

Claude Certified Architect – Foundations

試験ガイド

バージョン1.0・発効：2026年7月・試験コード：CCAR-F・本ガイドは予告なく変更されることがあります。

1. この認定資格について

Claude Certified Architect – Foundations認定資格は、実務者がClaudeを使って実世界のソリューションを実装する際に、トレードオフについて十分な情報に基づいた判断を下せることを証明するものです。本試験では、Claude Code、Claude Agent SDK、Claude API、Model Context Protocol（MCP）という、Claudeで本番環境レベルのアプリケーションを構築するために使用される中核技術に関する基礎知識を問います。

本試験の問題は、実際の顧客ユースケースから取り上げた現実的なシナリオに基づいています。これには、カスタマーサポート向けのエージェントシステムの構築、マルチエージェントのリサーチパイプラインの設計、CI/CDワークフローへのClaude Codeの統合、開発者の生産性向上ツールの構築、非構造化ドキュメントからの構造化データの抽出が含まれます。受験者は、概念的な知識だけでなく、本番環境へのデプロイにおけるアーキテクチャ、設定、トレードオフに関する実践的な判断力を示す必要があります。

本ガイドは、受験の準備をする受験者にとっての正式な参照資料です。試験の内容を説明し、出題される分野とタスクステートメントを一覧で示し、サンプル問題を掲載し、準備の進め方を推奨しています。試験を予約する前に、全体をお読みください。

2. 対象となる受験者

この認定資格の理想的な受験者は、Claudeを使って本番アプリケーションを設計・実装するソリューションアーキテクトです。この受験者は、次の実務経験を有しています：

- Claude Agent SDKを使用したエージェントアプリケーションの構築（マルチエージェントのオーケストレーション、サブエージェントへの委任、ツール統合、ライフサイクルフックを含む）
- CLAUDE.mdファイル、Agent Skills、MCPサーバー統合、プランモードを使用した、チームのワークフローに合わせたClaude Codeの設定とカスタマイズ
- バックエンドシステム統合のための、Model Context Protocol（MCP）のツールおよびリソースのインターフェースの設計
- JSONスキーマ、few-shotの例、抽出パターンを活用した、信頼性の高い構造化出力を生成するプロンプトの設計

- 長いドキュメント、マルチターンの会話、マルチエージェント間のハンドオフにわたるコンテキストウィンドウの効果的な管理
- 自動コードレビュー、テスト生成、プルリクエストへのフィードバックのための、CI/CDパイプラインへのClaudeの統合
- エラー処理、ヒューマン・イン・ザ・ループのワークフロー、自己評価パターンを含む、エスカレーションと信頼性に関する適切な判断

受験者は通常、Claude API、Agent SDK、Claude Code、MCPを使った構築の実務経験を6か月以上有しており、本番環境における大規模言語モデルの能力と限界の両方を理解しています。

3. 試験の概要

資格	Claude Certified Architect – Foundations
試験コード	CCAR-F
問題数	60
出題形式	択一式および複数選択式の問題。各問題に、選択する解答の数が明記されています
試験の構成	6つのシナリオのバンクから出題される4つのシナリオ
制限時間	120分
実施方法	監督付き：プログラムのポリシーに従い、オンライン監督付きまたはテストセンター、あるいはその両方
合格点	100-1,000のスケールでスケールスコア720
受験料	\$125 USD
有効期間	資格が授与された日から12か月
結果の通知	合格／不合格とスケールスコア（100-1,000）。スコアレポートには分野別の正答率も記載されます

4. 試験内容の概要（ブループリント）

試験ブループリントは、測定対象となる内容分野と、試験における各分野のおおよその配点比率を定めています。配点比率は、職務タスク分析によって決定された、十分な能力での業務遂行に対する各分野の相対的な重要度を反映しています。パーセンテージは、採点対象の問題のうち各分野から出題されるおおよその割合を示します。

分野	内容分野	配点比率
1	エージェントアーキテクチャとオーケストレーション	27%
2	ツール設計とMCP統合	18%
3	Claude Codeの設定とワークフロー	20%
4	プロンプトエンジニアリングと構造化出力	20%
5	コンテキスト管理と信頼性	15%
合計	100%	

5. 試験シナリオ

本試験ではシナリオベースの問題を使用します。各シナリオは、一連の問題の前提となる現実的な本番環境の状況を提示します。試験では、以下の6つのシナリオ全体の中から無作為に選ばれた4つのシナリオが出題されます。

シナリオ1：カスタマーサポート解決エージェント

あなたは、Claude Agent SDKを使用してカスタマーサポート解決エージェントを構築しています。このエージェントは、返品、請求に関する異議、アカウントの問題など、曖昧さの大きいリクエストを処理します。カスタムのModel Context Protocol（MCP）ツール（`get_customer`、`lookup_order`、`process_refund`、`escalate_to_human`）を通じて、バックエンドシステムにアクセスできます。目標は、エスカレーションすべきタイミングを見極めながら、初回対応での解決率80%以上を達成することです。

主な分野：エージェントアーキテクチャとオーケストレーション、ツール設計とMCP統合、コンテキスト管理と信頼性

シナリオ2：Claude Codeによるコード生成

あなたは、ソフトウェア開発を加速するためにClaude Codeを使用しています。チームでは、コード生成、リファクタリング、デバッグ、ドキュメント作成にClaude Codeを使っています。カスタムスラッシュコマンドやCLAUDE.mdの設定を用いて開発ワークフローに統合し、プランモードと直接実行をどのように使い分けるかを理解する必要があります。

主な分野：Claude Codeの設定とワークフロー、コンテキスト管理と信頼性

シナリオ3：マルチエージェントリサーチシステム

あなたは、Claude Agent SDKを使用してマルチエージェントリサーチシステムを構築しています。コーディネーターエージェントが、専門化されたサブエージェントに作業を委任します。サブエージェントには、ウェブを検索するもの、ドキュメントを分析するもの、調査結果を統合するもの、レポートを生成するものがあります。このシステムはトピックを調査し、出典を明記した包括的なレポートを作成します。

主な分野：エージェントアーキテクチャとオーケストレーション、ツール設計とMCP統合、コンテキスト管理と信頼性

シナリオ4：Claudeによる開発者の生産性向上

あなたは、Claude Agent SDKを使用して開発者の生産性向上ツールを構築しています。このエージェントは、エンジニアが不慣れなコードベースを探索し、レガシーシステムを理解し、ボイラープレートコードを生成し、反復的なタスクを自動化するのを支援します。組み込みツール（Read、Write、Bash、Grep、Glob）を使用し、Model Context Protocol（MCP）サーバーと統合します。

主な分野：ツール設計とMCP統合、Claude Codeの設定とワークフロー、エージェントアーキテクチャとオーケストレーション

シナリオ5：継続的インテグレーションのためのClaude Code

あなたは、Claude Codeを継続的インテグレーション／継続的デプロイ（CI/CD）パイプラインに統合しています。このシステムは、自動コードレビューを実行し、テストケースを生成し、プルリクエストに対するフィードバックを提供します。実行可能なフィードバックを提供し、誤検知を最小限に抑えるプロンプトを設計する必要があります。

主な分野：Claude Codeの設定とワークフロー、プロンプトエンジニアリングと構造化出力

シナリオ6：構造化データ抽出

あなたは、Claudeを使用して構造化データ抽出システムを構築しています。このシステムは、非構造化ドキュメントから情報を抽出し、JavaScript Object Notation（JSON）スキーマを使用して出力を検証し、高い精度を維持します。エッジケースを適切に処理し、下流のシステムと統合する必要があります。

主な分野：プロンプトエンジニアリングと構造化出力、コンテキスト管理と信頼性

6. 分野別の詳細な目標

以下の各分野には、出題されるタスクステートメントと、それぞれについて測定される知識およびスキルを記載しています。試験問題は、これらの目標に基づいて作成されています。

分野1：エージェントアーキテクチャとオーケストレーション

タスクステートメント1.1：自律的なタスク実行のためのエージェントループを設計し実装する

以下に関する知識：

- エージェントループのライフサイクル：Claudeへのリクエスト送信、stop_reasonの確認 ("tool_use"か"end_turn"か)、要求されたツールの実行、次の反復に向けた結果の返却
- モデルが次のアクションについて推論できるように、ツールの結果が会話履歴に追加される仕組み
- モデル主導の意思決定（Claudeがコンテキストに基づいて次に呼び出すツールを推論する）と、事前に設定された決定木やツールシーケンスとの違い

以下のスキル：

- stop_reasonが"tool_use"の場合は継続し、"end_turn"の場合は終了する、エージェントループの制御フローを実装する
- モデルが新しい情報を推論に取り込めるように、反復の間にツールの結果を会話コンテキストに追加する
- ループ終了の判定に自然言語のシグナルを解析する、主要な停止メカニズムとして任意の反復回数上限を設定する、完了の指標としてアシスタントのテキストコンテンツの有無を確認する、といったアンチパターンを避ける

タスクステートメント1.2：コーディネーター・サブエージェントパターンでマルチエージェントシステムをオーケストレーションする

以下に関する知識：

- コーディネーターエージェントがサブエージェント間のすべての通信、エラー処理、情報のルーティングを管理する、ハブアンドスポーク型アーキテクチャ
- サブエージェントが分離されたコンテキストで動作する仕組み（コーディネーターの会話履歴を自動的に継承しない）
- タスクの分解、委任、結果の集約、およびクエリの複雑さに基づいて呼び出すサブエージェントを決定することにおけるコーディネーターの役割

- コーディネーターによるタスク分解が狭すぎることで、幅広いリサーチトピックを十分にカバーできなくなるリスク

以下のスキル：

- 常にパイプライン全体を経由させるのではなく、クエリの要件を分析して呼び出すサブエージェントを動的に選択するコーディネーターエージェントを設計する
- 重複を最小限に抑えるために、リサーチの範囲をサブエージェント間で分割する（例：各エージェントに異なるサブトピックや情報源の種類を割り当てる）
- コーディネーターが統合の出力に不足がないかを評価し、的を絞ったクエリで検索および分析のサブエージェントに再度委任し、カバー範囲が十分になるまで統合を再度呼び出す、反復的な改善ループを実装する
- 可観測性、一貫したエラー処理、制御された情報フローのために、サブエージェントのすべての通信をコーディネーター経由でルーティングする

タスクステートメント1.3：サブエージェントの呼び出し、コンテキストの受け渡し、生成を設定する

以下に関する知識：

- サブエージェントを生成する仕組みとしてのTaskツール、およびコーディネーターがサブエージェントを呼び出すにはallowedToolsに"Task"を含める必要があるという要件
- サブエージェントのコンテキストはプロンプトで明示的に提供が必要があること（サブエージェントは親のコンテキストを自動的に継承せず、呼び出し間でメモリを共有しない）
- 各サブエージェントタイプの説明、システムプロンプト、ツール制限を含むAgentDefinitionの設定
- 共通の分析ベースラインから異なるアプローチを探るための、フォークベースのセッション管理

以下のスキル：

- 先行するエージェントの調査結果全体をサブエージェントのプロンプトに直接含める（例：ウェブ検索結果とドキュメント分析の出力を統合サブエージェントに渡す）
- エージェント間でコンテキストを渡す際に、出典の帰属を保持するため、構造化データ形式を使用してコンテンツとメタデータ（ソースURL、ドキュメント名、ページ番号）を分離する
- 別々のターンに分けるのではなく、コーディネーターの単一の応答で複数のTaskツール呼び出しを出力することで、サブエージェントを並列に生成する
- サブエージェントが適応できるように、段階的な手順の指示ではなく、リサーチの目標と品質基準を指定するコーディネータープロンプトを設計する

タスクステートメント1.4：強制とハンドオフのパターンを用いて複数ステップのワークフローを実装する

以下に関する知識：

- ワークフローの順序付けにおける、プログラムによる強制（フック、前提条件ゲート）とプロンプトベースのガイダンスの違い
- 決定論的な遵守が求められる場合（例：金融操作の前の本人確認）、プロンプトの指示だけでは失敗率がゼロにならないこと
- プロセス途中でのエスカレーションのための、顧客の詳細、根本原因の分析、推奨アクションを含む構造化されたハンドオフプロトコル

以下のスキル：

- 前提となるステップが完了するまで下流のツール呼び出しをブロックする、プログラムによる前提条件を実装する（例：get_customerが検証済みの顧客IDを返すまでprocess_refundをブロックする）
- 複数の懸案を含む顧客リクエストを個別の項目に分解し、共有コンテキストを使ってそれぞれを並列に調査したうえで、統一された解決策にまとめる
- 会話の記録にアクセスできない人間の担当者にエスカレーションする際に、構造化されたハンドオフサマリー（顧客ID、根本原因、返金額、推奨アクション）を作成する

タスクステートメント1.5：ツール呼び出しのインターセプトとデータ正規化のためにAgent SDKのフックを適用する

以下に関する知識：

- モデルが処理する前にツールの結果をインターセプトして変換するフックパターン（例：PostToolUse）
- 送信されるツール呼び出しをインターセプトしてコンプライアンスルールを強制するフックパターン（例：しきい値を超える返金のブロック）
- 決定論的な保証のためにフックを使用することと、確率的な遵守のためにプロンプトの指示に頼ることの違い

以下のスキル：

- エージェントが処理する前に、異なるMCPツールから返される不均一なデータ形式（Unixタイムスタンプ、ISO 8601、数値のステータスコード）を正規化するPostToolUseフックを実装する
- ポリシーに違反するアクション（例：\$500を超える返金）をブロックし、代替のワークフロー（例：人間へのエスカレーション）にリダイレクトする、ツール呼び出しインターセプトフックを実装する
- ビジネスルールで確実な遵守が求められる場合に、プロンプトベースの強制ではなくフックを選択する

タスクステートメント1.6：複雑なワークフローのためのタスク分解戦略を設計する

以下に関する知識：

- 固定の逐次パイプライン（プロンプトチェーン）と、中間的な発見に基づく動的で適応的な分解を、どのような場合に使い分けるか
- レビューを逐次的なステップに分割するプロンプトチェーンのパターン（例：各ファイルを個別に分析してから、ファイル横断の統合パスを実行する）
- 各ステップで発見された内容に基づいてサブタスクを生成する、適応的な調査計画の価値

以下のスキル：

- ワークフローに適したタスク分解パターンを選択する：予測可能な多面的レビューにはプロンプトチェーン、オープンエンドな調査タスクには動的な分解
- 注意の分散を避けるために、大規模なコードレビューを、ファイルごとのローカル分析パスと、別途実施するファイル横断の統合パスに分割する
- オープンエンドなタスク（例：「レガシーコードベースに包括的なテストを追加する」）を、まず構造を把握し、影響の大きい領域を特定したうえで、依存関係が見つかるたびに適応する優先順位付きの計画を作成することで分解する

タスクステートメント1.7：セッションの状態、再開、フォークを管理する

以下に関する知識：

- 特定の過去の会話を続行するための、`--resume <session-name>`を使用した名前付きセッションの再開
- 共通の分析ベースラインから独立したブランチを作成し、異なるアプローチを探るための`fork_session`
- コード変更後にセッションを再開する際、以前に分析したファイルへの変更をエージェントに伝えることの重要性
- 古くなったツール結果を抱えたまま再開するよりも、構造化されたサマリーを使って新しいセッションを開始するほうが信頼性が高い理由

以下のスキル：

- セッション名を指定して`--resume`を使用し、複数の作業セッションにまたがって名前付きの調査セッションを続行する
- `fork_session`を使用して並列の探索ブランチを作成する（例：共通のコードベース分析から、2つのテスト戦略やリファクタリングのアプローチを比較する）
- セッションの再開（以前のコンテキストがおおむね有効な場合）と、サマリーを注入して新たに開始すること（以前のツール結果が古くなっている場合）のいずれかを選択する
- 全体を再探索させるのではなく、的を絞った再分析のために、再開したセッションに具体的なファイル変更を伝える

分野2：ツール設計とMCP統合

タスクステートメント2.1：明確な説明と境界を備えた効果的なツールインターフェースを設計する

以下に関する知識：

- LLMがツール選択に用いる主要な仕組みとしてのツールの説明。最小限の説明では、類似したツール間での選択の信頼性が低下する
- ツールの説明に、入力形式、クエリの例、エッジケース、境界の説明を含めることの重要性
- 曖昧な、または重複するツールの説明が誤ったルーティングを引き起こす仕組み（例：ほぼ同一の説明を持つanalyze_contentとanalyze_document）
- システムプロンプトの文言がツール選択に与える影響：キーワードに敏感な指示が、意図しないツールの関連付けを生む可能性がある

以下のスキル：

- 各ツールの目的、想定される入力、出力、および類似の代替ツールではなくそのツールを使うべき場面を明確に区別するツールの説明を作成する
- 機能の重複をなくすために、ツールの名前を変更し説明を更新する（例：analyze_contentをextract_web_resultsに名前変更し、ウェブに特化した説明を付ける）
- 汎用的なツールを、入出力の契約が定義された目的別のツールに分割する（例：汎用的なanalyze_documentを、extract_data_points、summarize_content、verify_claim_against_sourceに分割する）
- 適切に書かれたツールの説明を上書きしてしまう可能性のある、キーワードに敏感な指示がないかシステムプロンプトをレビューする

タスクステートメント2.2：MCPツールの構造化されたエラーレスポンスを実装する

以下に関する知識：

- ツールの失敗をエージェントに伝えるための、MCPのisErrorフラグパターン
- 一時的なエラー（タイムアウト、サービスの利用不可）、検証エラー（無効な入力）、ビジネスエラー（ポリシー違反）、権限エラーの違い
- 一律のエラーレスポンス（汎用的な"Operation failed"）では、エージェントが適切な復旧の判断を下せなくなる理由
- 再試行可能なエラーと再試行不可能なエラーの違い、および構造化されたメタデータを返すことで無駄な再試行を防ぐ仕組み

以下のスキル：

- `errorCategory` (`transient/validation/permission`)、ブール値の`isRetryable`、人間が読める説明を含む、構造化されたエラーメタデータを返す
- エージェントが適切に伝えられるように、ビジネスルール違反に対して`retriable: false`フラグと顧客にわかりやすい説明を含める
- 一時的な失敗についてはサブエージェント内でローカルにエラー復旧を実装し、ローカルで解決できないエラーのみを、部分的な結果および試行した内容とともにコーディネーターに伝播する
- アクセスの失敗（再試行の判断が必要）と、有効な空の結果（一致するものがなかった成功したクエリを表す）を区別する

タスクステートメント2.3：エージェント間でツールを適切に配分し、ツール選択を設定する

以下に関する知識：

- エージェントに多すぎるツール（例：4-5個ではなく18個）へのアクセスを与えると、判断の複雑さが増し、ツール選択の信頼性が低下するという原則
- 専門外のツールを持つエージェントが、それらを誤用しがちである理由（例：統合エージェントがウェブ検索を試みる）
- スコープを限定したツールアクセス：エージェントにはその役割に必要なツールのみを与え、頻度の高い特定のニーズに対しては役割横断のツールを限定的に与える
- `tool_choice`の設定オプション：`"auto"`、`"any"`、強制的なツール選択（`{"type": "tool", "name": "..."}`）

以下のスキル：

- 各サブエージェントのツールセットをその役割に関連するものに制限し、専門外のツールの誤用を防ぐ
- 汎用的なツールを制約付きの代替ツールに置き換える（例：`fetch_url`を、ドキュメントURLを検証する`load_document`に置き換える）
- 複雑なケースはコーディネーター経由でルーティングしつつ、頻度の高いニーズに対してはスコープを限定した役割横断のツールを提供する（例：統合エージェント向けの`verify_fact`ツール）
- `tool_choice`の強制選択を使用して特定のツールが最初に呼び出されるようにし（例：エンリッチメントツールの前に`extract_metadata`を強制する）、後続のステップはフォローアップのターンで処理する
- モデルが会話テキストを返すのではなく確実にツールを呼び出すように、`tool_choice: "any"`を設定する

タスクステートメント2.4：MCPサーバーをClaude Codeとエージェントワークフローに統合する

以下に関する知識：

- MCPサーバーのスコープ設定：チームで共有するツール向けのプロジェクトレベル（.mcp.json）と、個人用／実験的なサーバー向けのユーザーレベル（~/.claude.json）
- シークレットをコミットせずに認証情報を管理するための、.mcp.jsonでの環境変数の展開（例：\${GITHUB_TOKEN}）
- 設定されたすべてのMCPサーバーのツールが接続時に検出され、エージェントから同時に利用可能になること
- 探索的なツール呼び出しを減らすために、コンテンツカタログ（例：課題のサマリー、ドキュメントの階層、データベーススキーマ）を公開する仕組みとしてのMCPリソース

以下のスキル：

- 認証トークン用の環境変数の展開を使用して、プロジェクトスコープの.mcp.jsonで共有MCPサーバーを設定する
- ユーザースコープの~/.claude.jsonで個人用／実験的なMCPサーバーを設定する
- MCPツールの説明を拡充して機能と出力を詳しく説明し、エージェントがより高機能なMCPツールよりも組み込みツール（Grepなど）を優先してしまうのを防ぐ
- 標準的な統合（例：Jira）には、カスタム実装ではなく既存のコミュニティ製MCPサーバーを選択し、カスタムサーバーはチーム固有のワークフローのために取っておく
- 探索的なツール呼び出しを必要とせずにエージェントが利用可能なデータを把握できるように、コンテンツカタログをMCPリソースとして公開する

タスクステートメント2.5：組み込みツール（Read、Write、Edit、Bash、Grep、Glob）を効果的に選択し適用する

以下に関する知識：

- コンテンツ検索のためのGrep（関数名、エラーメッセージ、import文などのパターンをファイルの内容から検索する）
- ファイルパスのパターンマッチングのためのGlob（名前や拡張子のパターンでファイルを見つける）
- ファイル全体の操作のためのRead/Write。一意のテキストマッチングを使用した的を絞った変更のためのEdit
- テキストの一致が一意でないためにEditが失敗した場合に、確実なファイル変更のためのフォールバックとしてRead + Writeを使用すること

以下のスキル：

- コードベース全体でコードの内容を検索するためにGrepを選択する（例：ある関数のすべての呼び出し元を見つける、エラーメッセージの場所を特定する）
- 命名パターンに一致するファイルを見つけるためにGlobを選択する（例：`**/*.test.tsx`）
- Editが一意のアンカーテキストを見つけられない場合に、Readでファイルの内容全体を読み込んでからWriteを使用する
- 最初にすべてのファイルを読むのではなく、Grepでエントリーポイントを見つけることから始め、次にReadでimportをたどってフローを追跡することで、コードベースの理解を段階的に構築する
- まずエクスポートされたすべての名前を特定し、次にコードベース全体で各名前を検索することで、ラッパーモジュールをまたぐ関数の使用箇所を追跡する

分野3：Claude Codeの設定とワークフロー

タスクステートメント3.1：適切な階層、スコープ設定、モジュール構成でCLAUDE.mdファイルを設定する

以下に関する知識：

- CLAUDE.mdの設定階層：ユーザーレベル（`~/.claude/CLAUDE.md`）、プロジェクトレベル（`.claude/CLAUDE.md`またはルートのCLAUDE.md）、ディレクトリレベル（サブディレクトリのCLAUDE.mdファイル）
- ユーザーレベルの設定はそのユーザーにのみ適用されること（`~/.claude/CLAUDE.md`内の指示は、バージョン管理を通じてチームメイトと共有されない）
- 外部ファイルを参照してCLAUDE.mdをモジュール化しておくための@import構文（例：各パッケージに関連する特定の標準ファイルをインポートする）
- 単一の巨大なCLAUDE.mdの代替として、トピック別のルールファイルを整理するための`.claude/rules/`ディレクトリ

以下のスキル：

- 設定階層の問題を診断する（例：指示がプロジェクトレベルではなくユーザーレベルの設定にあるために、新しいチームメンバーに指示が届いていない）
- @importを使用し、メンテナのドメイン知識に基づいて、関連する標準ファイルを各パッケージのCLAUDE.mdに選択的に取り込む
- 大きなCLAUDE.mdファイルを、`.claude/rules/`内の焦点を絞ったトピック別ファイルに分割する（例：`testing.md`、`api-conventions.md`、`deployment.md`）

- /memoryコマンドを使用して、どのメモリファイルが読み込まれているかを確認し、セッション間で一貫しない動作を診断する

タスクステートメント3.2：カスタムスラッシュコマンドとスキルを作成し設定する

以下に関する知識：

- .claude/commands/内のプロジェクトスコープのコマンド（バージョン管理を通じて共有）と、~/claude/commands/内のユーザースコープのコマンド（個人用）の違い
- context: fork、allowed-tools、argument-hintなどのフロントマター設定をサポートするSKILL.mdファイルを備えた、.claude/skills/内のスキル
- スキルを分離されたサブエージェントのコンテキストで実行し、スキルの出力がメインの会話を汚染するのを防ぐためのcontext: forkフロントマターオプション
- 個人用のスキルのカスタマイズ：チームメイトに影響を与えないように、~/claude/skills/内に別の名前で個人用のバリエーションを作成すること

以下のスキル：

- バージョン管理を通じてチーム全体で利用できるように、.claude/commands/内にプロジェクトスコープのスラッシュコマンドを作成する
- context: forkを使用して、冗長な出力（例：コードベース分析）や探索的なコンテキスト（例：代替案のブレインストーミング）を生成するスキルをメインセッションから分離する
- スキルのフロントマターでallowed-toolsを設定し、スキル実行中のツールアクセスを制限する（例：破壊的な操作を防ぐためにファイル書き込み操作に限定する）
- argument-hintフロントマターを使用して、開発者が引数なしでスキルを呼び出したときに必須パラメータの入力を促す
- スキル（タスク固有のワークフローのためのオンデマンド呼び出し）とCLAUDE.md（常に読み込まれる全体共通の標準）のどちらを使用するかを選択する

タスクステートメント3.3：規約を条件付きで読み込むためにパス固有のルールを適用する

以下に関する知識：

- 条件付きでルールを有効化するためのglobパターンを含むpathsフィールドをYAMLフロントマターに持つ.claude/rules/ファイル
- パススコープのルールが、一致するファイルを編集するときのみ読み込まれ、無関係なコンテキストとトークン使用量を削減する仕組み
- 複数のディレクトリにまたがる規約（例：コードベース全体に散在するテストファイル）において、ディレクトリレベルのCLAUDE.mdファイルよりもglobパターンのルールが優れている点

以下のスキル：

- YAMLフロントマターでパスのスコープを指定した`claude/rules/`ファイルを作成し（例：`paths: ["terraform/**/*"]`）、一致するファイルを編集するときのみルールが読み込まれるようにする
- パス固有のルールでglobパターンを使用し、ディレクトリの場所に関係なくファイルの種類ごとに規約を適用する（例：すべてのテストファイルに`**/*.test.tsx`）
- コードベース全体に散在するファイルに規約を適用する必要がある場合に、サブディレクトリのCLAUDE.mdファイルではなくパス固有のルールを選択する

タスクステートメント3.4：プランモードと直接実行のどちらを使用するかを判断する

以下に関する知識：

- プランモードは、大規模な変更、複数の有効なアプローチ、アーキテクチャ上の決定、複数ファイルの修正を伴う複雑なタスク向けに設計されていること
- 直接実行は、単純で範囲が明確な変更に適していること（例：ある関数に単一のバリデーションチェックを追加する）
- プランモードでは、変更を確定する前にコードベースの探索と設計を安全に行うことができ、コストのかかる手戻りを防げること
- 冗長な調査出力を分離し、要約を返すことでメインの会話のコンテキストを保全するExploreサブエージェント

以下のスキル：

- アーキテクチャに影響するタスクにプランモードを選択する（例：マイクロサービスの再構成、45+ファイルに影響するライブラリ移行、インフラストラクチャ要件が異なる統合アプローチの選択）
- 内容が十分に理解されていて範囲が明確な変更直接実行を選択する（例：スタックトレースが明確な単一ファイルのバグ修正、日付バリデーションの条件分岐の追加）
- 冗長な調査フェーズにExploreサブエージェントを使用し、複数フェーズのタスクでコンテキストウィンドウが枯渇するのを防ぐ
- 調査のためのプランモードと実装のための直接実行を組み合わせる（例：ライブラリ移行を計画し、その後、計画したアプローチを実行する）

タスクステートメント3.5：段階的な改善のために反復的な改良手法を適用する

以下に関する知識：

- 文章による説明が一貫性なく解釈される場合に、期待する変換を伝える最も効果的な方法としての具体的な入出力の例

- テスト駆動の反復：先にテストスイートを作成し、テストの失敗を共有しながら反復して段階的な改善を導くこと
- インタビューパターン：実装の前に、開発者が想定していなかった可能性のある考慮事項を洗い出すために、Claudeに質問させること
- すべての問題を単一のメッセージで提示すべき場合（相互に影響する問題）と、順番に修正すべき場合（独立した問題）の見極め

以下のスキル：

- 自然言語による説明では一貫しない結果になる場合に、2-3個の具体的な入出力の例を提示して変換の要件を明確にする
- 実装の前に、期待される動作、エッジケース、パフォーマンス要件をカバーするテストスイートを作成し、その後テストの失敗を共有しながら反復する
- インタビューパターンを使用して、不慣れな領域でソリューションを実装する前に、設計上の考慮事項（例：キャッシュの無効化戦略、障害モード）を洗い出す
- 入力例と期待される出力を伴う具体的なテストケースを提示して、エッジケースの処理を修正する（例：移行スクリプトにおけるnull値）
- 修正が相互に影響する場合は、複数の相互に影響する問題を単一の詳細なメッセージで扱い、独立した問題の場合は順番に反復する

タスクステートメント3.6：Claude CodeをCI/CDパイプラインに統合する

以下に関する知識：

- 自動化されたパイプラインでClaude Codeを非対話モードで実行するための-p（または--print）フラグ
- CIのコンテキストで構造化出力を強制するための--output-format jsonおよび--json-schemaのCLIフラグ
- CIから呼び出されるClaude Codeにプロジェクトのコンテキスト（テスト標準、フィクスチャの規約、レビュー基準）を提供する仕組みとしてのCLAUDE.md
- セッションコンテキストの分離：コードを生成したのと同じClaudeセッションは、独立したレビューインスタンスと比べて、自身の変更をレビューする効果が低い理由

以下のスキル：

- CIで-pフラグを付けてClaude Codeを実行し、対話的な入力待ちによるハングを防ぐ
- --output-format jsonを--json-schemaとともに使用し、PRのインラインコメントとして自動投稿するための、機械で解析可能な構造化された指摘事項を生成する
- 新しいコミットの後にレビューを再実行する際に、以前のレビューの指摘事項をコンテキストに含め、重複コメントを避けるために新規または未対応の問題のみを報告するようClaudeに指示する

- 既存のテストファイルをコンテキストに提供し、テスト生成において、テストスイートですでにカバーされている重複シナリオが提案されないようにする
- テスト標準、価値のあるテストの基準、利用可能なフィクスチャをCLAUDE.mdに文書化し、テスト生成の品質を高め、価値の低いテスト出力を減らす

分野4：プロンプトエンジニアリングと構造化出力

タスクステートメント4.1：精度を高め誤検知を減らすために、明示的な基準を備えたプロンプトを設計する

以下に関する知識：

- あいまいな指示よりも明示的な基準が重要であること（例：「コメントの正確性を確認する」ではなく「主張されている動作が実際のコードの動作と矛盾する場合にのみコメントを指摘する」）
- 「保守的に判断する」や「確信度の高い指摘事項のみを報告する」といった一般的な指示が、具体的なカテゴリ別の基準と比べて精度の向上につながらない理由
- 誤検知率が開発者の信頼に与える影響：誤検知の多いカテゴリが、正確なカテゴリへの信頼まで損なうこと

以下のスキル：

- 確信度に基づくフィルタリングに頼るのではなく、報告すべき問題（バグ、セキュリティ）とスキップすべき問題（軽微なスタイル、局所的なパターン）を定義する具体的なレビュー基準を記述する
- 誤検知の多いカテゴリを一時的に無効化し、それらのカテゴリのプロンプトを改善する間に開発者の信頼を回復する
- 一貫した分類を実現するために、重大度レベルごとに具体的なコード例を伴う明示的な重大度基準を定義する

タスクステートメント4.2：出力の一貫性と品質を高めるためにfew-shotプロンプティングを適用する

以下に関する知識：

- 詳細な指示だけでは一貫しない結果になる場合に、一貫した形式で実用的な出力を得るための最も効果的な手法としてのfew-shotの例
- あいまいなケースの扱い方を示すうえでのfew-shotの例の役割（例：あいまいなリクエストに対するツール選択、ブランチレベルのテストカバレッジの不足）

- few-shotの例によって、モデルが事前に指定されたケースにのみ一致させるのではなく、新しいパターンに判断を一般化できるようになる仕組み
- 抽出タスクにおけるハルシネーションの低減に対するfew-shotの例の有効性（例：非公式な計量表現や多様な文書構造の処理）

以下のスキル：

- あいまいなシナリオ向けに、妥当と思われる代替案ではなくあるアクションが選ばれた理由の推論を示す、的を絞ったfew-shotの例を2-4個作成する
- 一貫性を実現するために、求める具体的な出力形式（場所、問題、重大度、修正案）を示すfew-shotの例を含める
- 許容されるコードパターンと真の問題を区別するfew-shotの例を提示し、一般化を可能にしながら誤検知を減らす
- few-shotの例を使用して、多様な文書構造（インライン引用と参考文献リスト、方法論のセクションと本文に埋め込まれた詳細）の正しい扱い方を示す
- 多様な形式の文書からの正しい抽出を示すfew-shotの例を追加し、必須フィールドが空またはnullで抽出される問題に対処する

タスクステートメント4.3：ツール使用とJSONスキーマを用いて構造化出力を強制する

以下に関する知識：

- スキーマに準拠した構造化出力を保証し、JSON構文エラーを排除する最も信頼性の高いアプローチとしての、JSONスキーマを用いたツール使用（tool_use）
- tool_choice: "auto"（モデルはツールを呼び出さずにテキストを返す場合がある）、"any"（モデルは必ずツールを呼び出すが、どのツールかは選択できる）、強制的なツール選択（モデルは指定された名前の特定のツールを必ず呼び出す）の違い
- ツール使用による厳密なJSONスキーマは構文エラーを排除するが、意味的なエラー（例：合計と一致しない明細項目、誤ったフィールドに入った値）は防げないこと
- スキーマ設計上の考慮事項：必須フィールドと任意フィールド、拡張可能なカテゴリのための"other"+詳細文字列パターンを伴うenumフィールド

以下のスキル：

- JSONスキーマを入力パラメータとする抽出ツールを定義し、tool_useレスポンスから構造化データを抽出する
- 複数の抽出スキーマが存在し文書の種類が不明な場合に、tool_choice: "any"を設定して構造化出力を保証する
- tool_choice: {"type": "tool", "name": "extract_metadata"}で特定のツールを強制し、エンリッチメントのステップの前に特定の抽出が確実に実行されるようにする

- ソース文書に情報が含まれていない可能性がある場合にスキーマのフィールドを任意（null許容）として設計し、モデルが必須フィールドを満たすために値を捏造するのを防ぐ
- あいまいなケース用の"unclear"などのenum値や、拡張可能な分類のための"other"+詳細フィールドを追加する
- 一貫しないソースの書式に対処するために、厳密な出力スキーマとあわせて形式の正規化ルールをプロンプトに含める

タスクステートメント4.4：抽出品質のための検証、リトライ、フィードバックループを実装する

以下に関する知識：

- エラーフィードバック付きリトライ：リトライ時に具体的な検証エラーをプロンプトに追加し、モデルを修正へと導くこと
- リトライの限界：必要な情報がそもそもソース文書に存在しない場合（形式や構造のエラーの場合とは異なり）、リトライは効果がないこと
- フィードバックループの設計：どのコード構造が指摘事項を引き起こすかを追跡し（detected_patternフィールド）、却下パターンの体系的な分析を可能にすること
- 意味的な検証エラー（値の合計が合わない、フィールドの配置が誤っている）とスキーマ構文エラー（ツール使用によって排除される）の違い

以下のスキル：

- モデルが自己修正できるように、元の文書、失敗した抽出結果、具体的な検証エラーを含むフォローアップリクエストを実装する
- リトライが効果を持たない場合（例：提供されていない外部文書にしか情報が存在しない）と成功する場合（形式の不一致、構造的な出力エラー）を見極める
- 構造化された指摘事項にdetected_patternフィールドを追加し、開発者が指摘事項を却下した際の誤検知パターンの分析を可能にする
- 自己修正のための検証フローを設計する：不一致を検出するために"stated_total"とあわせて"calculated_total"を抽出し、整合しないソースデータに対して"conflict_detected"ブール値を追加する

タスクステートメント4.5：効率的なバッチ処理戦略を設計する

以下に関する知識：

- Message Batches API：50%のコスト削減、最大24時間の処理ウィンドウ、レイテンシSLAの保証なし

- バッチ処理は、ノンブロッキングでレイテンシを許容できるワークロード（夜間レポート、週次監査、夜間のテスト生成）に適しており、ブロッキングなワークフロー（マージ前チェック）には適していないこと
- バッチAPIは、単一のリクエスト内でのマルチターンのツール呼び出しをサポートしていないこと（リクエストの途中でツールを実行して結果を返すことはできない）
- バッチのリクエストとレスポンスのペアを対応付けるためのcustom_idフィールド

以下のスキル：

- ワークフローのレイテンシ要件にAPIのアプローチを合わせる：ブロッキングなマージ前チェックには同期API、夜間や週次の分析にはバッチAPI
- SLAの制約に基づいてバッチ送信の頻度を計算する（例：24時間のバッチ処理で30時間のSLAを保証するための4時間のウィンドウ）
- バッチの失敗に対処する：失敗した文書（custom_idで特定）のみを、適切な修正（例：コンテキスト制限を超えた文書のチャンク分割）を加えて再送信する
- 大量のバッチ処理を行う前にサンプルセットでプロンプトを改良し、初回の成功率を最大化して、反復的な再送信のコストを削減する

タスクステートメント4.6：マルチインスタンスおよびマルチパスのレビューアーキテクチャを設計する

以下に関する知識：

- セルフレビューの限界：モデルは生成時の推論コンテキストを保持しているため、同じセッション内では自身の判断を疑いにくくなること
- 独立したレビューインスタンス（以前の推論コンテキストを持たない）は、セルフレビューの指示や拡張思考よりも、微妙な問題を見つけるうえで効果的であること
- マルチパスレビュー：注意の分散や矛盾した指摘事項を避けるために、大規模なレビューをファイル単位の局所的な分析パスとファイル横断の統合パスに分割すること

以下のスキル：

- 生成側の推論コンテキストを持たない、別の独立したClaudeインスタンスを使用して、生成されたコードをレビューする
- 大規模な複数ファイルのレビューを、局所的な問題に焦点を絞ったファイル単位のパスと、ファイル横断のデータフロー分析のための別個の統合パスに分割する
- モデルが各指摘事項とあわせて確信度を自己申告する検証パスを実行し、較正されたレビューのルーティングを可能にする

分野5：コンテキスト管理と信頼性

タスクステートメント5.1：長いやり取りを通じて重要な情報を保持するために会話コンテキストを管理する

以下に関する知識：

- 段階的な要約のリスク：数値、パーセンテージ、日付、顧客が述べた期待があいまいな要約に圧縮されてしまうこと
- 「lost in the middle」効果：モデルは長い入力の冒頭と末尾の情報は確実に処理するが、中間部分の指摘事項を見落とす場合があること
- ツールの結果がコンテキストに蓄積し、関連性に見合わない量のトークンを消費する仕組み（例：関連するのは5個だけなのに、注文照会ごとに40+個のフィールドが返される）
- 会話の一貫性を維持するために、後続のAPIリクエストで完全な会話履歴を渡すことの重要性

以下のスキル：

- 取引に関する事実（金額、日付、注文番号、ステータス）を、要約された履歴の外に置かれ各プロンプトに含まれる永続的な「ケースの事実情報」ブロックに抽出する
- 複数の問題を扱うセッションのために、構造化された問題データ（注文ID、金額、ステータス）を抽出して別のコンテキストレイヤーに永続化する
- 冗長なツール出力がコンテキストに蓄積する前に、関連するフィールドのみに切り詰める（例：注文照会から返品に関連するフィールドのみを残す）
- 位置による影響を軽減するために、集約した入力の冒頭に主要な指摘事項の要約を配置し、詳細な結果を明示的なセクション見出しで整理する
- 下流での正確な統合を支えるために、サブエージェントに対して構造化出力にメタデータ（日付、情報源の場所、方法論上のコンテキスト）を含めることを要求する
- 下流のエージェントのコンテキスト予算が限られている場合に、冗長なコンテンツや推論チェーンではなく構造化データ（主要な事実、引用、関連性スコア）を返すように上流のエージェントを変更する

タスクステートメント5.2：効果的なエスカレーションとあいまいさの解消のパターンを設計する

以下に関する知識：

- 適切なエスカレーションのトリガー：顧客による人間の担当者への要求、ポリシーの例外や空白（複雑なケースだけではない）、有意義な進展が得られない状況
- 顧客が明示的に要求した場合に直ちにエスカレーションすることと、問題が単純な場合に解決を申し出ることの違い

- 感情に基づくエスカレーションや自己申告の確信度スコアが、実際のケースの複雑さを示す指標として信頼できない理由
- 複数の顧客が一致した場合に、ヒューリスティックによる選択ではなく、確認（追加の識別情報の要求）が必要となること

以下のスキル：

- エスカレーションすべき場合と自律的に解決すべき場合を示すfew-shotの例を伴う明示的なエスカレーション基準を、システムプロンプトに追加する
- 人間の担当者を求める顧客の明示的な要求には、先に調査を試みることなく直ちに応じる
- 問題がエージェントの対応可能な範囲内である場合は、不満を受け止めつつ解決を申し出て、顧客が希望を重ねて伝えた場合にのみエスカレーションする
- 顧客の具体的な要求についてポリシーがあいまいであるか何も定めていない場合にエスカレーションする（例：ポリシーが自社サイトでの価格調整のみを扱っている場合の、競合他社との価格マッチング）
- ツールの結果が複数の一致を返した場合に、ヒューリスティックに基づいて選択するのではなく、追加の識別情報を尋ねるようエージェントに指示する

タスクステートメント5.3：マルチエージェントシステム全体でのエラー伝播戦略を実装する

以下に関する知識：

- コーディネーターによる賢明な回復判断を可能にするものとしての、構造化されたエラーコンテキスト（失敗の種類、試行したクエリ、部分的な結果、代替アプローチ）
- アクセスの失敗（リトライの判断が必要なタイムアウト）と有効な空の結果（一致するものがなかった成功したクエリ）の違い
- 汎用的なエラーステータス（「検索を利用できません」）が、コーディネーターから有用なコンテキストを隠してしまう理由
- エラーを黙って握りつぶすこと（空の結果を成功として返す）も、単一の失敗でワークフロー全体を終了させることも、いずれもアンチパターンである理由

以下のスキル：

- コーディネーターによる回復を可能にするために、失敗の種類、試行した内容、部分的な結果、考えられる代替案を含む構造化されたエラーコンテキストを返す
- コーディネーターが適切な判断を下せるように、エラー報告においてアクセスの失敗と有効な空の結果を区別する
- サブエージェントに一時的な障害に対するローカルでの回復を実装させ、解決できないエラーのみを、試行した内容と部分的な結果を含めて伝播させる
- どの指摘事項が十分に裏付けられているか、どのトピック領域に情報源が利用できなかったことによる不足があるかを示すカバレッジの注記を付けて、統合の出力を構造化する

タスクステートメント5.4：大規模なコードベースの探索においてコンテキストを効果的に管理する

以下に関する知識：

- 長時間のセッションにおけるコンテキストの劣化：モデルが一貫しない回答を返し始め、以前に発見した具体的なクラスではなく「典型的なパターン」に言及するようになること
- コンテキストの境界を越えて主要な発見事項を保持するためのスクラッチパッドファイルの役割
- メインエージェントが高レベルの理解を調整する一方で、冗長な探索出力を分離するためのサブエージェントへの委任
- クラッシュからの回復のための構造化された状態の永続化：各エージェントが既知の場所に状態をエクスポートし、コーディネーターが再開時にマニフェストを読み込むこと

以下のスキル：

- メインエージェントが高レベルの調整を維持する一方で、特定の問い（例：「すべてのテストファイルを見つける」、「返金フローの依存関係をたどる」）を調査するサブエージェントを起動する
- エージェントに主要な発見事項を記録するスクラッチパッドファイルを維持させ、以降の質問でそれを参照させて、コンテキストの劣化に対抗する
- 次のフェーズのサブエージェントを起動する前に、ある探索フェーズの主要な発見事項を要約し、その要約を初期コンテキストに注入する
- コーディネーターが再開時に読み込んでエージェントのプロンプトに注入する、構造化されたエージェント状態のエクスポート（マニフェスト）を用いて、クラッシュからの回復を設計する
- 長時間の探索セッション中に冗長な調査出力でコンテキストが埋まった場合に、/compactを使用してコンテキスト使用量を削減する

タスクステートメント5.5：人間によるレビューのワークフローと確信度の校正を設計する

以下に関する知識：

- 集計された精度指標（例：全体で97%）が、特定の文書の種類やフィールドにおける低いパフォーマンスを覆い隠す可能性があるというリスク
- 確信度の高い抽出におけるエラー率を測定し、新しいエラーパターンを検出するための層化無作為抽出
- レビューの注意を振り分けるための、ラベル付き検証セットを用いて校正されたフィールドレベルの確信度スコア
- 確信度の高い抽出を自動化する前に、文書の種類およびフィールドのセグメントごとに精度を検証することの重要性

以下のスキル：

- 継続的なエラー率の測定と新しいパターンの検出のために、確信度の高い抽出に対する層化無作為抽出を実装する
- 人間によるレビューを減らす前に、文書の種類およびフィールドごとに精度を分析し、すべてのセグメントで一貫したパフォーマンスが得られていることを確認する
- モデルにフィールドレベルの確信度スコアを出力させ、その後、ラベル付き検証セットを用いてレビューのしきい値を較正する
- モデルの確信度が低い抽出や、ソース文書があいまいまたは矛盾している抽出を人間によるレビューに回し、限られたレビュー担当者のキャパシティに優先順位を付ける

タスクステートメント5.6：複数の情報源の統合において情報の出所を保持し不確実性に対処する

以下に関する知識：

- 主張と情報源の対応付けを保持せずに発見事項が圧縮されると、要約のステップで情報源の帰属が失われる仕組み
- 発見事項を統合する際に統合エージェントが保持しマージしなければならない、構造化された主張と情報源の対応付けの重要性
- 信頼できる情報源の間で矛盾する統計の扱い方：一方の値を恣意的に選ぶのではなく、情報源の帰属を付けて矛盾に注記すること
- 時間に関するデータ：時間的な差異が矛盾として誤って解釈されるのを防ぐために、構造化出力に公開日や収集日を要求すること

以下のスキル：

- サブエージェントに対して、下流のエージェントが統合を通じて保持する、構造化された主張と情報源の対応付け（情報源のURL、文書名、関連する抜粋）を出力することを要求する
- 十分に確立された発見事項と異論のある発見事項を区別する明示的なセクションでレポートを構造化し、元の情報源での特徴付けと方法論上のコンテキストを保持する
- 矛盾する値を含めて明示的に注記したうえで文書の分析を完了し、統合に渡す前にどのように整合させるかをコーディネーターに判断させる
- 時間的に正しい解釈を可能にするために、サブエージェントに対して構造化出力に公開日またはデータ収集日を含めることを要求する
- すべてを画一的な形式に変換するのではなく、統合の出力においてコンテンツの種類ごとに適切な表現を用いる（財務データは表、ニュースは文章、技術的な発見事項は構造化されたリスト）

7. 準備の方法

この認定資格試験の準備として、以下に取り組んでください。

- Claude Agent SDKでエージェントを構築する：ツール呼び出し、エラー処理、セッション管理を備えた完全なエージェントループを実装します。サブエージェントの起動と、サブエージェント間でのコンテキストの受け渡しを練習します。
- 実際のプロジェクト向けにClaude Codeを設定する：設定階層を持つCLAUDE.mdをセットアップし、.claude/rules/にパス固有のルールを作成し、フロントマターオプション（context: fork、allowed-tools）を使用したカスタムスキルを構築し、少なくとも1つのMCPサーバーを統合します。
- MCPツールを設計しテストする：類似したツールを明確に区別するツールの説明を記述します。エラーカテゴリとリトライ可否のフラグを備えた構造化されたエラーレスポンスを実装します。あいまいなリクエストでツール選択の信頼性をテストします。
- 構造化データの抽出パイプラインを構築する：JSONスキーマとともにtool_useを使用し、検証とリトライのループを実装し、任意（null許容）フィールドを持つスキーマを設計し、Message Batches APIでバッチ処理を練習します。
- プロンプトエンジニアリングの手法を練習する：あいまいなシナリオ向けのfew-shotの例を記述します。誤検知を減らすための明示的なレビュー基準を定義します。大規模なコードレビュー向けのマルチパスのレビューアーキテクチャを設計します。
- コンテキスト管理のパターンを学習する：冗長なツール出力からの構造化された事実の抽出、長時間のセッション向けのスクラッチパッドファイルの実装、コンテキスト制限に対処するためのサブエージェントへの委任の設計を練習します。
- エスカレーションとヒューマン・イン・ザ・ループのパターンを復習する：エスカレーションすべき場合（ポリシーの空白、顧客の要求、進展が得られない状況）と自律的に解決すべき場合を理解します。確信度に基づくルーティングを備えた人間によるレビューのワークフローの設計を練習します。

8. 準備のための演習

以下の実践的な演習に取り組み、試験で扱われるトピックへの実践的な理解を深めてください。各演習は、1つ以上の試験分野にわたる知識を強化するように設計されています。

演習1：エスカレーションロジックを備えたマルチツールエージェントの構築

目的：ツール統合、構造化されたエラー処理、エスカレーションパターンを備えたエージェントループの設計を練習します。

手順：

- 各ツールの目的、想定される入力、境界条件を明確に区別する詳細な説明を備えたMCPツールを3-4個定義します。選択時の混同を避けるために慎重な説明が必要となる、機能が類似したツールを少なくとも2つ含めます。
- stop_reasonを確認して、ツールの実行を続けるか最終的な応答を提示するかを判断するエージェントループを実装します。"tool_use"と"end_turn"の両方の停止理由を正しく処理します。
- ツールに構造化されたエラーレスポンスを追加します：errorCategory（transient/validation/permission）、isRetryableブール値、人間が読める説明を含めます。エージェントが各エラーの種類を適切に処理すること（一時的なエラーのリトライ、ビジネスエラーのユーザーへの説明）をテストします。
- プログラムによるフックを実装し、ツール呼び出しをインターセプトしてビジネスルールを適用する（例：しきい値の金額を超える操作をブロックする）。フックが作動した場合はエスカレーションワークフローにリダイレクトする。
- 複数の要件を含むメッセージ（例：複数の問題にまたがるリクエスト）でテストし、エージェントがリクエストを分解して各要件に対応し、1つにまとまった応答に統合することを確認する。

強化される分野：分野1（エージェントアーキテクチャとオーケストレーション）、分野2（ツール設計とMCP統合）、分野5（コンテキスト管理と信頼性）

演習2：チーム開発ワークフロー向けのClaude Codeの設定

目的：複数の開発者が参加するプロジェクトを想定し、CLAUDE.mdの階層、カスタムスラッシュコマンド、パス固有のルール、MCPサーバー統合の設定を練習する。

手順：

- 汎用的なコーディング規約とテスト規約を含むプロジェクトレベルのCLAUDE.mdを作成する。プロジェクトレベルに置いた指示が、チームメンバー全員に一貫して適用されることを確認する。
- コード領域ごとに、YAMLフロントマターのglobパターンを指定した.claude/rules/ファイルを作成する（例：API規約にはpaths: ["src/api/**/*"]、テスト規約にはpaths: ["**/*.test.*"]）。一致するファイルを編集するときにだけルールが読み込まれることをテストする。
- .claude/skills/に、context: forkとallowed-toolsの制限を指定したプロジェクトスコープのスキルを作成する。スキルが分離された状態で実行され、メインの会話コンテキストを汚さないことを確認する。
- .mcp.jsonで、認証情報に環境変数展開を使用するMCPサーバーを設定する。~/.claude.jsonに個人用の実験的なMCPサーバーを追加し、両方が同時に利用できることを確認する。
- 複雑さの異なるタスクで、プランモードと直接実行を比較してテストする：単一ファイルのバグ修正、複数ファイルにまたがるライブラリ移行、有効な実装アプローチが複数ある新機能。プランモードが効果を発揮する場面を観察する。

強化される分野：分野3（Claude Codeの設定とワークフロー）、分野2（ツール設計とMCP統合）

演習3：構造化データ抽出パイプラインの構築

目的：JSONスキーマの設計、構造化出力のためのtool_useの使用、検証・リトライループの実装、バッチ処理戦略の設計を練習する。

手順：

- 抽出ツールを定義する。そのJSONスキーマには、必須フィールドと任意フィールド、"other"+詳細文字列のパターンを持つenum、ソース文書に存在しない可能性のある情報のためのnullableフィールドを含める。一部のフィールドが欠けている文書进行处理し、モデルが値を捏造せずにnullを返すことを確認する。
- 検証・リトライループを実装する：PydanticまたはJSONスキーマの検証が失敗した場合、文書、失敗した抽出結果、具体的な検証エラーを含むフォローアップリクエストを送信する。リトライで解決できるエラー（形式の不一致）と解決できないエラー（ソースに情報が存在しない）を区別して記録する。
- さまざまな形式の文書（例：本文中の引用と参考文献リスト、文章による説明と構造化された表）からの抽出を示すfew-shotの例を追加し、構造のばらつきへの対応が改善されることを確認する。
- バッチ処理戦略を設計する：Message Batches APIを使用して100件の文書をバッチで送信し、custom_idごとに失敗进行处理し、失敗した文書を修正して再送信する（例：サイズ超過の文書をチャンクに分割する）。さらに、SLAの制約に照らして合計処理時間を算出する。
- 人間によるレビューへのルーティング戦略を実装する：モデルにフィールド単位の信頼度スコアを出力させ、信頼度の低い抽出結果を人間によるレビューに回し、文書タイプ別・フィールド別に精度を分析して、性能が一貫していることを確認する。

強化される分野：分野4（プロンプトエンジニアリングと構造化出力）、分野5（コンテキスト管理と信頼性）

演習4：マルチエージェントリサーチパイプラインの設計とデバッグ

目的：サブエージェントのオーケストレーション、コンテキスト受け渡しの管理、エラー伝播の実装、出所の追跡を伴う統合処理を練習する。

手順：

- 少なくとも2つのサブエージェント（例：ウェブ検索と文書分析）に委任するコーディネーターエージェントを構築する。コーディネーターのallowedToolsに"Task"が含まれていること、および各サブエージェントが自動的なコンテキスト継承に頼らず、プロンプト内で調査結果を直接受け取ることを確認する。
- コーディネーターが1つの応答で複数のTaskツール呼び出しを発行するようにして、サブエージェントの並列実行を実装する。逐次実行と比較したレイテンシの改善を測定する。
- コンテンツとメタデータを分離する、サブエージェント用の構造化出力を設計する：各調査結果には、主張、根拠となる抜粋、ソースURL/文書名、公開日を含める。統合サブエージェントが調査結果をまとめる際に、出典の帰属を保持することを確認する。

- エラー伝播を実装する：サブエージェントのタイムアウトをシミュレートし、コーディネーターが構造化されたエラーコンテキスト（失敗の種類、試行したクエリ、部分的な結果）を受け取ることを確認する。コーディネーターが部分的な結果で処理を続行し、最終出力にカバレッジの欠落を注記できることをテストする。
- 矛盾するソースデータ（例：信頼できる2つの情報源で統計値が異なる場合）でテストする。統合出力が、どちらか一方を恣意的に選ばずに両方の値を出典の帰属とともに保持していること、また確立された調査結果と見解の分かれる調査結果を区別できるようにレポートが構成されていることを確認する。

強化される分野：分野1（エージェントアーキテクチャとオーケストレーション）、分野2（ツール設計とMCP統合）、分野5（コンテキスト管理と信頼性）

9. サンプル問題

以下のサンプル問題は、試験の形式と難易度を示すものです。これらは模擬試験から抜粋したもので、学習に役立つよう解説が付いています。

シナリオ：カスタマーサポート解決エージェント

問題1：本番環境のデータによると、12%のケースでエージェントが`get_customer`を完全にスキップし、顧客が申告した名前だけを使って`lookup_order`を呼び出しており、その結果、アカウントの誤認や誤った返金がときどき発生しています。この信頼性の問題に最も効果的に対処できる変更はどれですか。

- A. `get_customer`が検証済みの顧客IDを返すまで、`lookup_order`と`process_refund`の呼び出しをブロックする、プログラムによる前提条件を追加する。
- B. システムプロンプトを強化し、あらゆる注文操作の前に`get_customer`による顧客確認が必須であることを明記する。
- C. 顧客が自分から注文の詳細を伝えてきた場合でも、エージェントが常に最初に`get_customer`を呼び出す様子を示すfew-shotの例を追加する。
- D. 各リクエストを分析し、そのリクエストタイプに適したツールのサブセットだけを有効にするルーティング分類器を実装する。

正解：A。重要なビジネスロジックのために特定のツール実行順序が必要な場合（返金処理の前に顧客の本人確認を行うなど）、プログラムによる強制は、プロンプトベースのアプローチでは得られない決定論的な保証を提供します。選択肢BとCは、LLMが確率的に指示に従うことに依存しており、エラーが金銭的な影響をもたらす場合には不十分です。選択肢Dはツールの順序ではなくツールの利用可否に対処するもので、実際の問題とは異なります。

問題2：本番環境のログによると、ユーザーが注文について尋ねたとき（例：「注文#12345を確認して」）、エージェントは`lookup_order`ではなく`get_customer`を頻繁に呼び出しています。どちらのツールも説明が最小限で（「顧客情報を取得する」／「注文の詳細を取得する」）、似た形式の識別子を受け付けます。ツ

ル選択の信頼性を高めるための、最も効果的な最初の一手はどれですか。

- A. システムプロンプトに、正しいツール選択パターンを示すfew-shotの例を追加する。注文関連のクエリがlookup_orderにルーティングされる例を5-8件含める。
- B. 各ツールの説明を拡充し、対応する入力形式、クエリの例、エッジケース、類似ツールとの使い分けを説明する境界を含める。
- C. 各ターンの前にユーザー入力を解析し、検出したキーワードと識別子のパターンに基づいて適切なツールを事前に選択するルーティング層を実装する。
- D. 2つのツールを、任意の識別子を受け付けて問い合わせ先のバックエンドを内部で判断する単一のlookup_entityツールに統合する。

正解：B。 ツールの説明は、LLMがツールを選択する際に用いる主要な仕組みです。説明が最小限だと、モデルは類似したツールを区別するためのコンテキストを得られません。選択肢Bは、少ない労力で大きな効果が得られる修正によって、この根本原因に直接対処します。few-shotの例（A）は、根本的な問題を解決しないままトークンのオーバーヘッドを増やします。ルーティング層（C）は過剰な設計であり、LLMの自然言語理解を迂回してしまいます。ツールの統合（D）はアーキテクチャ上の選択として妥当ですが、当面の問題が不十分な説明である以上、「最初の一手」に見合う以上の労力がかかります。

問題3： エージェントの初回解決率は55%で、目標の80%を大きく下回っています。ログによると、エージェントは単純なケース（写真の証拠がある標準的な破損品交換）をエスカレーションする一方で、ポリシーの例外対応が必要な複雑な状況を自律的に処理しようとしています。エスカレーションの判断精度を改善する最も効果的な方法はどれですか。

- A. システムプロンプトに明示的なエスカレーション基準を追加し、エスカレーションすべき場合と自律的に解決すべき場合を示すfew-shotの例を添える。
- B. 各応答の前にエージェントに信頼度スコア（1-10）を自己申告させ、信頼度がしきい値を下回った場合はリクエストを自動的に人間に回す。
- C. 過去のチケットで学習させた別の分類器モデルをデプロイし、メインのエージェントが処理を始める前に、どのリクエストにエスカレーションが必要かを予測する。
- D. 感情分析を実装して顧客の不満の度合いを検出し、否定的な感情がしきい値を超えた場合に自動的にエスカレーションする。

正解：A。 few-shotの例を添えて明示的なエスカレーション基準を追加することは、判断の境界が不明確であるという根本原因に直接対処します。これは、インフラストラクチャを追加する前に取るべき、問題に見合った最初の対応です。選択肢Bは、LLMが自己申告する信頼度のキャリブレーションが不十分であるため、うまくいきません。エージェントはすでに、難しいケースで誤って自信を持っているからです。選択肢Cは過剰な設計であり、プロンプトの最適化をまだ試していない段階で、ラベル付きデータとMLインフラストラクチャを必要とします。選択肢Dはまったく別の問題を解決するものです。感情は、実際の課題であるケースの複雑さとは関連しません。

シナリオ：Claude Codeによるコード生成

問題4：チームの標準的なコードレビューチェックリストを実行する、カスタムの/reviewスラッシュコマンドを作成したいと考えています。このコマンドは、開発者がリポジトリをクローンまたはプルした時点で全員が利用できる必要があります。このコマンドファイルはどこに作成すべきですか。

- A. プロジェクトリポジトリ内の.claude/commands/ディレクトリ
- B. 各開発者のホームディレクトリにある~/.claude/commands/
- C. プロジェクトルートのCLAUDE.mdファイル
- D. commands配列を持つ.claude/config.jsonファイル

正解：A。プロジェクトスコープのカスタムスラッシュコマンドは、リポジトリ内の.claude/commands/ディレクトリに保存します。これらのコマンドはバージョン管理され、開発者がリポジトリをクローンまたはプルすると、全員が自動的に利用できるようになります。選択肢B（~/.claude/commands/）は、バージョン管理で共有されない個人用コマンドのためのものです。選択肢C（CLAUDE.md）は、コマンド定義ではなく、プロジェクトの指示とコンテキストのためのものです。選択肢Dは、Claude Codeに存在しない設定の仕組みを説明しています。

問題5：チームのモノリシックなアプリケーションをマイクロサービスに再構築する担当になりました。これには数十のファイルにわたる変更が伴い、サービスの境界とモジュールの依存関係についての判断が必要です。どのアプローチを取るべきですか。

- A. プランモードに入り、変更を加える前にコードベースを調査して依存関係を把握し、実装アプローチを設計する。
- B. 直接実行で開始して段階的に変更を加え、実装を進める中で自然なサービス境界が見えてくるようにする。
- C. 各サービスをどのように構成すべきかを正確に記した包括的な事前指示を与えたうえで、直接実行を使用する。
- D. 直接実行モードで開始し、実装中に予期しない複雑さに直面した場合にのみプランモードに切り替える。

正解：A。プランモードは、大規模な変更、複数の有効なアプローチ、アーキテクチャ上の判断を伴う複雑なタスクのために設計されています。これはまさに、モノリスからマイクロサービスへの再構築に必要なものです。変更に進み切る前に、コードベースの調査と設計を安全に行えます。選択肢Bは、依存関係の発見が遅れた場合に、コストのかかる手戻りが生じるリスクがあります。選択肢Cは、コードを調査せずに正しい構成をすでに知っていることを前提としています。選択肢Dは、複雑さが後から現れるかもしれないものではなく、要件の時点ですでに明示されていることを無視しています。

問題6：コードベースには、コーディング規約の異なる複数の領域があります。Reactコンポーネントはフックを用いた関数型スタイル、APIハンドラーは特定のエラー処理を伴うasync/await、データベースモデルはリポジトリパターンに従います。テストファイルはテスト対象のコードと並べてコードベース全体に

分散しており（例：Button.tsxの隣にButton.test.tsx）、場所に関係なくすべてのテストを同じ規約に従わせたいと考えています。Claudeがコード生成時に正しい規約を自動的に適用するようにする、最も保守しやすい方法はどれですか。

- A. .claude/rules/に、globパターンを指定するYAMLフロントマター付きのルールファイルを作成し、ファイルパスに基づいて規約を条件付きで適用する
- B. すべての規約をルートのCLAUDE.mdファイルに領域ごとの見出しを付けて集約し、どのセクションが該当するかの推測をClaudeに任せる
- C. コードの種類ごとに.claude/skills/にスキルを作成し、そのSKILL.mdファイルに該当する規約を含める
- D. 各サブディレクトリに、その領域固有の規約を記した個別のCLAUDE.mdファイルを配置する

正解：A。 選択肢Aが正解である理由は、globパターン（例：**/*.test.tsx）を指定した.claude/rules/を使うと、ディレクトリの場所に関係なく、ファイルパスに基づいて規約を自動的に適用できるためです。これは、コードベース全体に分散したテストファイルには不可欠です。選択肢Bは明示的なマッチングではなく推測に頼るため、信頼性に欠けます。選択肢Cは、スキルを手動で呼び出すか、Claudeがスキルの読み込みを選択することに頼る必要があり、ファイルパスに基づく決定論的な「自動」適用という要件と矛盾します。選択肢Dは、CLAUDE.mdファイルがディレクトリに紐づくため、多数のディレクトリに分散したファイルを容易には扱えません。

シナリオ：マルチエージェントリサーチシステム

問題7：「AIがクリエイティブ産業に与える影響」というトピックでシステムを実行したところ、各サブエージェントは正常に完了していることがわかりました。ウェブ検索エージェントは関連する記事を見つけ、文書分析エージェントは論文を正しく要約し、統合エージェントは一貫性のある出力を生成しています。しかし、最終レポートはビジュアルアートしか扱っておらず、音楽、執筆、映画制作が完全に抜け落ちています。コーディネーターのログを調べると、トピックが「デジタルアート制作におけるAI」「グラフィックデザインにおけるAI」「写真におけるAI」という3つのサブタスクに分解されていました。最も可能性の高い根本原因はどれですか。

- A. 統合エージェントに、他のエージェントから受け取った調査結果のカバレッジの欠落を特定するための指示が不足している。
- B. コーディネーターエージェントのタスク分解が狭すぎるため、サブエージェントへの割り当てがトピックの関連領域をすべてカバーできていない。
- C. ウェブ検索エージェントのクエリが十分に網羅的ではなく、クリエイティブ産業のより多くのセクターをカバーするよう拡張する必要がある。
- D. 文書分析エージェントが、過度に厳しい関連性基準のために、ビジュアル以外のクリエイティブ産業に関するソースを除外している。

正解：B。 コーディネーターのログが根本原因を直接示しています。「クリエイティブ産業」をビジュアルアートのサブタスク（デジタルアート、グラフィックデザイン、写真）だけに分解し、音楽、執筆、映画を完全に省いています。サブエージェントは割り当てられたタスクを正しく実行しており、問題は割り当てら

れた内容にあります。選択肢A、C、Dは、割り当てられた範囲内で正しく動作している下流のエージェントに誤って原因を求めています。

問題8：ウェブ検索サブエージェントが、複雑なトピックの調査中にタイムアウトします。この失敗の情報をコーディネーターエージェントにどのように戻すかを設計する必要があります。インテリジェントな復旧を最もよく可能にするエラー伝播のアプローチはどれですか。

- A. 失敗の種類、試行したクエリ、部分的な結果、考えられる代替アプローチを含む構造化されたエラーコンテキストをコーディネーターに返す。
- B. サブエージェント内に指数バックオフによる自動リトライロジックを実装し、すべてのリトライを使い切った後にのみ、汎用的な「検索を利用できません」というステータスを返す。
- C. サブエージェント内でタイムアウトをキャッチし、成功とマークした空の結果セットを返す。
- D. タイムアウト例外を、リサーチワークフロー全体を終了させるトップレベルのハンドラーにそのまま伝播させる。

正解：A。 構造化されたエラーコンテキストは、修正したクエリでリトライするか、代替アプローチを試すか、部分的な結果で続行するかといった、インテリジェントな復旧の判断に必要な情報をコーディネーターに与えます。選択肢Bの汎用的なステータスは、価値のあるコンテキストをコーディネーターから隠し、十分な情報に基づく判断を妨げます。選択肢Cは、失敗を成功としてマークすることでエラーを握りつぶし、復旧を一切できなくするうえ、リサーチの出力が不完全になるリスクがあります。選択肢Dは、復旧戦略が成功する可能性があるにもかかわらず、ワークフロー全体を不必要に終了させます。

問題9：テスト中に、統合エージェントが調査結果をまとめる際、特定の主張を検証する必要が頻繁に生じることがわかりました。現在は、検証が必要になると統合エージェントがコーディネーターに制御を戻し、コーディネーターがウェブ検索エージェントを呼び出し、その結果を添えて統合を再度呼び出します。これによりタスクごとに2-3回のラウンドトリップが追加され、レイテンシが40%増加します。評価によると、これらの検証の85%は単純なファクトチェック（日付、名前、統計）で、15%はより深い調査が必要です。システムの信頼性を維持しながらオーバーヘッドを削減する最も効果的なアプローチはどれですか。

- A. 統合エージェントに、単純な照会用にスコープを限定したverify_factツールを与え、複雑な検証は引き続きコーディネーター経由でウェブ検索エージェントに委任する。
- B. 統合エージェントに検証の必要事項をすべて蓄積させ、処理パスの最後にまとめてコーディネーターに返させる。コーディネーターはそれらを一括でウェブ検索エージェントに送る。
- C. 統合エージェントにすべてのウェブ検索ツールへのアクセスを与え、コーディネーターを経由するラウンドトリップなしに、あらゆる検証の必要に直接対応できるようにする。
- D. ウェブ検索エージェントに、統合エージェントが検証を必要としそうな内容を見越して、最初の調査中に各ソースの周辺コンテキストを余分に先回りしてキャッシュさせる。

正解：A。 選択肢Aは最小権限の原則を適用し、85%を占める一般的なケース（単純な事実確認）に必要なものだけを統合エージェントに与えつつ、複雑なケースについては既存の連携パターンを維持します。選択肢Bのバッチ化のアプローチは、統合のステップがそれ以前に検証された事実依存する場合があるため、

ブロッキングの依存関係を生みます。選択肢Cは統合エージェントに過剰な権限を与え、関心の分離に反します。選択肢Dは投機的なキャッシュに頼っており、統合エージェントが何を検証する必要があるかを確実に予測できません。

シナリオ：継続的インテグレーションのためのClaude Code

問題10：パイプラインスクリプトで`claude "Analyze this pull request for security issues"`を実行していますが、ジョブがいつまでも応答しません。ログによると、Claude Codeは対話的な入力を待っています。自動化されたパイプラインでClaude Codeを実行する正しいアプローチはどれですか。

- A. `-p`フラグを追加する：`claude -p "Analyze this pull request for security issues"`
- B. コマンドを実行する前に環境変数`CLAUDE_HEADLESS=true`を設定する
- C. `stdin`を`/dev/null`からリダイレクトする：`claude "Analyze this pull request for security issues" < /dev/null`
- D. `--batch`フラグを追加する：`claude --batch "Analyze this pull request for security issues"`

正解：A。 `-p`（または`--print`）フラグは、Claude Codeを非対話モードで実行するための、ドキュメントに記載された方法です。プロンプトを処理して結果を`stdout`に出力し、ユーザー入力を待たずに終了します。これはまさにCI/CDパイプラインに必要な動作です。その他の選択肢は、存在しない機能（`CLAUDE_HEADLESS`環境変数、`--batch`フラグ）を参照しているか、Claude Codeのコマンド構文に適切に対応していないUnixの回避策を使用しています。

問題11：チームは自動分析のAPIコストを削減したいと考えています。現在、リアルタイムのClaude呼び出しが2つのワークフローを支えています。(1)開発者がマージする前に完了しなければならない、ブロッキングのマージ前チェック、(2)翌朝のレビューのために夜間に生成される技術的負債レポートです。マネージャーは、50%のコスト削減を理由に、両方をMessage Batches APIに切り替えることを提案しています。この提案をどのように評価すべきですか。

- A. 技術的負債レポートにのみバッチ処理を使用し、マージ前チェックにはリアルタイム呼び出しを維持する。
- B. 両方のワークフローをバッチ処理に切り替え、ステータスのポーリングで完了を確認する。
- C. バッチ結果の順序の問題を避けるため、両方のワークフローでリアルタイム呼び出しを維持する。
- D. 両方をバッチ処理に切り替え、バッチに時間がかかりすぎる場合はリアルタイムにフォールバックするタイムアウトを設ける。

正解：A。 Message Batches APIは50%のコスト削減をもたらしますが、処理時間は最大24時間で、レイテンシのSLAは保証されていません。そのため、開発者が結果を待つブロッキングのマージ前チェックには適しませんが、技術的負債レポートのような夜間のバッチジョブには最適です。選択肢Bは誤りです。「多くの場合はもっと早く」完了することに頼るのは、ブロッキングのワークフローでは許容できないためです。

選択肢Cは誤解を反映しています。バッチの結果は、custom_idフィールドを使って対応付けることができます。選択肢Dは、それぞれのAPIを適切なユースケースに合わせるという、より単純な解決策があるにもかかわらず、不必要な複雑さを加えます。

問題12：あるプルリクエストが、在庫追跡モジュール全体で14ファイルを変更しています。すべてのファイルをまとめて分析する単一パスのレビューでは、結果に一貫性がありません。一部のファイルには詳細なフィードバックがある一方で他のファイルには表面的なコメントしかなく、明らかなバグが見逃され、あるファイルではあるパターンを問題として指摘しながら同じPR内の別の箇所では同一のコードを承認するといった、矛盾したフィードバックも見られます。レビューをどのように再構成すべきですか。

- A. 焦点を絞ったパスに分割する：各ファイルを個別に分析してローカルな問題を調べ、その後、ファイル間のデータフローを調べる、統合に焦点を当てた別のパスを実行する。
- B. 自動レビューを実行する前に、大きなPRを3-4ファイルの小さな提出単位に分割することを開発者に義務付ける。
- C. コンテキストウィンドウの大きい上位のモデルに切り替え、1回のパスで14ファイルすべてに十分な注意が行き届くようにする。
- D. PR全体に対して独立したレビューパスを3回実行し、3回のうち少なくとも2回で現れた問題だけを指摘する。

正解：A。 レビューを焦点を絞ったパスに分割することは、多数のファイルを一度に処理する際の注意力の希薄化という根本原因に直接対処します。ファイル単位の分析によって一貫した深さが確保され、別個の統合パスによってファイル間の問題が検出されます。選択肢Bは、システムを改善せずに開発者に負担を転嫁します。選択肢Cは、コンテキストウィンドウを大きくしても注意の質の問題は解決しないことを誤解しています。選択肢Dは、断続的にしか検出されない可能性のある問題について合意を求めることで、実際のバグの検出をかえって抑制してしまいます。

10. 試験の採点方法

Claude Certified Architect - Foundations試験は、基準準拠型の評価です。各受験者は、他の受験者との比較ではなく、固定された能力基準に照らして測定されます。合格するには、一定割合の他の受験者を上回ることではなく、ブループリントで定義された知識とスキルを示す必要があります。

合格基準。 合格点は、訓練を受けた対象分野の専門家が、最低限の能力を備えた受験者に期待される能力水準を判定する、正式な基準設定調査を通じて設定されました。スコアは100-1,000のスケール範囲で報告され、カットスコアは720です。スケールスコア方式は、難易度がわずかに異なる可能性のある複数の試験フォーム間でスコアを等化するのに役立ちます。

結果の通知。 結果は、合格または不合格のステータスと、100から1,000までのスケールスコアで報告されます。スコアレポートには、各内容分野であなたが正解した問題の割合も記載されます。セクション別のパーセンテージは、ご自身の成績を理解するために提供されるものであり、可否の判定には使用されません。合

否は総合のスケールスコアに基づいて判定されます。

11. 申し込みと予約

申し込みと予約は、Anthropic Partner AcademyとPearson VUEを通じて行います。

1. Anthropic Partner Academyで受験する試験の認定資格ページにアクセスし、試験の詳細を確認します。
2. 申し込む前に、試験ガイドをダウンロードし、「認定資格利用規約」(Certification Terms and Conditions)と「認定試験ポリシー」(Certification Exam Policy)を確認します。
3. 試験に申し込み、支払い手続きを完了します。支払い手続きの画面に表示される受験料には、ご自身のパートナーティアに適用される割引が反映されています。
4. 確認の案内に従ってPearson VUEアカウントを作成し、サインインして試験セッションを予約します。
5. 空いている日程を選び、オンライン監督付きまたはPearsonのテストセンターのいずれかを選択します。試験はPearson VUEによって実施されます。
6. 予約時刻の24時間前までは、キャンセルまたは予約変更ができます。24時間以内に行った変更では、受験料は返金されません。

12. 試験ポリシー

本人確認書類

試験当日は、有効期限内の、政府発行の有効な写真付き本人確認書類を提示する必要があります。本人確認書類の氏名は、申し込み時の氏名と完全に一致していなければなりません。申し込み時の氏名を訂正する必要がある場合は、試験を予約する前にcertifications-support@anthropic.comまでご連絡ください。

受験上の配慮

障害またはニーズを証明する書類のある受験者は、適用される法律に従い、合理的な受験上の配慮を受けられます。受験上の配慮は、試験を予約する前に申請し、Pearson VUEの承認を受ける必要があります。申請が承認されるまで、予約をしないでください。受験上の配慮はpearsonvue.com/us/en/test-takers/accommodationsで申請してください。

再受験ポリシー

合格しなかった受験者は、所定の待機期間の後に再受験できます。待機期間は不合格の回数が増えるごとに長くなり、初回の後は14日、2回目の後は30日、3回目の後は90日です。同じ試験は、任意の連続する12か月の期間内に4回まで受験できます。上限は試験ごとに適用されるため、ある試験に合格しなかったことで、別の試験に申し込めなくなることはありません。受験料は受験のたびに必要です。

無断欠席と遅刻

予約した試験に現れなかった受験者、または認められた遅刻許容時間を過ぎて到着した受験者は、受験料を没収され、改めて申し込む必要があります。キャンセルと予約変更の期限は、セクション11に記載されています。

13. 試験当日の流れと受験規則

試験は、標準化され、安全が確保された、監督付きの環境で実施されます。オンラインで受験する場合も、Pearson VUEのテストセンターで受験する場合も、資格を保有するすべての人のために資格の信頼性を守る目的で、以下の規則が適用されます。

試験中は、次のことを守る必要があります。

- オンラインで受験する場合、セッションの間ずっと試験監督者とウェブカメラの視界内にとどまる
- 作業スペースにメモ、書籍、電話、サブモニター、その他の物品を置かない
- 試験中、他のいかなる人とも連絡や会話をしない
- 試験内容を、いかなる形式でも記録、複写、撮影、複製しない

持ち込みが禁止されている物品には、携帯電話、スマートウォッチ、ヘッドホン、学習教材、あらゆる録音・録画機器が含まれます。試験監督者が提供するメモ用紙など、持ち込みが許可される物品がある場合は、Pearson VUEが指定します。

不正行為の結果。カンニング、禁止されているリソースへのアクセスの試み、試験内容の開示があった場合、結果の無効化、資格の取り消し、今後の試験の受験禁止となることがあります。

14. 機密保持および秘密保持契約

試験が始まる前に、機密保持および秘密保持契約に同意する必要があります。同意することにより、問題、解答の選択肢、シナリオを含むすべての試験内容がAnthropicの機密情報であり専有財産であること、およびそのいかなる部分も開示、複製、配布しないことに同意したことになります。契約に同意しない場合、試験セッションは終了し、返金が行われません。

15. 資格の維持と再認定

Claude Certified Architect – Foundationsの資格は、授与された日から12か月間有効です。基盤となる技術は急速に進化するため、資格保有者が最新の知識を維持できるよう、この資格には有効期間が設けられています。

期限内に更新するには、認定を受けてからの変更点を確認し、Anthropic Partner Academyで無料の監督なしのアセスメントを完了します。期限内の更新に費用はかかりません。資格が失効した場合、認定されたステータスを取り戻すには、正規の受験料で試験全体を再受験する必要があります。

試験内容が大幅に変更された場合、Anthropicは資格保有者に対し、再認定のために、更新アセスメントの完了ではなく試験全体の再受験を求めることがあります。

資格保有者には、セクション13に記載された受験規則が引き続き適用されます。

16. 受験者サポート、異議申し立て、プライバシー

サポート

申し込み時の氏名を訂正するには、certifications-support@anthropic.comまでご連絡ください。申し込み、予約、受験上の配慮、結果など、その他すべての質問については、Pearson VUEのサポート (pearsonvue.com/us/en/anthropic.html) にお問い合わせください。

異議申し立てと苦情

決定に対しては、その通知を受けた日から14日以内に異議申し立てができます。試験結果に関する懸念の場合は、試験日から14日以内です。異議申し立ては、上記のリンクからPearson VUEのサポートに提出してください。異議申し立ては、プログラムの異議申し立てポリシーに基づいて審査されます。基準設定の結果および個々の試験問題の内容は、異議申し立ての対象になりません。

プライバシー

申し込みおよび受験の際に収集される個人データは、Anthropicのプライバシーポリシー、Pearson VUEのプライバシーポリシー、および適用されるデータ保護法に従って取り扱われます。

17. 付録

テクノロジーと概念

試験に出題される可能性のあるテクノロジーと概念は、以下のとおりです：

- Claude Agent SDK — エージェント定義、エージェントループ、`stop_reason`の処理、フック（`PostToolUse`、ツール呼び出しのインターセプト）、Taskツールによるサブエージェントの生成、`allowedTools`の設定
- Model Context Protocol (MCP) — MCPサーバー、MCPツール、MCPリソース、`isError`フラグ、ツールの説明、ツールの配分、`.mcp.json`の設定、環境変数展開
- Claude Code — `CLAUDE.md`の設定階層（ユーザー/プロジェクト/ディレクトリ）、YAMLフロントマターによるパススコープ指定を用いた`.claude/rules/`、スラッシュコマンド用の`.claude/commands/`、`SKILL.md`フロントマター（`context: fork`、`allowed-tools`、`argument-hint`）を用いた`.claude/skills/`、プランモード、直接実行、`/memory`コマンド、`/compact`、`--resume`、`fork_session`、Exploreサブエージェント
- Claude Code CLI — 非対話モード用の`-p / --print`フラグ、`--output-format json`、構造化されたCI出力用の`--json-schema`
- Claude API — JSONスキーマを用いた`tool_use`、`tool_choice`のオプション（`"auto"`、`"any"`、強制的なツール選択）、`stop_reason`の値（`"tool_use"`、`"end_turn"`）、`max_tokens`、システムプロンプト
- Message Batches API — 50%のコスト削減、最大24時間の処理ウィンドウ、リクエストとレスポンスを対応付けるための`custom_id`、完了確認のためのポーリング、マルチターンのツール呼び出しは非対応
- JSON Schema — 必須フィールドと任意フィールド、enum型、nullableフィールド、`"other"`+詳細文字列のパターン、構文エラーを排除するstrictモード
- Pydantic — スキーマ検証、意味的な検証エラー、検証・リトライループ
- 組み込みツール — Read、Write、Edit、Bash、Grep、Glob — それぞれの目的と選択基準
- few-shotプロンプティング — 曖昧なシナリオに的を絞った例、形式の提示、新しいパターンへの汎化
- プロンプトチェーン — 焦点を絞ったパスへの逐次的なタスク分解
- コンテキストウィンドウの管理 — トークン予算、段階的な要約、lost-in-the-middle効果、コンテキスト抽出、スクラッチパッドファイル
- セッション管理 — セッションの再開、`fork_session`、名前付きセッション、セッションコンテキストの分離
- 信頼度スコアリング — フィールド単位の信頼度、ラベル付き検証セットによるキャリブレーション、エラー率測定のための層化サンプリング

出題範囲のトピック

以下のトピックは、試験で明示的に出題されます：

- エージェントループの実装：stop_reasonに基づく制御フロー、ツール結果の処理、ループの終了条件
- マルチエージェントオーケストレーション：コーディネーターとサブエージェントのパターン、タスク分解、サブエージェントの並列実行、反復的な改善ループ
- サブエージェントのコンテキスト管理：明示的なコンテキスト受け渡し、構造化された状態の永続化、マニフェストを用いたクラッシュからの復旧
- ツールインターフェース設計：効果的なツール説明の記述、ツールの分割と統合、曖昧さを減らすツールの命名
- MCPツールとリソースの設計：コンテンツカタログにはリソース、アクションにはツール、採用を促す説明の品質
- MCPサーバーの設定：プロジェクトスコープとユーザースコープ、環境変数展開、複数サーバーへの同時アクセス
- エラー処理と伝播：構造化されたエラーレスポンス、一時的エラー・ビジネスエラー・権限エラーの区別、エスカレーション前のローカルでの復旧
- エスカレーションの意思決定：明示的な基準、顧客の希望の尊重、ポリシーの欠落の特定
- CLAUDE.mdの設定：階層（ユーザー/プロジェクト/ディレクトリ）、@importパターン、globパターンを用いた.claude/rules/
- カスタムコマンドとスキル：プロジェクトスコープとユーザースコープ、context: fork、allowed-tools、argument-hintのフロントマター
- プランモードと直接実行：複雑さの評価、アーキテクチャ上の判断、単一ファイルの変更
- 反復的な改善：入出力の例、テスト駆動の反復、インタビューパターン、問題の逐次的な解決と並列的な解決
- tool_useによる構造化出力：スキーマ設計、tool_choiceの設定、ハルシネーションを防ぐためのnullableフィールド
- few-shotプロンプティング：曖昧なシナリオへの的の絞り込み、形式の一貫性、偽陽性の削減
- バッチ処理：Message Batches APIの適否、レイテンシ許容度の評価、custom_idによる失敗の処理
- コンテキストウィンドウの最適化：冗長なツール出力の切り詰め、構造化された事実の抽出、位置を考慮した入力の手配
- 人間によるレビューのワークフロー：信頼度のキャリブレーション、層化サンプリング、文書タイプ別・フィールド別の精度の分析
- 情報の出所：主張と出典の対応付け、時間的データの扱い、矛盾の注記、カバレッジの欠落の報告

出題範囲外のトピック

以下の関連トピックは、試験には出題されません：

- Claudeモデルのファインチューニングまたはカスタムモデルのトレーニング
- Claude APIの認証、請求、アカウント管理
- 特定のプログラミング言語やフレームワークの詳細な実装（ツールとスキーマの設定に必要な範囲を超えるもの）
- MCPサーバーのデプロイまたはホスティング（インフラストラクチャ、ネットワーク、コンテナオーケストレーション）
- Claudeの内部アーキテクチャ、トレーニングプロセス、モデルの重み
- Constitutional AI、RLHF、安全性トレーニングの方法論
- 埋め込みモデルまたはベクトルデータベースの実装の詳細
- コンピュータ使用（ブラウザ自動化、デスクトップ操作）
- ビジョン/画像分析機能
- ストリーミングAPIの実装またはサーバー送信イベント
- レート制限、クォータ、API料金の計算
- OAuth、APIキーのローテーション、認証プロトコルの詳細
- 特定のクラウドプロバイダーの設定（AWS、GCP、Azure）
- パフォーマンスベンチマークまたはモデル比較の指標
- プロンプトキャッシングの実装の詳細（存在を知っていること以上の内容）
- トークンカウントのアルゴリズムまたはトークン化の詳細

18. 文書管理

バージョン	変更の概要	日付
1.0	書式とレイアウトの更新	2026年7月
0.2	ドラフトの改訂	2026年6月
0.1	初版ドラフト	2026年2月