち込み方でも、何でも聞いてください。

アンケート

本日はおつかれさまでした

最後にアンケートへのご協力をお願いします。今日の内容と、第2回で扱ってほしいテーマをぜひお聞かせください。

QRコード

確定後に差し替え

回答用URL

https://（確定後に差し替え）

所要時間の目安: 3分程度

アンケートの QR コードと URL は、確定後にこの枠へ差し替えます。

指示を言葉にし、計画を確かめ、生成物を点検する。今日手に馴染ませたこの型が、そのまま現場での AI 活用の出発点になります。