NID TRAINING

GitHub Copilot研修

AI駆動開発力強化 座学編

2026年7月30日(木)・オンライン(Zoom)

Givery

表示名変更のお願い

- **表示名の変更にご協力ください◎**
- **「ローマ字氏名 / 漢字氏名」という形式でお願いします。**
- **表示名の例**
    - Taro Yamada / 山田 太郎

09:00からスタートします◎

講師紹介

講師紹介

人的資本インテリジェンス部門
講師・生成AIエバンジェリスト

安田 光喜

Mitsuki Yasuda

＜プロフィール＞
公立はこだて未来大学 大学院（メディアデザイン領域）を卒業し、街づくり会社での複合商業ビルの立ち上げにおける情報インフラ整備やデザイン全般業務の責任として関与。その後デザイン事務所を立ち上げ、飲食店を中心としたデザイン業務や美術館展示のアートコンテンツ開発などを行う。また、地域ブランディングにも注力し、市主催の企画運営業務を行なった。2018年10月『小中学生向けのプログラミング教育』に注目し、函館市で初めての小中学生向けプログラミングスクール『 D-SCHOOL北海道』を立ち上げて運営。その後北海道のドラッグストア『サッポロドラッグストアー』からスクールの買収を受け、教育を専門とするグループ会社『株式会社シーラクンス』にて技術開発責任者として運営。スクール設計・運営、20以上の自治体連携（イベントや出張教室運営）、大人向けプログラミングスクールメンターなどさまざまな『次世代IT教育』に取り組む。
Dify公式資料の日本語翻訳や、SecondMeアンバサダーなど生成AI最新トレンドに注力。

得意分野／経歴

・生成AIをはじめとした世界最先端トレンド
・生成AIを用いた新規事業の開発、運営、補助
・アプリケーション開発/自動化プログラム環境構築
・教育（大学講師、研修、スクール運営など）

研修運営／講演実績

・生成AI研修
・大手企業、外資、教育機関(小中高大）でのITトレンド講演
・自治体と連携した地域教育実績
・上場企業社員対象のリスキリング支援

その他

みなさんがこれから生成AIをパートナーに新しい時代を生きていけるように、本日1日、全力でサポートしていきます！
なにかご不明点などありましたら、講師陣になんでも気軽にご質問ください。
#ClaudeCode #Codex #Cursor #Antigravity
#全国統一生成AI活用技能試験S1　#生成AIパスポート　#G検定

講師紹介

株式会社ベクターデザイン 取締役
ジェネラリスト系エンジニア

松原 孝司

Koji Matsubara

＜プロフィール＞
東芝グループのソフトウェアエンジニアとしてキャリアをスタートし、携帯電話向け組み込みソフトウェアや車載用ポータブルナビゲーションシステムの開発に従事。約10年の勤務後、東大生・東工大生が立ち上げたジョイントベンチャーの設立に参画し、AI・機械学習につながる技術を活用したSNS投稿の自動フィルタリングシステムの開発・事業化を担当する。
その後、複数のスタートアップで経験を重ね、NewsPicksのAndroidアプリ開発を行ない、エンジニア兼プレイングマネージャーとして携わる。現在は株式会社ベクターデザインの取締役として、気象観測・防災IoTシステムなどに技術面から関わる一方、複数のスタートアップでもエンジニアとして活動。
幅広い技術領域を理解し、エンジニアとデザイナー、編集、営業など異なる職種の間をつなぐ「ジェネラリスト型エンジニア」。事業やプロジェクトの「0.1を2にする」段階を前進させることを得意としている。

得意分野／経歴

・組み込み、Android、Web、インフラ、IoTを横断した開発
・新技術を素早く理解し、実務へ取り入れる技術調査
・新規事業やプロジェクトの「0.1を2にする」推進
・エンジニアと非エンジニアをつなぐ橋渡し

研修運営／講演実績

・ソフトウェア開発の実務に基づく生成AI活用支援

その他

生成AIも、正解を教えてくれる魔法の道具ではなく、一緒に考え、試し、成果を磨いていくパートナーだと思っています。
20年以上のエンジニア経験を生かし、皆さんと同じ目線で手を動かしながらサポートします。疑問や気づきがあれば、ぜひ気軽に共有してください！

講義スケジュール

No	アジェンダ名	ゴール
1	イントロダクション	研修の目的・全体像を共有し、学習文脈を把握した状態でスタートできる
2	グループワーク：AI時代のエンジニアに必要な能力	AI時代に求められるスキルをグループで議論し、自分ごととして認識できる
3	AIとの協働の仕方	AIを協働パートナーとして活用する基本的な思考法・姿勢を習得できる
4	プロンプトエンジニアリング概論＋GitHub Copilot演習	プロンプト設計の基本を理解し、GitHub Copilotを即時活用できるレベルに達する
5	コンテキストエンジニアリング概論・実践	適切な文脈をAIに与えることで出力品質が上がることを体験し、実務に応用できる
6	ワーク：AIに問いを立てる力（問いの設計力）	AIへの問いを自ら設計できる状態になり、問いの質が出力品質に直結することを実感できる
7	解説・振り返り　—　AIに問いを立てる力	午前ワークの成果をフィードバックし、問いの設計力に関する学びを定着させる
8	AIとの協働の仕方（出力を評価する力）	AIの出力を批判的に評価・検証する視点を身につけられる
9	生成AI利用におけるセキュリティとリスク	情報漏洩・著作権・ハルシネーション等のリスクを理解し、業務利用の判断基準を持てる
10	AI時代の開発スタイル	AIネイティブな開発の潮流を概観し、自分の開発スタイルをアップデートする土台を作れる
11	GitHub Copilot エージェントモードを活用した不具合調査実践	エージェントモードを使った不具合調査のワークフローを体験し、実務での活用イメージを持てる
12	AIネイティブなシステム設計	AI活用を前提としたシステム設計の考え方を理解し、設計フェーズからAIを組み込む視点を持てる
13	まとめ・振り返り　／　質疑応答・アンケート	1日の学びを整理・言語化し、明日から実践に移すネクストアクションを設定できる

本資料の二次転載禁止について

本資料の無断での二次転載・複製・再配布は禁止しています。

本資料の著作権は株式会社ギブリーに帰属します。

受講者ご本人の学習目的の範囲でご利用ください。

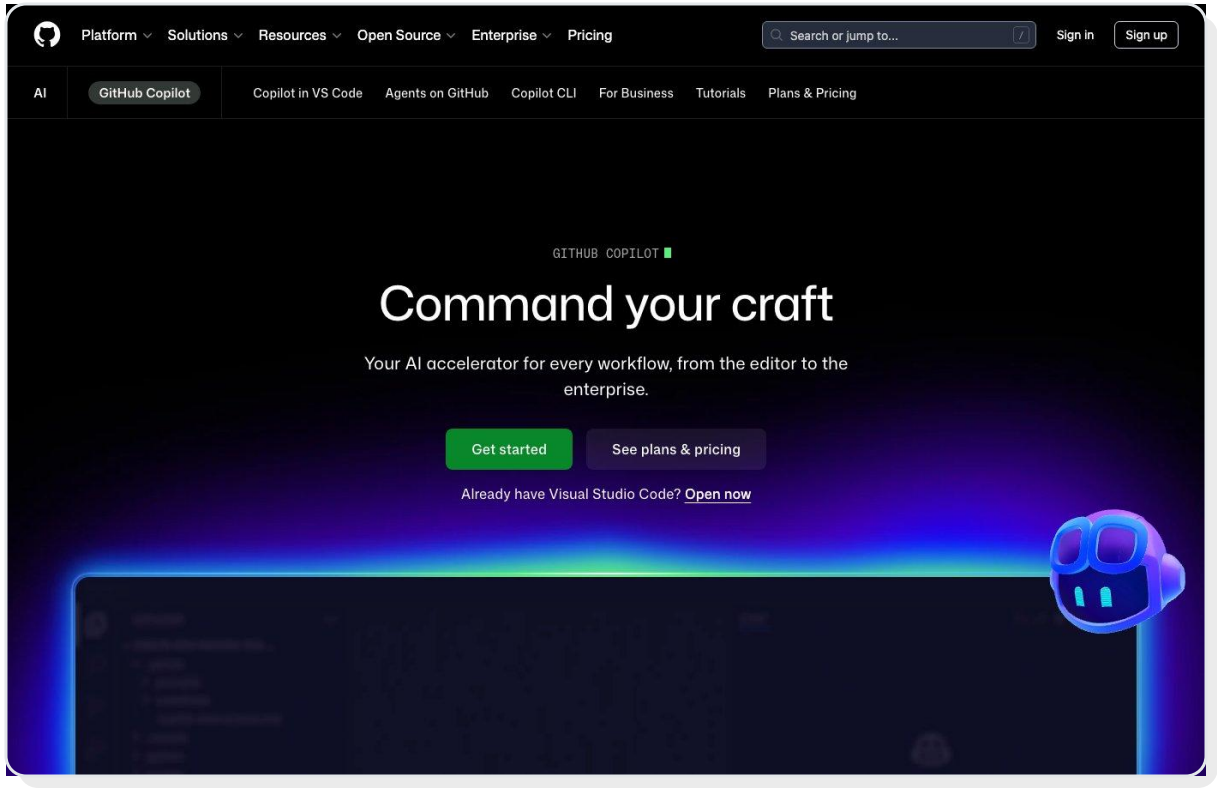
第三者への共有・SNS等への掲載はご遠慮ください。

本日の進め方

補完から、**設計・実装まで協働する**使い方へ進みます。

- **対象**
  GitHub Copilot を業務利用中の
  開発エンジニア（Java主軸）
- **時間**
  座学 [200min]・演習 [220min]
- **開催**
  2026年7月30日(木)・オンライン

  Zoom で開催します



画面: GitHub Copilot 公式サイト

※ 各ロゴは各社の商標です。

座学の地図

座学は4つのまとまりで、能力から設計まで進みます。

1-2章
能力と協働
AI時代に価値が移る3つの力と、AIとの分担の考え方。

3-4章
依頼の質を上げる
プロンプトの骨格と、渡す文脈を設計するコンテキストエンジニアリング。

5-6章
評価と安全
出力を評価する4観点と、業務利用のセキュリティの線引き。

7-9章
開発スタイルと設計
Vibe Coding から仕様駆動へ。上流から下流を一気通貫で進める設計の方法論。

SECTION 00-1

最新インパクトニュース

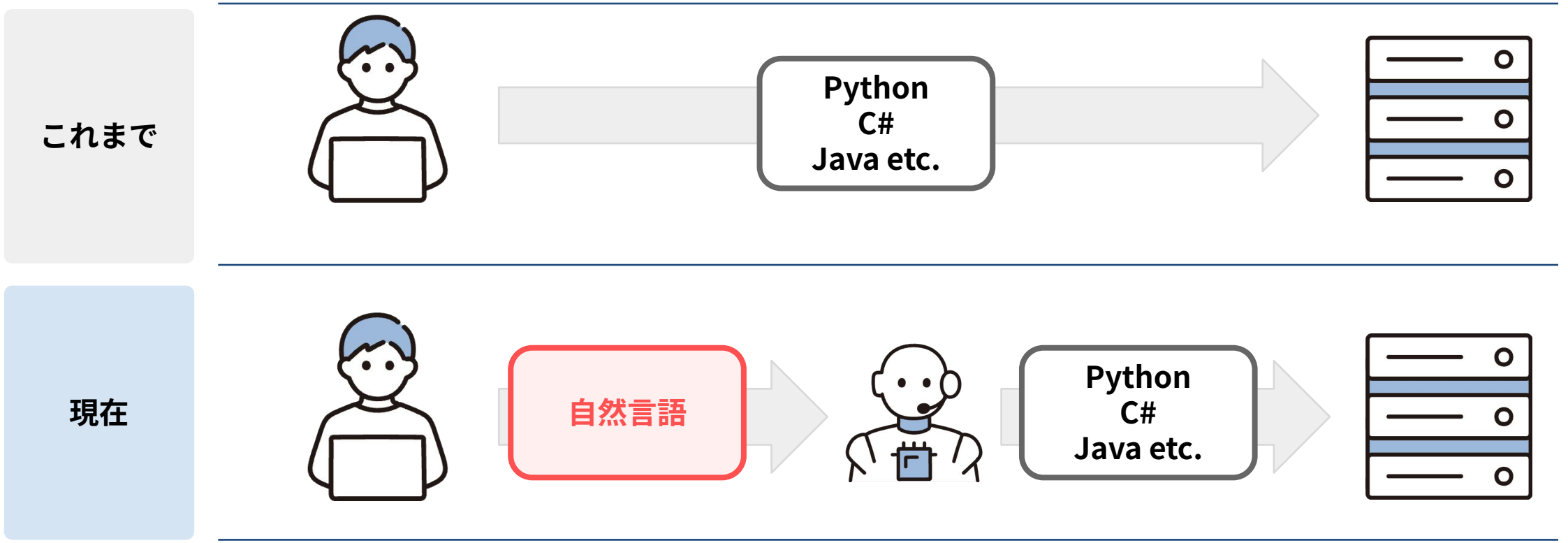
2026年4月以降の衝撃を出典つきで6本＋α

00-1

生成AIネイティブ開発の概論 - プログラミング言語の変遷

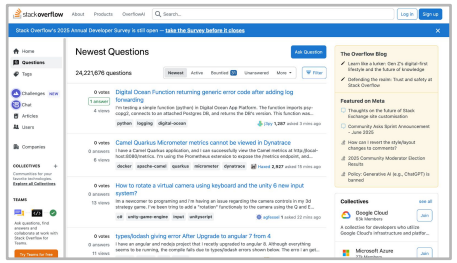
生成AIの登場により、『人工言語から自然言語によるプログラミング』へと進化しつつあります。

プログラミングの進化の概念図（人工言語→自然言語）



生成AIネイティブ開発の概論 - プログラミングにおける考え方の分類

プログラミング手法は IT技術の発展とともに、BOOP、GOOP、SOOP、CHOPと発展をしてきました。

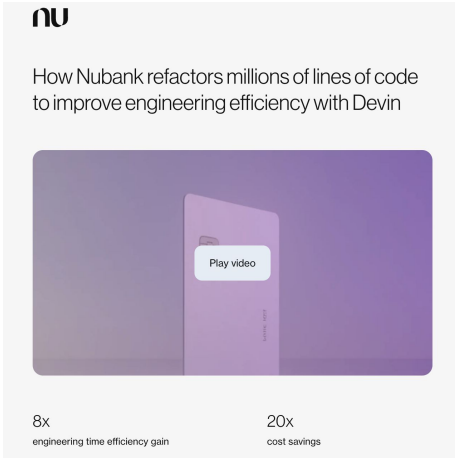
	BOOP	GOOP	SOOP	CHOP
名称	Book-Oriented Programming	Google-Oriented Programming	StackOverflow -Oriented Programming	Chat-Oriented Programming
概要	本を活用したプログラミング	Google検索を使用したプログラミング	StackOverflow communityを活用したプログラミング	AIチャットを活用したプログラミング
イメージ				

参考：[Medium『Understanding Chat-Oriented Programming (CHOP): A Paradigm Shift in Development』]

生成AIネイティブ開発の概論 - 生成AIネイティブ開発とは

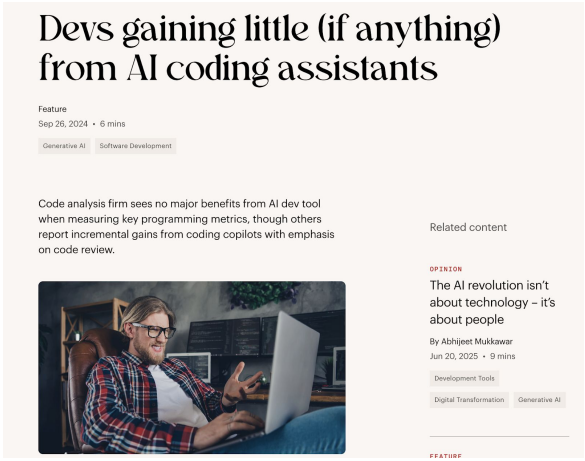
**生成AIネイティブ開発とは、AI（特に生成AI）を開発プロセスの中心に据え、
AIの力を最大限に活用しながらソフトウェアやサービスを設計・構築する新しい開発スタイルです。**

事例1（Devinの活用）



**エンジニアリング工数の節約という
点で12倍の効率改善と、
20倍以上のコスト削減を達成**

事例2



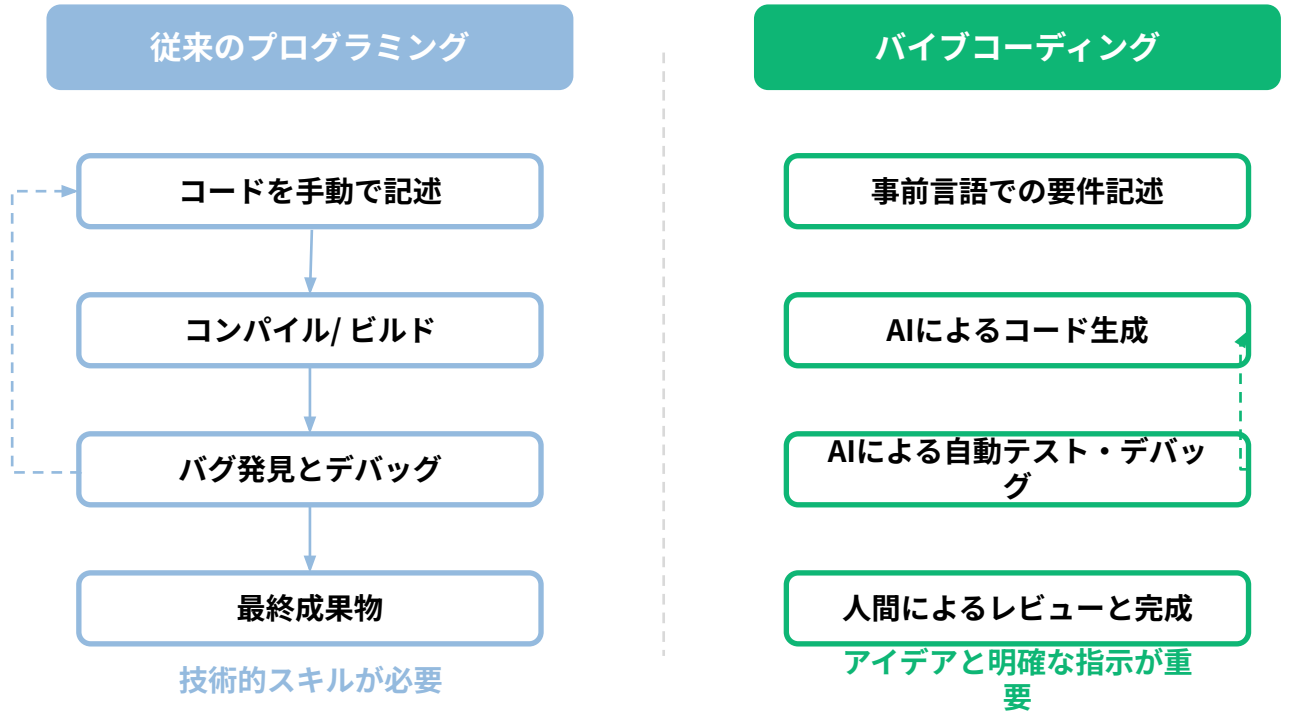
**コーディングアシスタントを活用し、
約30日かかっていた プロジェクトを
わずか24時間で完了**

参考1：Devin『How Nubank refactors millions of lines of code to improve engineering efficiency with Devin』
参考2：CIO『Devs gaining little (if anything) from AI coding assistants』

生成AIネイティブ開発の概論 - Vibe Codingとは

Vibe Condingとは技術的な厳密さよりも、アイデアや「こういうものが欲しい」というフィーリング（vibe）を重視し、AIと会話しながら開発を進める開発手法です。

従来のプログラミングとバイブコーディングの比較



参考：[ARPABLE『【2025年版】Vibe Coding革命：話すだけ開発の最前線』]

NEWS 01　AIが80年未解決の数学難問を反証

2026.05.21 ／ OpenAI発表

- 1946年提起の単位距離予想（エルデシュ予想）を反証
- 人間には描くことも難しい高次元格子を単独構築
- 数学の主要未解決問題をAIが解いた史上初の事例
- ケンブリッジのゴワーズ教授「一流誌に載る価値」

エルデシュ＝シュトラウス予想

16 個の言語版

目次 非表示

ページ先頭
背景と歴史
定式化
相異なる単位分数
負数解
計算結果
理論結果
合同恒等式
恒等式の不在
解の個数
一般化
脚注
注釈
出典
参考文献
関連項目

ページ　ノート　　閲覧　編集　履歴を表示

出典: フリー百科事典『ウィキペディア（Wikipedia）』

エルデシュ＝シュトラウス予想（エルデシュ＝シュトラウスよそう、英語: Erdős–Straus conjecture）とは、数論上の未解決問題のひとつである。予想は、2以上の全ての整数 n に対し、

$$\frac{4}{n} = \frac{1}{x} + \frac{1}{y} + \frac{1}{z}$$

を満たす正の整数 x、y、z が存在するというものである。 すなわち、$4/n$ という数が3つの単位分数の和として表せるかどうかということである。

数学の未解決問題

全ての整数 $n \geq 2$ に対して $\frac{4}{n} = \frac{1}{x} + \frac{1}{y} + \frac{1}{z}$ は正整数解を持つか？ ?

エルデシュ＝シュトラウス予想という名称は1948年にこの予想を提唱したポール・エルデシュとエルンスト・G・シュトラウス（英語版）に由来するが、その内容はより古い時代の数学と関連する。予想のような単位分数の和はエジプト数学で用いられたことから、エジプト式分数として知られる。エルデシュ＝シュトラウス予想はエルデシュによって提唱された多くの予想（英語版）のひとつであり、またディオファントス方程式に関する数学上の多くの未解決問題のひとつでもある。

全ての n に対しての解が知られているわけではないものの、ある無限等比数列の無限の多くの値には解の簡単な公式があり、これらの既知の値をスキップすることで反例の探索を高速化することが可能である。さらに、すべての合成数の反例はその素因数により小さな反例を持つはずであるから、n が素数の場合についてのみ議論すればよい。計算機を用いた探索により、少なくとも $n \leq 10^{17}$ までは予想が成り立つことが知られている。

予想を負の単位分数まで拡張すれば成立することが知られている。また、分子が5以上となる分数に対する一般化についても研究されている。

背景と歴史 [編集]

有理数を単位分数の和に展開したものをエジプト式分数と呼ぶ。この表記法は、分数を現代的な $\frac{a}{b}$（分子 a および分母 b）という形

参考： openai.com/index/model-disproves-discrete-geometry-conjecture ／ Scientific American 2026-05

NEWS 02　GPT-5.6とChatGPT Work

2026.07.09 ／ OpenAI発表

- 3種構成、旗艦Solは過去最高のコーディングモデルを主張
- トークン効率を大きく改善
- ChatGPT Workは資料作成・表計算・プレゼン生成まで担う

参考： techcrunch.com 2026-07-09 ／ CNBC 2026-07-08

NEWS 03　救急診断でAIが医師を上回る

2026.04.30 ／ Science誌掲載

- ハーバード医科大とベス・イスラエル病院の研究
- 推論モデルが救急外来の実データで医師2名を上回る精度
- トリアージから入院まで3段階すべてで匹敵・凌駕

参考： npr.org 2026-04-30 ／ Fortune 2026-05-04 ／ Science 2026

NEWS 04　継ぎ目のない30秒AI動画

2026.06.24 ／ ByteDance「Seedance 2.5」

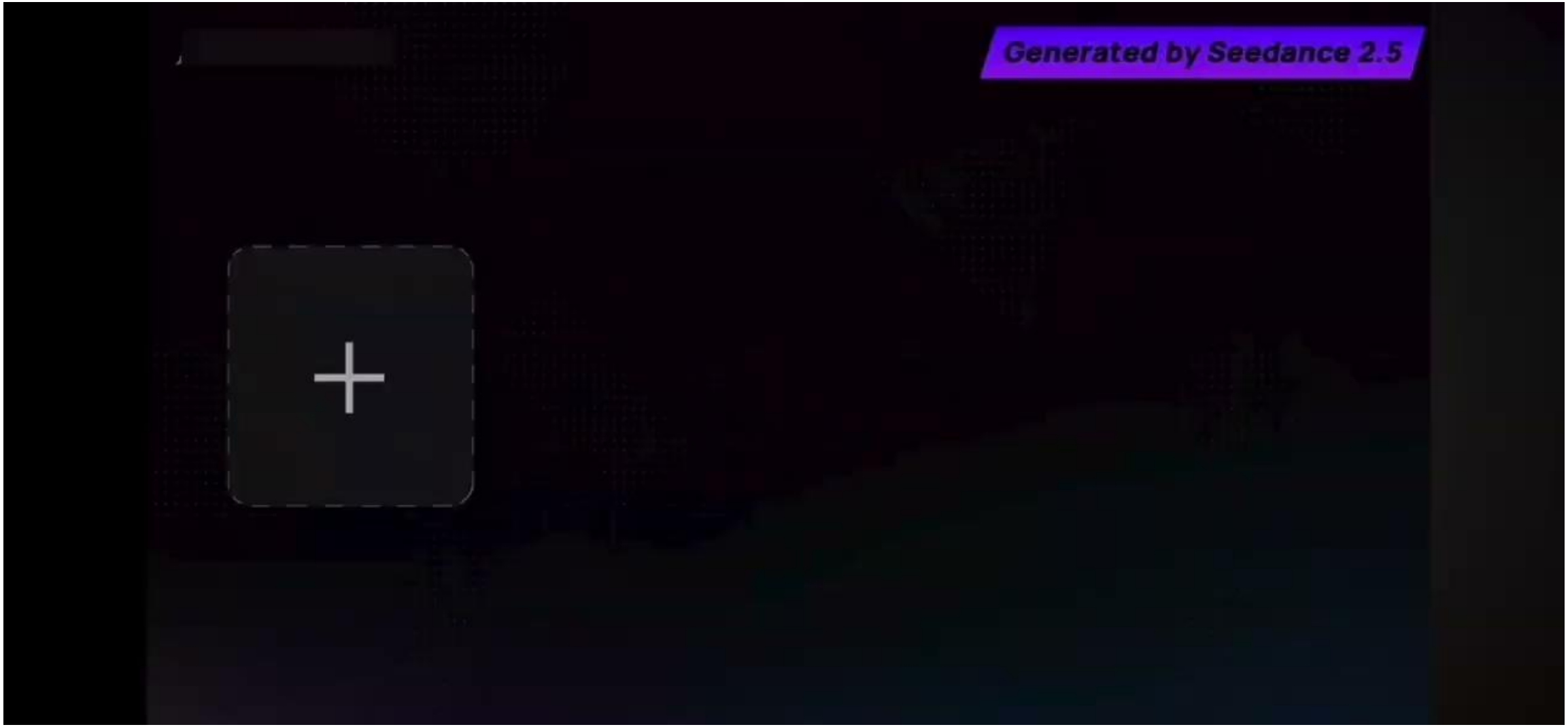
- 従来の8〜15秒・継ぎ接ぎを超え、1本の連続クリップに
- 音声と映像を共有の潜在空間で同時生成
- 参照入力は最大50点、月間4億人超のCapCutで配信予定

イメージ図（記事内容をもとに作成）　出典: techtimes.com ／ digitalapplied.com（2026-06-24）

参考： techtimes.com（bytedance-seedance-25） ／ digitalapplied.com 2026-06

NEWS 04　継ぎ目のない30秒AI動画

2026.06.24 ／ ByteDance「Seedance 2.5」



参考： techtimes.com（bytedance-seedance-25） ／ digitalapplied.com 2026-06

NEWS 05　Optimus 第3世代の実機

手の自由度は11→22に倍増

- 目標価格は3万ドル
- 将来的に年産100万～1000万台を掲げる
- 人型ロボットが試作から量産へ移る転換点

参考： techtimes.com（tesla-optimus-gen-3） ／ ifactoryapp.com 2026-06

NEWS 05　テスラ、Optimus量産工場へ転換

2026.06.09 ／ フリーモント工場の転換報道

2025年通期決算ウェブキャストの実画面（Tesla公式）

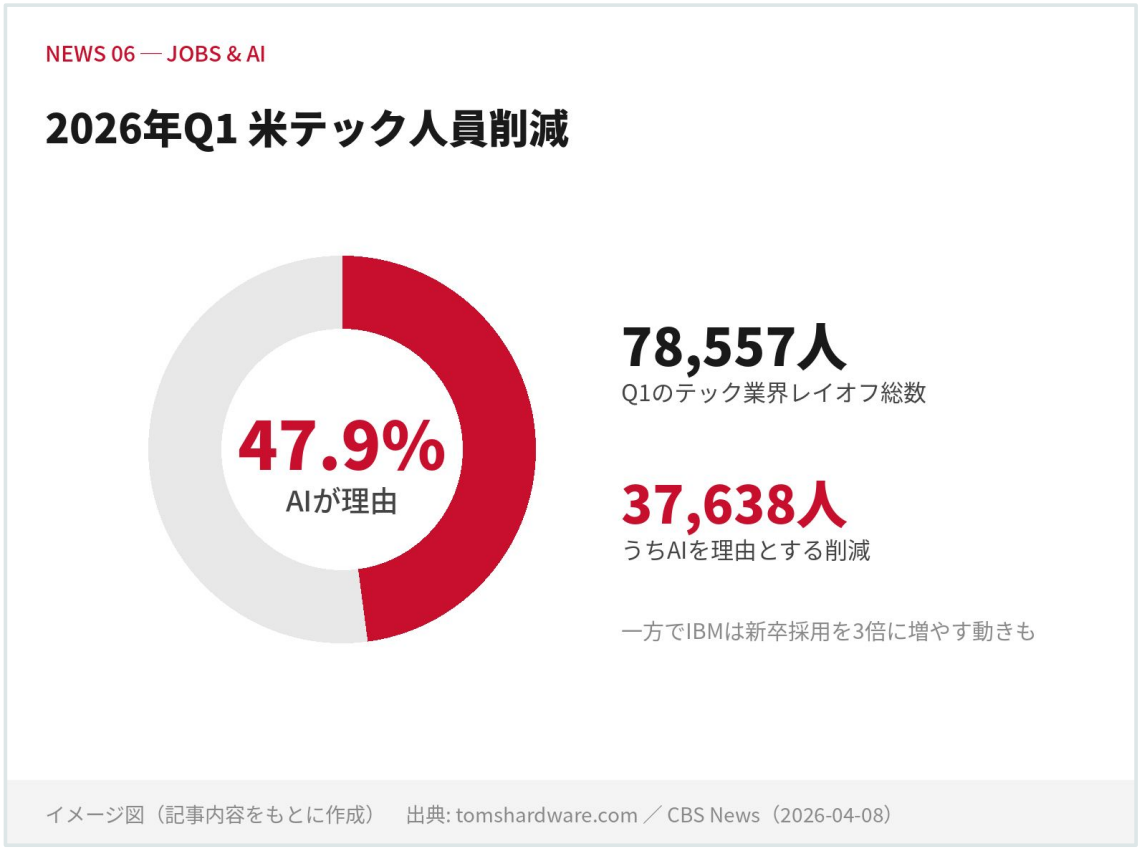
Model S/X生産を止めOptimusラインへ転換

参考： techtimes.com（tesla-optimus-gen-3） ／ ifactoryapp.com 2026-06

NEWS 06　テック解雇の47.9%がAI理由

2026.04.08 ／ 2026年1〜3月の集計

- 米テック業界で78,557人が解雇
- うち37,638人（約47.9%）がAIを理由とする削減
- IBMは新卒採用を3倍に増やす逆張りも
- AIと自分の仕事の関係を直視するテーマ



参考： tomshardware.com（tech-layoffs-q1-2026） ／ CBS News 2026

SECTION 00-2

生成AIの最新動向

主要モデルと性能比較、AIエージェント、各社の方針まで

00-2

00-2 モデル　主要ベンダーの最新フラッグシップ

2026年7月9〜12日時点の公表情報。モデル名・版は更新が速い分野

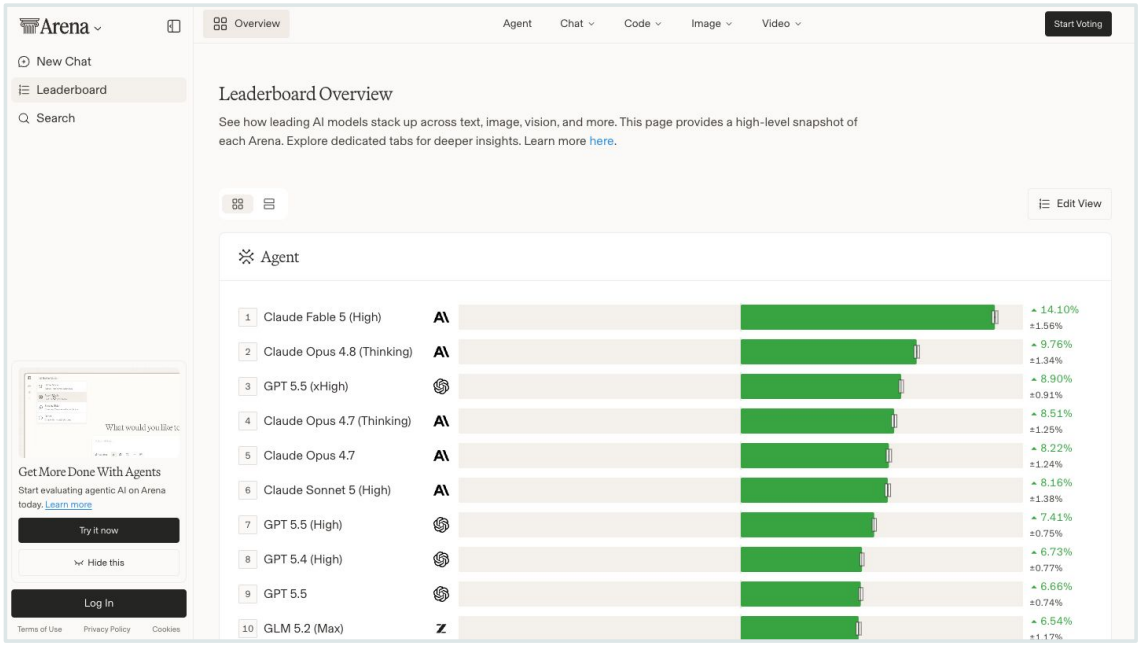
ベンダー	最新モデル	文脈長	特徴
OpenAI	GPT-5.6（Sol/Terra/Luna）	100万	コーディング最強を主張・推論4段階
Anthropic	Claude Opus 4.8 / Sonnet 5 / Fable 5	100万	コーディング・エージェントで高評価
Google	Gemini 3.1 Pro / 3.5 Flash	大規模	ネイティブ・マルチモーダル
xAI	Grok 4.5	非公開	低価格・高速、Xと連携
Meta	Muse Spark 1.1	非公開	視覚CoT・医療特化、超知能路線
DeepSeek	V4-Pro / V4-Flash	100万	オープン最強級・低コスト（MoE）
Alibaba	Qwen 3.7 Max	100万	推論・数学に強い中国系旗艦
Mistral	Mistral Large 3	非公開	欧州発・Apache2.0のオープン旗艦

参考：各社公式ドキュメント（platform.claude.com, openai.com, ai.google.dev 他）／llm-stats.com

00-2 性能比較　LMArenaリーダーボード

openlm.ai/chatbot-arena（2026-07-09）

- 利用者投票のEloレートで主要モデルを順位付け
- 1位 Claude Fable 5（Elo 1510）
- 2位 GPT-5.6 Sol、3位 Opus 4.8 Thinking
- トップ勢の差はごく僅か、順位は日々入れ替わる



参考： openlm.ai/chatbot-arena (2026-07-09)

00-2 モデルの特徴　主要モデルの強み 1/2

1 OpenAI GPT-5.6

コーディング・科学で最先端級。Sol/Terra/Lunaの3ティア、推論effort4段階

2 Anthropic Claude

コーディング・エージェントで高評価。安全性ブランドで大企業・公共に強い

3 Google Gemini

テキスト・画像・音声・動画をネイティブ対応。Flashは費用対効果が高い

4 xAI Grok 4.5

高速・低価格（$2/$6）。Xのリアルタイム情報と連携

参考： 各社公式 ／ llm-stats.com

00-2 モデルの特徴　主要モデルの強み 2/2

1 Meta Muse Spark 1.1

視覚CoT・医療特化（医師1000名超と協働）。並列推論Contemplating

2 DeepSeek V4

オープン最強級。1.6兆パラメータMoEでも低コスト、Thinking切替可

3 Qwen 3.7 Max

推論・数学が突出した中国系旗艦。100万トークン文脈・低価格帯

4 Mistral Large 3

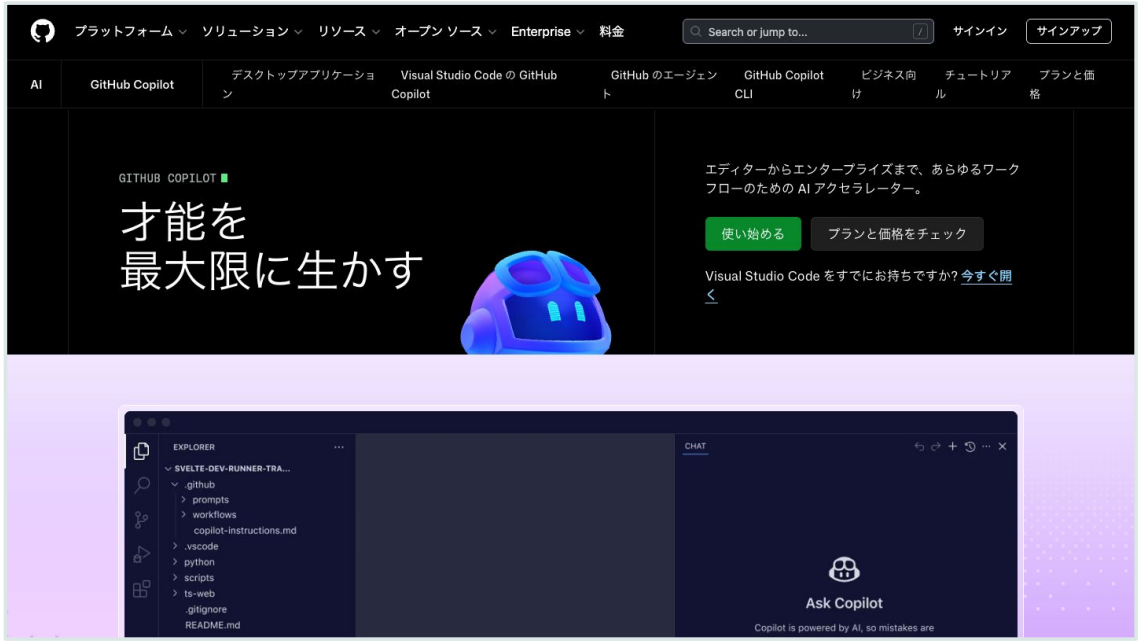
Apache2.0のオープン旗艦。40言語超、データ主権重視の用途に好適

参考： 各社公式 ／ mistral.ai

00-2 ツール　GitHub Copilot

github.com（2026年7月時点）

- IDE内で自律編集するcoding agent
- IssueからPRを非同期に作成
- 実行モデルを選べるマルチモデル化
- エディタからエンタープライズまで対応



参考： github.com

00-2 ツール　主要AIコーディングエージェント

1 GitHub Copilot

IDE内の自律編集、IssueからPRを作る非同期agent。マルチモデル化

2 Claude Code

ターミナル型。計画→編集→テスト→PRを自律実行、多階層サブエージェント

3 OpenAI Codex

CLI/IDE/クラウド横断。2026年7月にChatGPTデスクトップへ統合

4 Cursor

AIネイティブエディタ。自社モデルComposerで高速化、エージェント中心UI

5 Devin

Cognitionの自律型エンジニア。E2Eテスト・自己レビューまで実行

6 Antigravity / Jules

Gemini CLIを統合したGoogleの開発基盤。非同期でPRを提出

参考： github.com / code.claude.com / developers.openai.com ／ cursor.com / cognition.com / blog.google

00-2 潮流　書くAIから指揮するAIへ

開発者の役割は実装者から指揮者へ、レビューする力と任せる設計が重要に

書くAI

単一モデルの補完・チャット回答

並列エージェント

複数を並列に走らせ最良解を選ぶ

一般業務へ拡張

ChatGPT Workが計画→実行→納品まで自律遂行

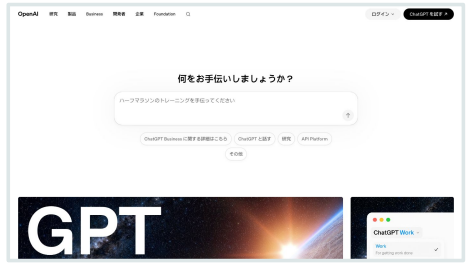
標準化

MCPとA2A v1.0、約18か月で相互運用が進展

参考： dev.to ／ machinelearningmastery.com

00-2 各社の方針　OpenAI

消費者スケールと巨額の計算投資でAGIへ最速、規模と資本で押し切る



代表製品：　ChatGPT・GPT-5.6（Sol/Terra/Luna）・OpenAI API・Stargate

主な動き

- ChatGPT 週間9億人超・有料5,000万人超
- Stargate: 4年5,000億ドルの自前インフラ
- 2026年3月 1,220億ドル調達、評価約8,520億ドル
- 2026年7月 GPT-5.6とChatGPT Workを発表

強み

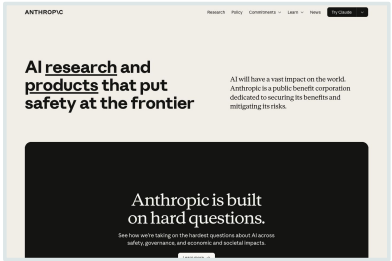
- 最大の堀は消費者スケールとデータ
- 売上は年換算 約240億ドルペース

参考： openai.com ／ techcrunch.com

00-2 各社の方針　Anthropic

安全性最優先、コーディングとエンタープライズAPIに集中する B2B・開発者ファースト

代表製品：　Claude（Opus 4.8 / Sonnet 5 / Fable 5）・Claude Code・API



主な動き

- 2026年5月 Series H $65B調達、評価約9,650億ドル
- run-rate売上 2月$14B→5月$47B超と公式発表
- Claude Opus 4.8でコーディング・エージェント強化
- Amazon・Googleが二大クラウド後ろ盾

強み

- Claude Code・Opus系が企業導入の起点
- 安全性ブランドで規制・大企業・公共に有利

参考： anthropic.com/news/series-h ／ techcrunch.com 2026-05-28

00-2 各社の方針　Google / DeepMind

チップ（TPU）からSearch・Workspace・Androidまでフルスタック垂直統合

代表製品：　Gemini 3.1 Pro / 3.5 Flash・Ironwood（TPU v7）・Antigravity・Veo

主な動き

- Gemini 3.1 ProをSearch・Vertexへ即展開
- 第7世代TPU Ironwood、前世代比約4倍
- Anthropicへ最大100万TPU供給契約
- Search/Gmail/Android等 数十億ユーザーへ配布

強み

- 学習・推論コストを内製化、AIのArm/Intel的位置
- DeepMindの基礎研究力（Gemini, AlphaFold）

参考：blog.google ／ cnbc.com（2025–2026）

00-2 各社の方針　xAI

巨大計算資源Colossus×Xのリアルタイムデータ×Muskエコシステム

代表製品：　Grok 4.5・Grok 4 Heavy・Grok Imagine・Grokipedia

主な動き

- 2025年 Xを買収、リアルタイムデータへ接続
- 2026年 SpaceXと統合、合算約1.25兆ドル規模と報道
- Colossus 約20万GPU→ギガワット級へ拡張中
- NVIDIAがSeries Eで出資、GPU取得へ大型資金

強み

- Xに統合された唯一のフロンティアAI
- 計算資源を最短で積み上げる速度

参考：cnbc.com ／ x.ai

00-2 各社の方針　Microsoft

Copilotをあらゆる製品に。自社MAI＋Foundryでモデル・オーケストレーターへ

代表製品：　M365 Copilot・Foundry・MAIシリーズ・GitHub Copilot

主な動き

- OpenAIを約27%保有、IP利用権を2032年まで延長
- 自社基盤モデルMAI-1/Voice/Imageを相次ぎ投入
- Foundryへ拡張、1,900超モデル・1,400超ツール
- 2026年の設備投資 約1,900億ドル規模

強み

- Windows/M365/Teams/GitHub/Azureの配布力
- 投資家・供給者・競合の三役で依存回避

参考： blogs.microsoft.com ／ microsoft.ai（2025–2026）

00-2 各社の方針　Amazon（AWS）

フルスタック×マルチモデル。自社NovaとAnthropicを両輪に垂直統合

代表製品：　Bedrock・Nova・Trainium3 / Inferentia・Bedrock AgentCore

主な動き

- Anthropicへ累計最大250億ドル、最大5GW確保
- Project Rainier: Trainium2 約50万→100万超
- 3nmチップTrainium3をGA、電力効率4倍
- re:Invent 2025でNova 2/Act等を発表

強み

- クラウド首位、Bedrockで複数ベンダーを単一API
- 低コスト自社チップ＋最先端モデル近接

参考： aboutamazon.com ／ cnbc.com

00-2 各社の方針　Nvidia

AIファクトリーのフルスタック供給者、CPU＋GPU＋CUDAで計算基盤を支配

代表製品：　Blackwell（GB200/GB300）・Rubin・CUDA・NIM・Grace/Vera

主な動き

- 2025年10月 時価総額が世界初の5兆ドル到達
- Q1 FY2027 総売上816億ドル（+85%）
- OpenAIへ最大1,000億ドル投資LOI ※未締結
- 次世代GPU Rubinを2026年後半に投入予定

強み

- CUDAの堀、約20年の開発者基盤
- AIデータセンターGPUで約92%（2025）

参考： nvidianews.nvidia.com ／ cnbc.com

00-2 各社の方針　その他の注目プレイヤー

1 Meta / Muse Spark

オープンウェイト＋巨額人材投資で超知能路線。Scale AIに出資

2 Apple

オンデバイス＋プライベートクラウド中心。新SiriにGemini採用と報道

3 DeepSeek

低コスト訓練＋オープンウェイトの安いフロンティア。V4-Proは1.6兆パラメータ

4 Mistral AI

欧州発の主権AI旗手。ASMLが筆頭出資、評価額約140億ドル

5 Alibaba / Baidu

Qwenが中国オープンソースを牽引、ERNIE＋AI検索で自国市場を防衛

6 Cohere / AI21 / Perplexity

企業RAG特化のCommand、長文脈のJamba、AI検索のPerplexityが急成長

参考： cnbc.com 他（2025–2026）

01

AI時代のエンジニアに求められる能力

価値の移動と、人に残る3つの力

価値の移動

コードを速く打つことの価値は、
AIの登場で**確実に下がりました**。

価値は、何を作るかを決め、AIの出力を評価し、責任を持って統合することへ移っています。

人に残る3つの力

手を動かす速さではなく、**頼み方と判断**が軸になります。

能力 01

問いを立てる力

解くべき問題を分解し、
AIに渡せる単位の問いへ
言い換える

能力 02

出力を評価する力

正しさ・要件・隠れた副作用を
判断し、自信ありげな誤りを
見抜く

能力 03

統合と責任の力

生成物を既存のコードベースへ
無理なく組み込み、最終的な
品質に責任を持つ

AIは提案者で、判断者は人のまま。
タスク単位で任せても、この分担は変わりません

生産性は数字で出始めた

国内SIでも、生産性向上が数字で出始めています。

20 %

開発工程全体の生産性向上目標

NTTデータが2025年度に約500プロジェクトへ生成AIを適用

出所: NTTデータ Data Insight「生成AIによるソフトウェア開発の変革」

50 億円

プログラミング・単体テスト工程の適用効果

日立製作所が2024年度の効果額として報告

出所: 日経クロステック「大手SIerの生成AI活用」

02

AIとの協働の考え方

人が方向を決め、AIが量をこなし、人が結果を引き受ける

AIの正しい距離感

賢い部下でも万能の専門家でもなく、文脈に弱く自信過剰な共同作業者です。

膨大な事例を学んだ相手だと捉えると、ちょうどよい距離感になります。

任せられることと、人が押さえること

何を渡すかとどう確かめるかが、人の仕事として残ります。

場面	AIに任せやすいこと	人が押さえること
実装	定型的な雛形、繰り返しの多いコード、 テストコードの素案	設計の意図、 命名や責務の妥当性
調査	エラーメッセージの解釈、 候補となる原因の列挙	どの原因が本当かの切り分け、 再現条件
ドキュメント	下書き、要約、 コメントの素案	事実の正確さ、 社外に出す表現

AIは与えられた文脈の範囲で最もそれらしい答えを返します。自社・顧客固有の事情までは知りません。

補足

頼む前の30秒仮説

「自分なら30秒でどう答えるか」を、
頼む前に一度想像してください。

その仮説があると、返ってきた出力の良し悪しを判断しやすくなります。

03

プロンプトエンジニアリングの要点

やってほしいこと・前提・制約・出力の形を、相手が誤解しない順序で並べる

依頼の骨格 4要素

良いプロンプトには、共通の骨格があります。

要素	書くこと	例
Role / Goal 役割と目的	誰として、何を達成したいかを 最初に置く	Java の保守担当として、 この例外の原因を特定したい
Context 文脈	対象のコード、データ形式、使用バージョン、再現手順など判断に要る材料	対象は InspectionService.java、 入力は data/equipment.csv
Constraint 制約	守ってほしい条件を明示する	既存の設計を壊さない、 外部ライブラリを足さない
Format 出力形式	返してほしい形を指定する	原因と修正案を分けて、修正は差分で

出所: GitHub Docs「Copilot のプロンプトエンジニアリング」

悪い依頼と良い依頼

同じ不具合でも、**依頼の書き方**で返ってくる提案が変わります。

悪い例

このコード直して

良い例

この `countByCategory` がカテゴリ別に
数えると常に `0` を返します。

原因の候補を挙げ、最小の修正案を
差分で示してください。

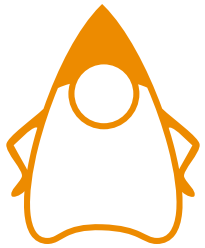
外部ライブラリは追加しないでください。

良い例は役割こそ省いていますが、対象・症状・制約・出力形式がそろっています。

午後の演習1で、この2つを実際に投げ比べます。

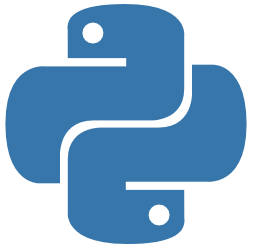
JavaとPythonで渡す情報

具体的な情報を渡すほど、AIは推測に頼らなくなります。



Java

型や例外の名前を文脈に入れる



Python

実行コマンドと想定するバージョンを渡す

言語ごとの具体情報を文脈に入れると、的外れな提案が減ります。

04

コンテキストエンジニアリング

感覚的にプロンプトを打つ進め方から、AIに渡す文脈そのものを設計する進め方へ

文脈を設計する

2025年は、プロンプトを打つから、
渡す文脈を設計するへ移った年でした。

コンテキストエンジニアリング

指示・コード・ドキュメント・会話履歴を体系的に設計・管理し、
出力の品質を安定させる取り組みです。

同じAIでも、渡す文脈が整っているかどうかで結果は大きく変わります。

出所: MIT Technology Review「From vibe coding to context engineering」

文脈設計の3つの問い

渡す文脈は、3つの問いで設計できます。

何を見せるか

無関係なコードまで渡すと、的を外した提案が増える

→

関係するファイルだけを開く

開いたファイルだけを文脈に含める

何を伝えるか

毎回書かずに済む形にしておく

→

規約・ライブラリ・観点を事前共有

命名規約や使ってよいライブラリ、レビュー観点をあらかじめ伝える

どう保つか

文脈は鮮度が落ちる

→

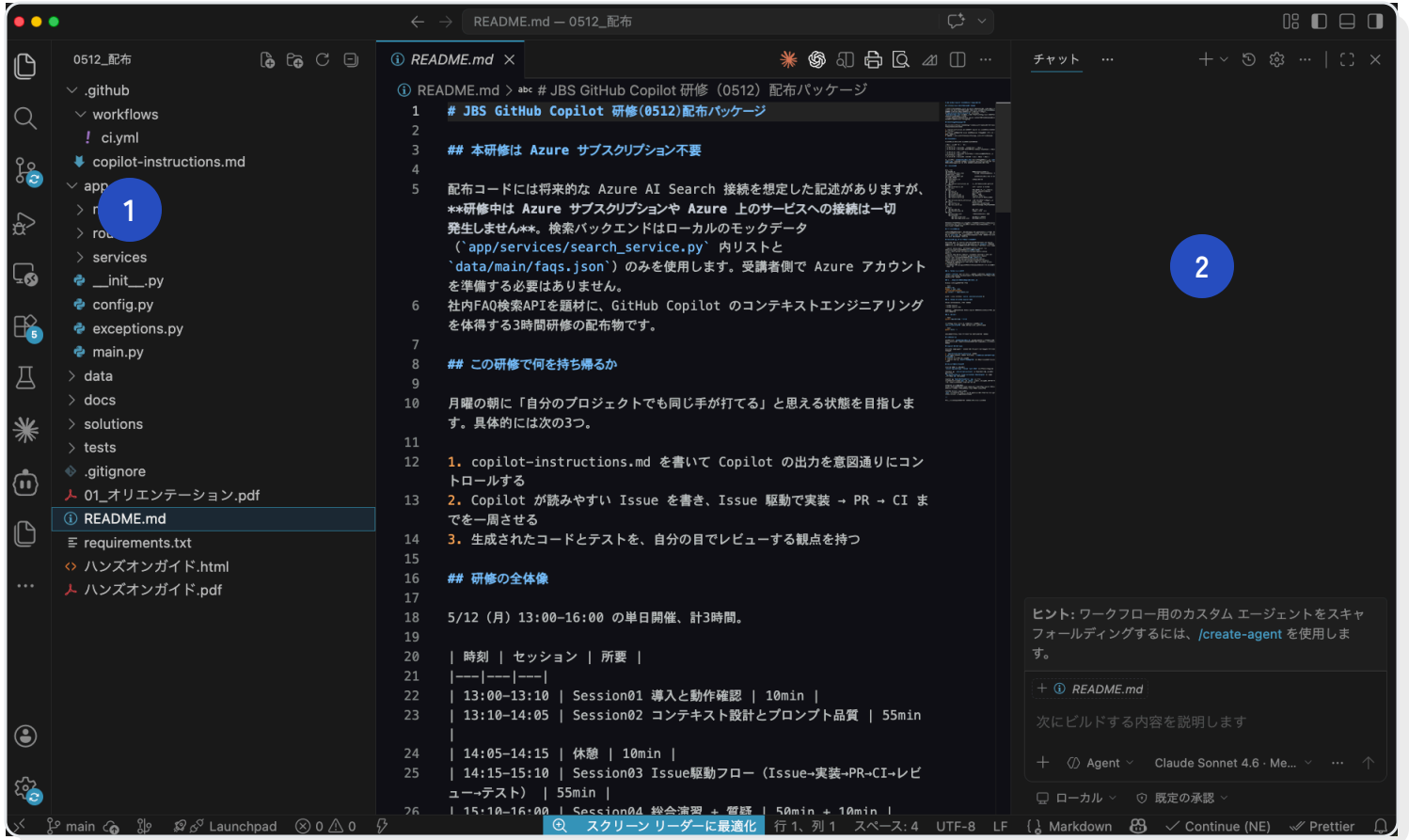
長い会話は要点を整理し直す

古い前提を引きずらせない

実演

VSCodeで文脈をつくる

関連ファイルを開き、Chatに対象ファイルを添える。この操作が文脈づくりそのものです。



画面: VSCode の Copilot Chat（実画面）

1 ファイルツリー

開いているファイルが
そのまま文脈になります。
開くファイルを絞ります。

2 Chatパネル

対象ファイルを明示的に
添えて依頼します。

05

出力を評価する力

AIの自信に流されず、根拠を自分で確かめる物差しを持つ

同じ口調で間違える

AIは、間違っているときも
正しいときと同じ口調で答えます。

評価する力とは、その自信に流されず、根拠を自分で確かめる習慣のことです。

評価の4観点

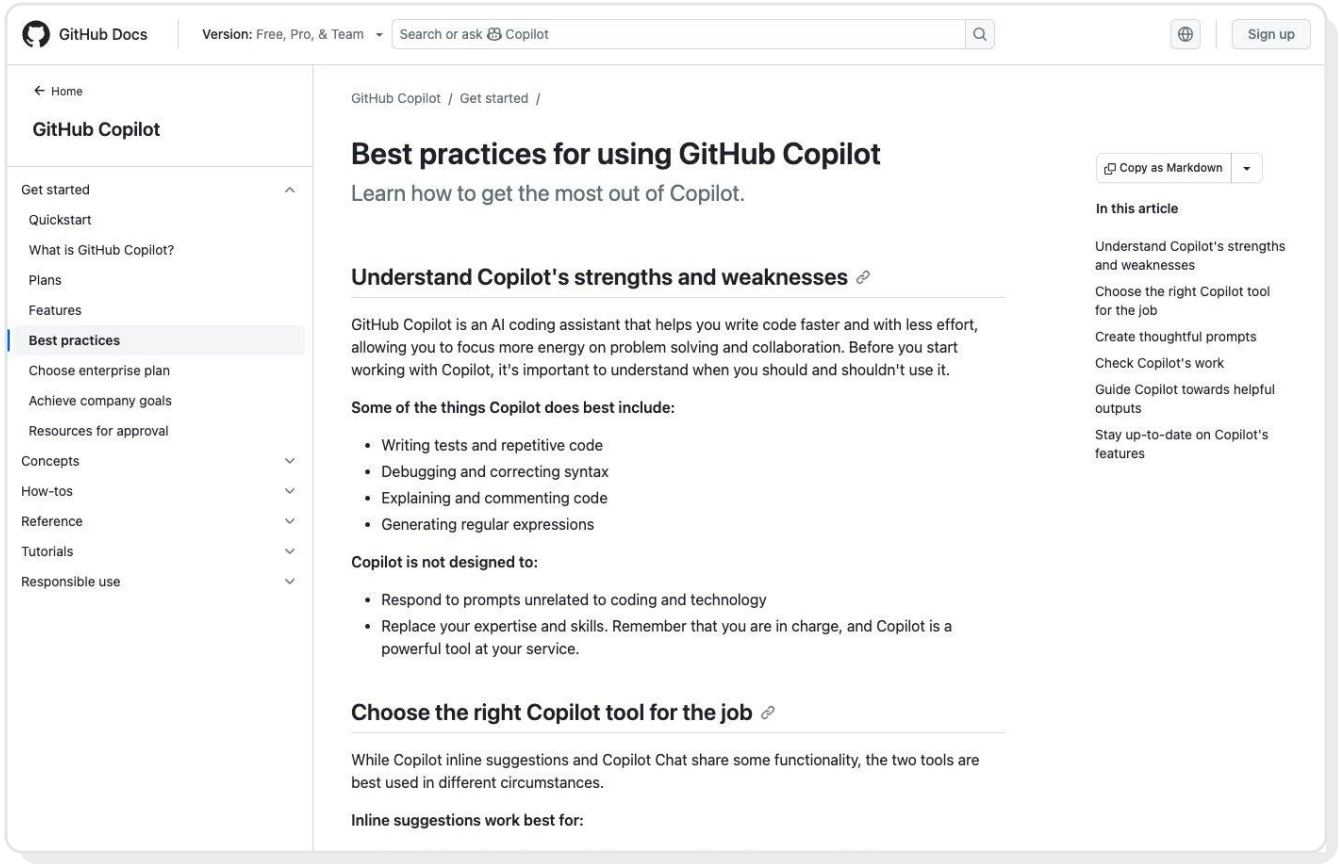
物差しを持つと、レビューの速さと確かさが両立します。

観点	確かめること
正しさ	要件どおりの結果を返すか。 境界条件（空、末尾、ゼロ件）で破綻しないか。
副作用	既存の挙動を変えていないか。 意図しない箇所まで書き換えていないか。
意図の一致	命名や構造が、自分たちの設計の言葉と合っているか。
過剰さ	頼んでいない機能やライブラリを勝手に足していないか。

とくに境界条件の誤りは、テストや実データで確かめないと表面化しません。午後の不具合調査で扱います。

公式も強みと弱みを挙げている

得意と苦手を理解して使うのは、公式も勧める基本です。



画面: GitHub Docs「Best practices for using GitHub Copilot」（出典スクショ）

境界条件の罠

文法的に正しく、一見動くコードにこそ罠が潜みます。

== で比較

```
if (category == target) {
count++;
}
```

参照の比較になるため、内容が同じ
文字列でも常に不一致になります。

equals で比較

```
if (category.equals(target)) {
count++;
}
```

文字列の内容どうしを比較するので、
カテゴリ別の件数が正しく数えられます。

動かして確かめる・テストを書く・実データで試す。

この種の誤りはAIでも人でも起こします。読んで納得しただけで通さないことが、評価する力の出発点です。

06

生成AI利用のセキュリティとリスク

扱う情報の線引きと生成物の検証を、言葉にしておく

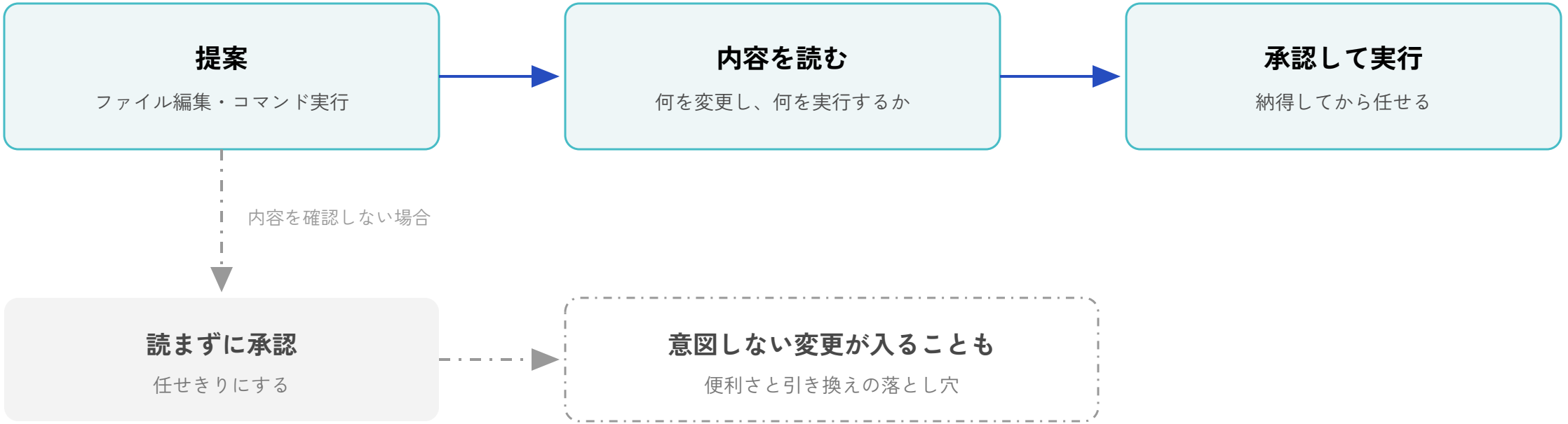
3つの論点

便利さと引き換えに、何を守るのかを言葉にしておきます。

入力する情報 顧客情報・認証情報・社外秘のコードを安易に貼らない	**社内のルールに従う** 何をどこまで渡してよいかの線引きを、社内ルールで確認する
生成物の検証 提案されたコードに脆弱な書き方が混じることがある	**入力値検証・例外処理は人が確認** 検証や例外処理の漏れは、人の目で確かめる
ライセンスと責任 生成コードの由来と依存ライブラリのライセンスを意識する	**最終的な責任は利用する側にある** 由来とライセンスを確かめてから取り込む

エージェントモードは読んでから承認

提案された操作を読んでから承認する、この一手間が安全の線です。



SSO・IP制限は情報の流出経路を絞るための仕組み
業務の Copilot はこの管理のもとで運用されています。普段どおりの管理された環境のまま参加してください。

※ 各ロゴは各社の商標です。

クイズ

理解度チェック Q1

Q1. カテゴリ別に数えるメソッドが常に0件を返します。
Java のコードで最初に疑うべき点はどれですか。

A メモリ不足

B CSVファイルが壊れている

C 文字列を == で比較している

D JDK が古い

正解と解説は次のページで確認します。

解説

理解度チェック Q1 解説

正解はCです。== は参照の比較で、文字列の内容一致は equals で確かめます。

A　メモリ不足

メモリ不足なら例外や停止として現れます。常に0件を返す症状とは結びつきません。

B　CSVファイルが壊れている

データ側の異常なら読み込みの時点で別の症状が出ます。最初に疑うのはコード側です。

C　文字列を == で比較している　正解

== は参照を比べる演算子なので、内容が同じでも常に不一致になり得ます。

D　JDK が古い

== と equals の挙動は JDK のバージョンで変わりません。

文法的に正しく、一見動きそうに見えるのが罠です。

動かして確かめる、テストを書く、実データで試す。午後の不具合調査で、まさにこの種の誤りを扱います。

クイズ

理解度チェック Q2

Q2. エージェントモードがファイル編集とコマンド実行を提案してきました。正しい対応はどれですか。

A AIの判断なのでそのまま承認する

B 変更内容と実行コマンドを読んでから承認する

C 提案はすべて却下して手作業で直す

D 実行後にログを確認する

正解と解説は次のページで確認します。

解説

理解度チェック Q2 解説

正解はBです。読んでから承認する一手間が、安全のための線になります。

A
AIの判断なのでそのまま承認する
確認せずに任せきりにすると、意図しない変更が入ることがあります。

B
変更内容と実行コマンドを読んでから承認する
何を変更し何を実行するかを把握してから任せる。便利さと安全を両立する対応です。
正解

C
提案はすべて却下して手作業で直す
内容を読めば任せられる提案まで捨てることになり、量をこなすAIの利点を失います。

D
実行後にログを確認する
実行した後では、意図しない変更が入るのを防げません。確認は承認の前に行います。

エージェントモードは IDE 内でターミナルコマンドの実行までこなします。
提案された操作の内容を読んでから承認する。この一手間が安全のための線です。

クイズ

理解度チェック Q3

Q3. 顧客から預かったコードを Copilot Chat に貼ってよいか迷いました。最初にすべきことはどれですか。

A 社内ルールと案件の契約・顧客規定を確認する

B AIに「貼ってよいか」と質問する

C 短いコードなら問題ないので貼る

D 上司に口頭で相談してから貼る

正解と解説は次のページで確認します。

解説

理解度チェック Q3 解説

正解はAです。判断の根拠は社内ルールと案件の契約・顧客規定にあります。

A **社内ルールと案件の契約・顧客規定を確認する** 正解

何をどこまで渡してよいかは、社内のルールと契約・顧客規定が決めます。

B **AIに「貼ってよいか」と質問する**

AIは自社のルールや案件の契約を知りません。判断を委ねる相手ではありません。

C **短いコードなら問題ないので貼る**

量の問題ではありません。扱う情報の性質と、預かりものである点が論点です。

D **上司に口頭で相談してから貼る**

相談は有効ですが、判断の根拠になるのはルールと契約です。まず確認が先です。

顧客情報や認証情報、社外秘のコードを安易に貼らない。

便利さと引き換えに何を守るのかを、言葉にしておくことが業務利用の前提です。

07

AI時代の開発スタイル

Vibe Coding から Agentic Coding、そして仕様を先に固める進め方へ

進め方は3段階で進化した

感覚で投げる進め方から、仕様を先に固める進め方へ流れが移っています。

業務の現場では、感覚で投げる進め方は試作までです。規模のあるコードや保守が前提のコードでは、
何を作るかを先に決め、AIに量をこなさせ、人が評価して統合する流れが安定します。

3スタイルの使い分け

今のタスクが試作か、仕様の固まった実装かで AIへの任せ方を変えます。

スタイル	進め方	向く場面
Vibe Coding	自然言語のプロンプトでAIを大まかに誘導し、 感覚的にコードを生成させる 2025年2月 Andrej Karpathy が提唱	素早い試作。規模が大きく なると破綻しやすい
Agentic Coding	詳細な仕様に対し、AIエージェントを 人の監督下で自走させて開発する	仕様が固まった 実装・調査・修正
Spec-Driven Development	詳細な仕様を先に固め、AIエージェントに その仕様へ沿って実装させる	規模のあるコード・ 保守が前提のコード

出所: MIT Technology Review「2025 in software development」 / Thoughtworks / Augment Code

08

AIネイティブなシステム設計

要件定義から実装・テストまでをAIと一気通貫で進める、設計の方法論

上流が下流を決める

曖昧な要件は、そのまま
曖昧なコードになります。

要件を振る舞いに、振る舞いをテストに落とし、その上でAIに実装を任せます。

設計を支える4方法論

作るものを実装の前に言葉やテストで固める点が共通します。

BDD 振る舞い駆動開発

期待する振る舞いをGiven-When-Thenの
シナリオで記述し、仕様兼テストとして開発を駆動。
AIに渡す要件の言語化と相性がよい。

TDD テスト駆動開発

実装前にテストを書き、通す最小実装と
リファクタリングを繰り返す。AIにテストを
起点にコードを書かせる流れに合う。

DDD ドメイン駆動設計

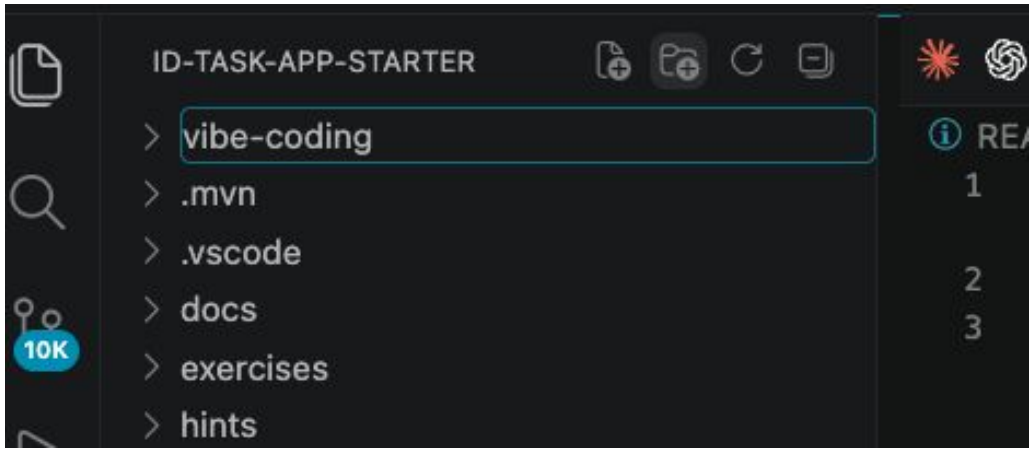
業務ドメインの言葉とモデルを中心に
構造を設計する。境界づけられたコンテキストと
ユビキタス言語が要点。

SDD 仕様駆動開発

詳細な仕様を先に固め、AIにその仕様へ
沿って実装させる。上流の精度が
下流の品質を決める。

バイブコーディング体験

GithubCopilotとバイブコーディング

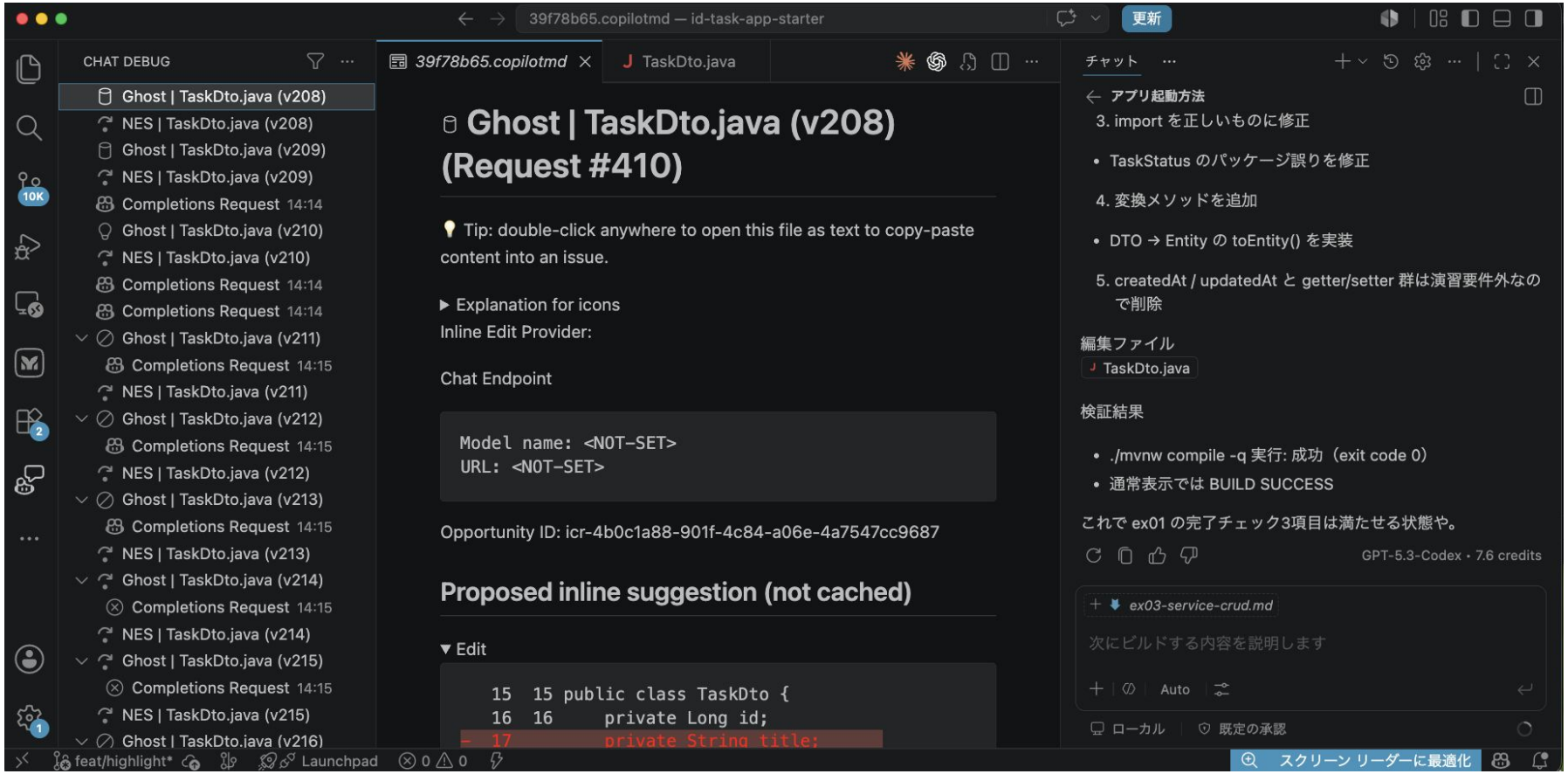


・フォルダ直下に
　『vibe-coding』というフォルダを作成
・
・GithubCopilotに
　『HTML単品ファイル形式で以下のWebアプリを開発してください
　　～（Webアプリ形式で動作するアプリの説明）～』

Copyright © Givery, Inc. All rights reserved.

Tips：ここまでの消費トークンを確認する

コマンドパレット（Ctrl+Shift+P / Cmd+Shift+P）から
Developer: Show Chat Debug View と検索して実行



Copyright © Givery, Inc. All rights reserved.

GitHub Copilot の使い方

GitHub Copilot の使い方 - ゴール

本パートのゴール	GitHub Copilot を使って生成AIネイティブ開発をスタートできる状態になる

GitHub Copilot の使い方

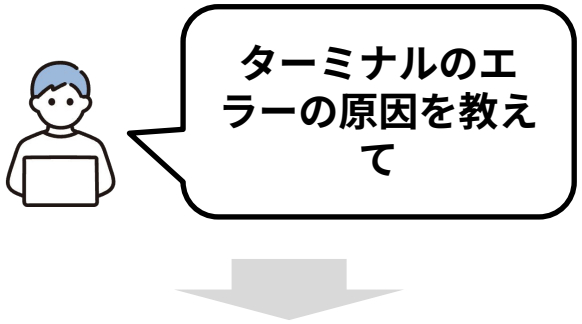
- 各種コマンド（@コマンド、#コマンド、/コマンド）の全体像のご説明
  - 『@コマンド』の説明と演習
  - 『#コマンド』の説明と演習
  - 『/コマンド』の説明と演習
- Custom Instructionsの説明と演習

GitHub Copilot の使い方

- 各種コマンド（@コマンド、#コマンド、/コマンド）の全体像のご説明
  - 『@コマンド』の説明と演習
  - 『#コマンド』の説明と演習
  - 『/コマンド』の説明と演習
- Custom Instructionsの説明と演習

GitHub Copilot の使い方 - GitHub Copilot の3種類のコマンド

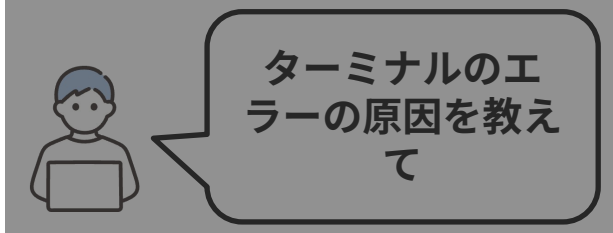
GitHub Copilotには3種類の基本コマンドがあり、それぞれ異なる用途で指示を最適化できます。

	@コマンド	#コマンド	/コマンド
概要	質問対象の スコープを指定する	Copilot に ツールの使用を指示する	特定の指示を ショートカットとして 利用する
使用例	 @vscode エディタの言語を日本語にして	 #terminalLastCommand エラーの原因を教えて	 @workspace /explain

参考：GitHub「GitHub Docs」

GitHub Copilot の使い方 - GitHub Copilot の3種類のコマンド

GitHub Copilotには3種類の基本コマンドがあり、それぞれ異なる用途で指示を最適化できます。

	@コマンド	#コマンド	/コマンド
概要	質問対象の スコープを指定する	Copilot に ツールの使用を指示する	特定の指示を ショートカットとして 利用する
使用例	 @vscode エディタの言語 を日本語にして	 #terminalLastCommand エラーの原因を教えて	 @workspace /explain

参考：GitHub「GitHub Docs」

GitHub Copilot の使い方 - @コマンド

VScode

Copilot Chat では、質問に正確なコンテキストを与えることで、より的確な回答を得られます。

Copilot Chat の @コマンド一覧と用途

コマンド名	機能内容
`@workspace`	ワークスペース全体に関する質問
`@vscode`	VS Code の使い方や設定を質問
`@terminal`	ターミナル操作に関する質問
`@github`	GitHub 固有の Copilot スキルを使用可能

参考：GitHub「GitHub Docs」

演習：GitHub Copilot の使い方 - 演習：@コマンド

VScode

演習内容

@terminalを実践してみましょう！

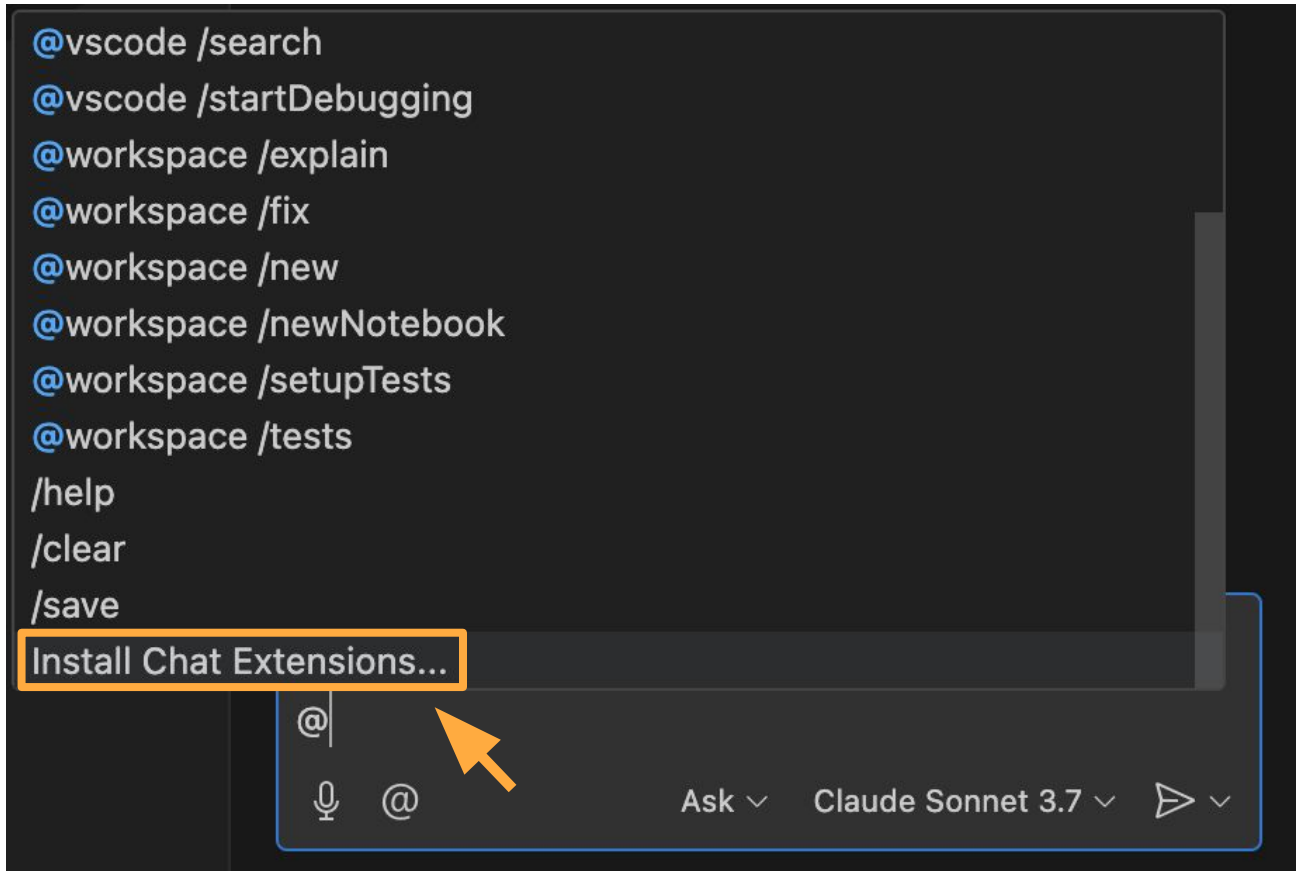
演習手順

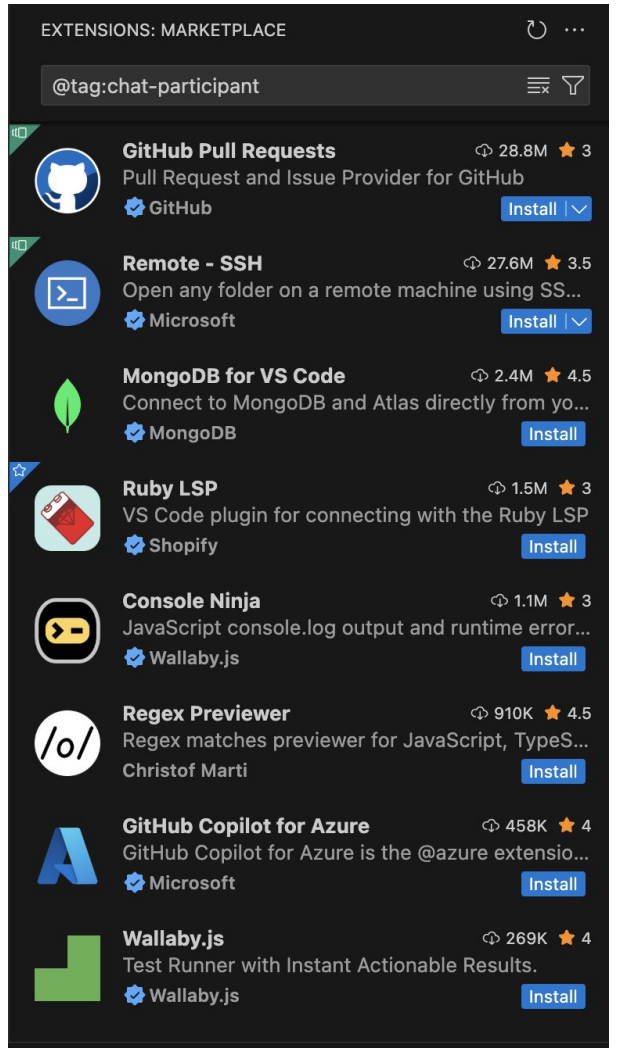
1. cd/src/work_exercise/fix をターミナルに入力
2. python exercise_bug_2.py を実行
3. エラーを確認
4. Copilotチャットで「@terminal /explain」を入力

GitHub Copilot の使い方 - @コマンドの補足

VScode

@コマンドには豊富な拡張機能があります。





GitHub Copilot の使い方 - GitHub Copilot の3種類のコマンド

GitHub Copilotには3種類の基本コマンドがあり、それぞれ異なる用途で指示を最適化できます。

	@コマンド	#コマンド	/コマンド
概要	質問対象の スコープを指定する	Copilot に ツールの使用を指示する	特定の指示を ショートカットとして 利用する
使用例	 @vscode エディタの言語 を日本語にして	 #terminalLastCommand エラーの原因を教えて	 @workspace /explain

参考：GitHub「GitHub Docs」

GitHub Copilot の使い方 - #コマンド一覧

VScode

コマンドでCopilot に使用する“ツール”を与えることができます。

#コマンド別 機能概要一覧

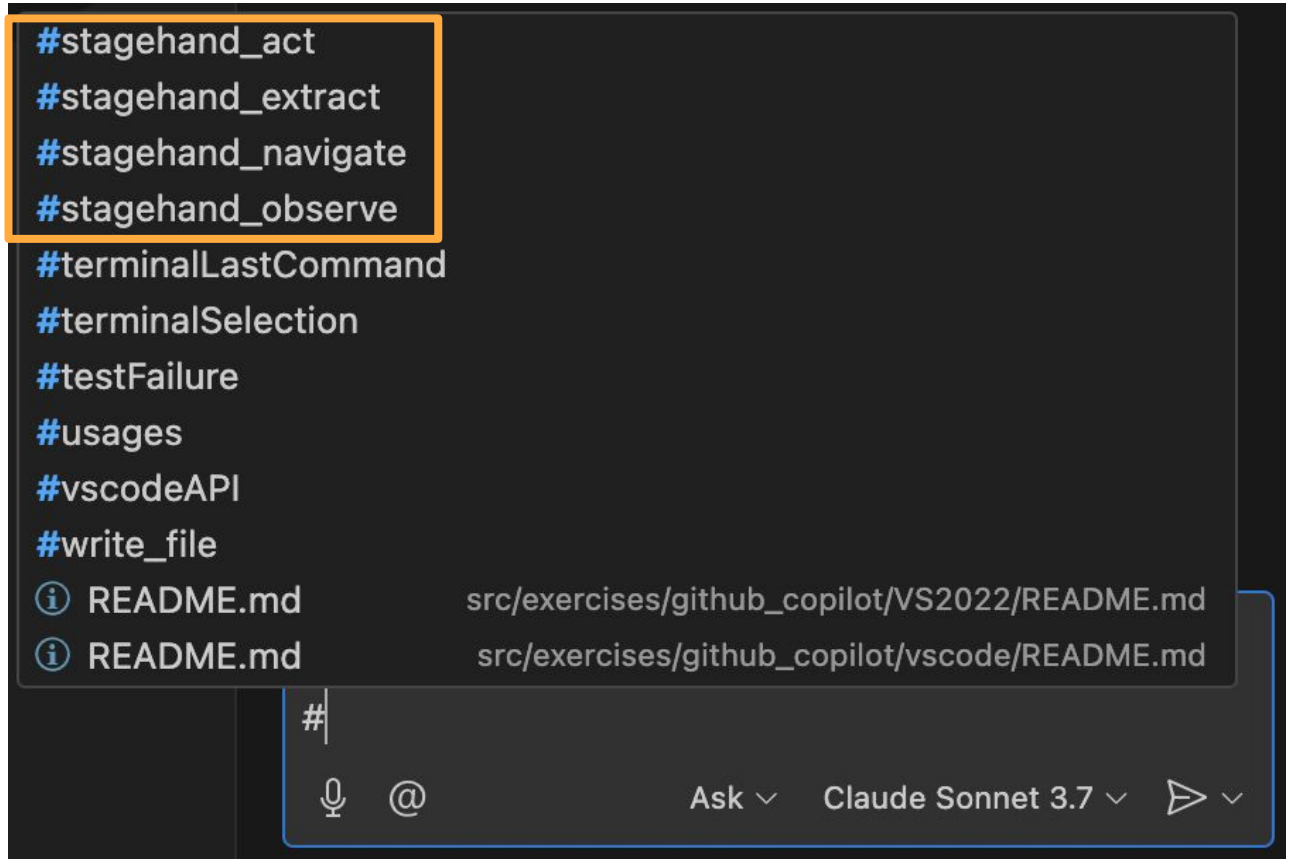
コマンド	説明
#changes	変更されたファイルの差分を取得
#codebase	ワークスペース全体を検索
#editFiles	指定したファイルを編集
#extensions	VS Code Marketplace で拡張機能を検索
#findTestFiles	テストファイル群を検索
#githubRepo	GitHub リポジトリで関連コードを検索
#new	空ディレクトリに新規ワークスペースの土台を作成
#openSimpleBrowser	Simple Browser でローカルサイトをプレビュー
#usages	シンボルの参照・定義・使用箇所を一覧取得

コマンド	説明
#problems	特定のファイルのエラーを確認
#runCommands	ターミナルでコマンドを実行
#runNotebooks	Notebook セルを実行
#runTasks	ワークスペースでタスクを実行
#searchResults	検索ビューからの結果
#terminalLastCommand	アクティブなターミナルの最後の実行コマンド
#terminalSelection	アクティブなターミナルの選択
#testFailure	前回の単体テストの失敗情報
#vscodeAPI	VS Code API リファレンスを元に回答

GitHub Copilot の使い方 - #コマンドの補足

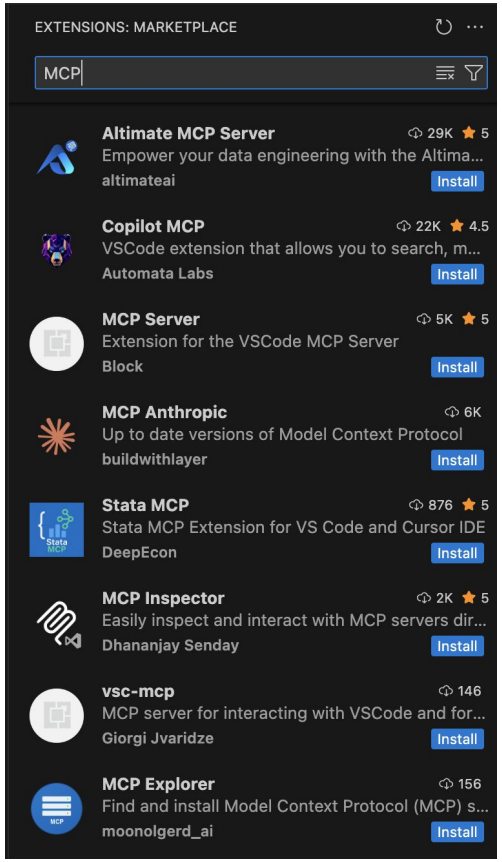
VScode

拡張機能から様々なMCPサーバーを追加することが可能です。
追加すると、#コマンドにMCPツールが表示されます。



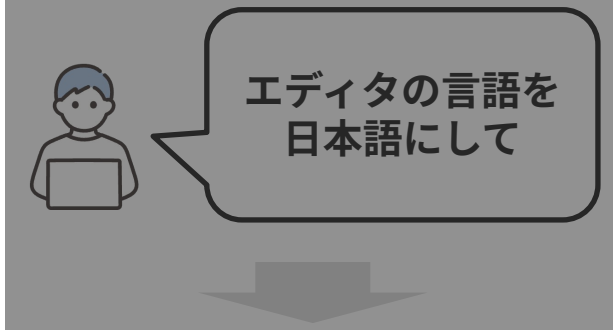
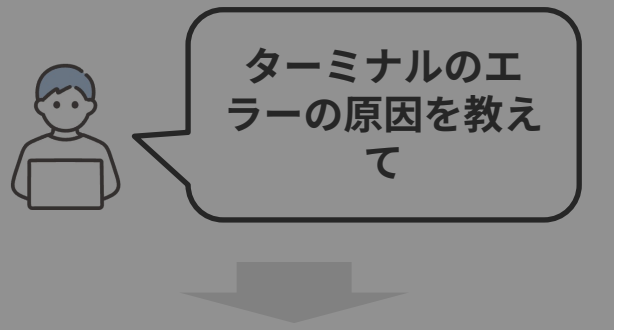
※１
MCP：Agentとツール間でコンテキストを標準化してやり取りする通信プロトコル

※２
拡張機能を追加する前に、必ず開発元を確認して、不明な拡張機能は追加しないことを推奨します



GitHub Copilot の使い方 - GitHub Copilot の3種類のコマンド

GitHub Copilotには3種類の基本コマンドがあり、それぞれ異なる用途で指示を最適化できます。

	@コマンド	#コマンド	/コマンド
概要	質問対象の スコープを指定する	Copilot に ツールの使用を指示する	特定の指示を ショートカットとして 利用する
使用例	 @vscode エディタの言語 を日本語にして	 #terminalLastCommand エラーの原因を教えて	 @workspace /explain

参考：GitHub「GitHub Docs」

GitHub Copilot の使い方 - Copilot Chat での / コマンド

VScode

/コマンドを使用することでCopilotがタスクを正確に理解し、適切な回答を提供します。

Copilot Chat での『/コマンド』一覧

#	コマンド	機能	対象
1	`/explain`	コードやターミナルの内容を説明	`@workspace, @terminal`
2	`/fix`	コードの修正案を提案	`@workspace`
3	`/doc`	コメントを付与	インラインチャットのみ
4	`/tests`	単体テストを生成	`@workspace`
5	`/search`	ワークスペース内検索クエリの生成	`@vscode`
6	`/startDebugging`	デバッグ設定を作成して開始（実験的）	`@vscode`
7	`/new`	ファイルツリー構造の提案	`@workspace`
8	`/newNotebook`	Jupyter Notebook を作成	`@workspace`
9	`/setupTests`	プロジェクトのテストを設定する	`@workspace`
10	`/help`, `/clear`, `/save`	ユーティリティ系操作	–

参考：GitHub「GitHub Docs」

GitHub Copilot の使い方 - Copilot Chat での / コマンド

VScode

/explain と /tests コマンドの解説と演習をさせていただきます。

Copilot Chat での『/コマンド』一覧

#	コマンド	機能	対象
1	/explain	コードやターミナルの内容を説明	@workspace, @terminal
2	/fix	コードの修正案を提案	@workspace
3	/doc	コメントを付与	インラインチャットのみ
4	/tests	単体テストを生成	@workspace
5	/search	ワークスペース内検索クエリの生成	@vscode
6	/startDebugging	デバッグ設定を作成して開始（実験的）	@vscode
7	/new	ファイルツリー構造の提案	@workspace
8	/newNotebook	Jupyter Notebook を作成	@workspace
9	/setupTests	プロジェクトのテストを設定する	@workspace
10	/help, /clear, /save	ユーティリティ系操作	–

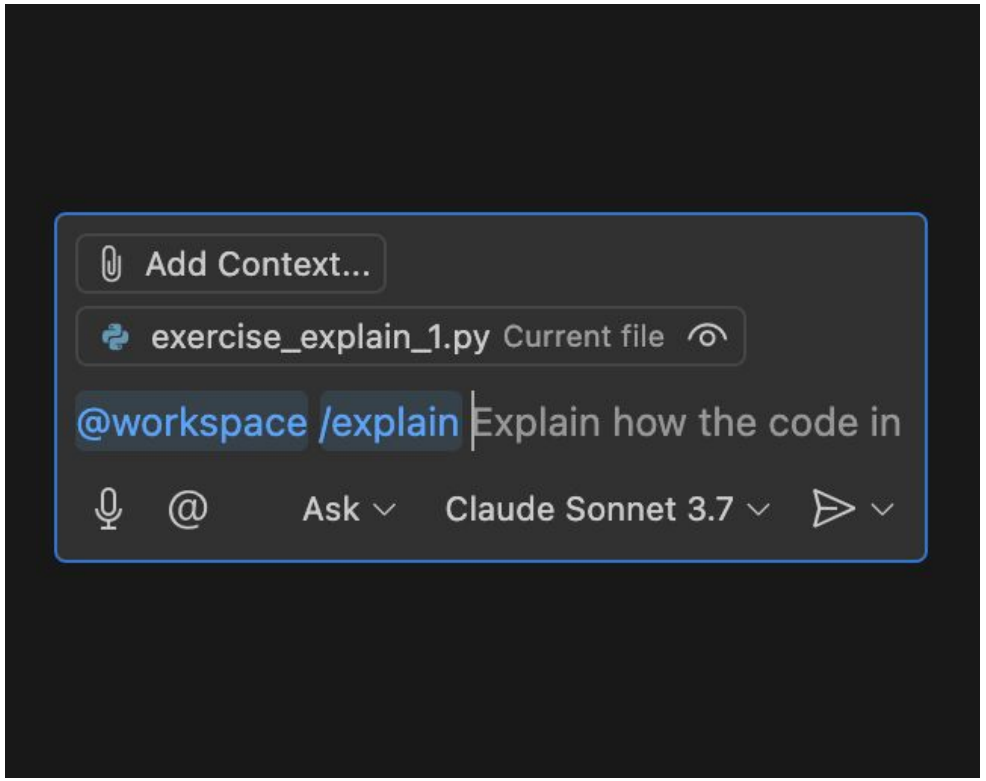
参考：GitHub「GitHub Docs」

GitHub Copilot の使い方 - /explain の使い方

VScode

『/explain』 コマンドでは、コードの内容を説明させることができます。

『/explain』コマンドの実行イメージ



実行

※Copilot チャットでは@workspace や @terminal と /explain はセットで使用します。

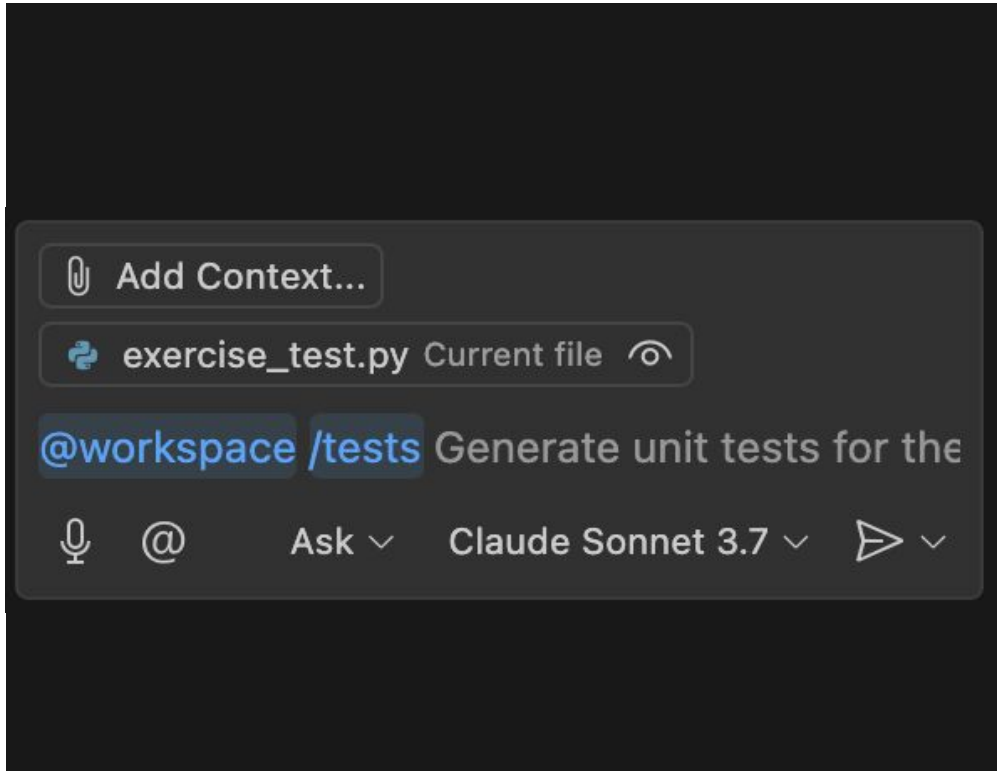
参考：GitHub「GitHub Docs」

GitHub Copilot の使い方 - /tests コマンド の使い方

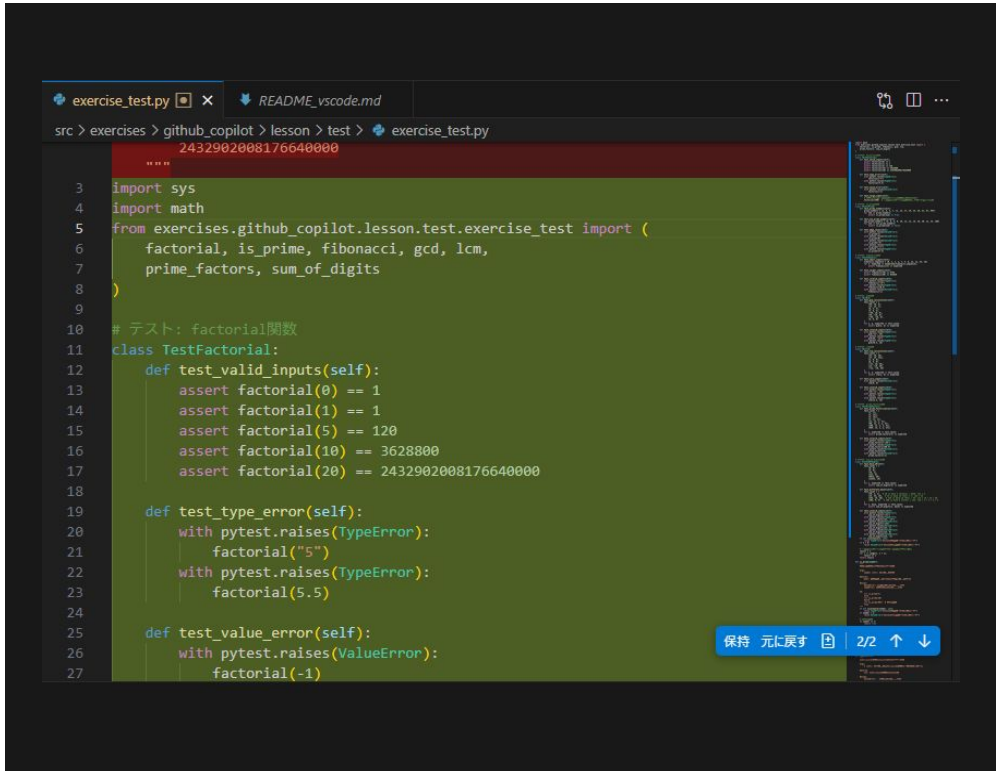
VScode

『/tests』コマンドでは、選択したコードやファイルのテストを実行することができます。

『/tests』コマンドの実行イメージ



実行



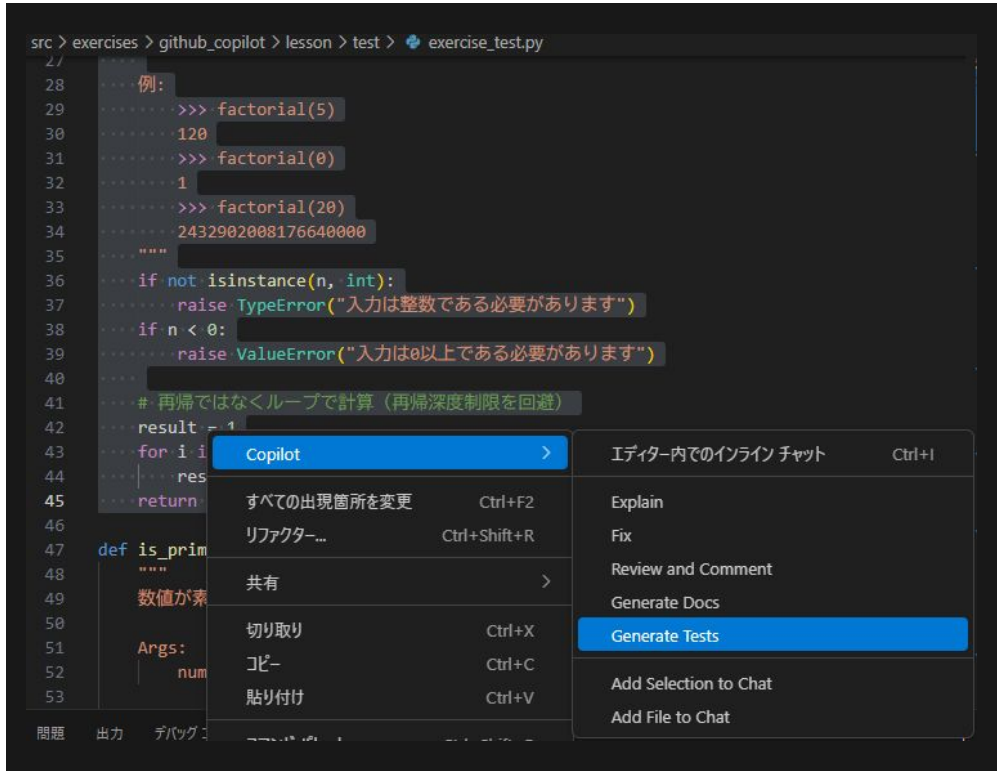
参考：GitHub「GitHub Docs」

GitHub Copilot の使い方 - /tests コマンド の使い方

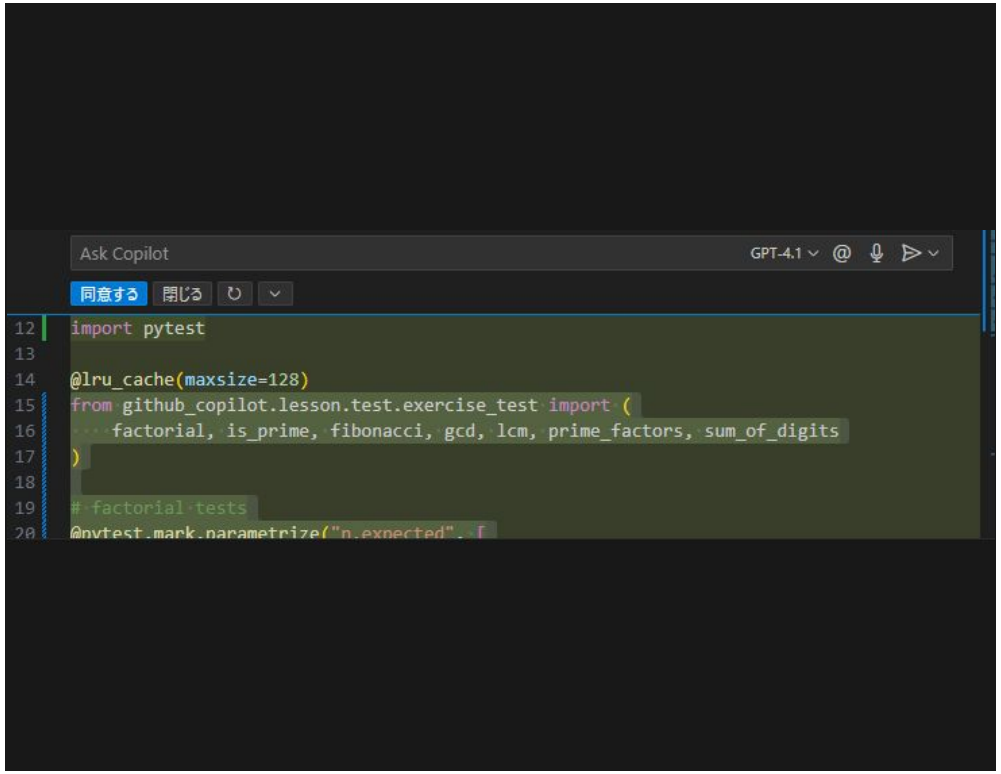
VScode

コードを範囲選択して、右クリック → Generate Tests を実行します。

『/tests』コマンドの実行イメージ



実行



例： インラインチャットでGenerate Test を実行

参考：GitHub「GitHub Docs」

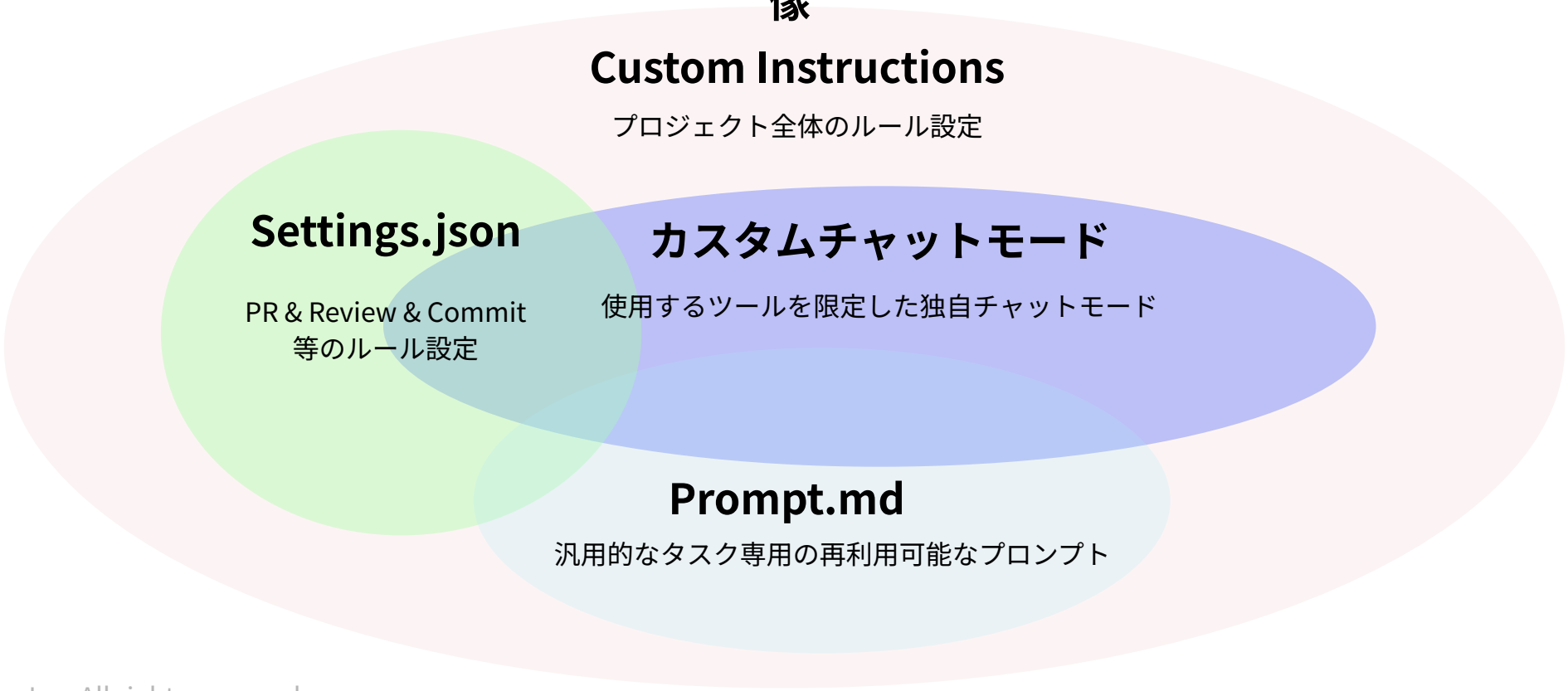
生成AI概論

- 各種コマンド（@コマンド、#コマンド、/コマンド）の全体像のご説明
  - 『@コマンド』の説明と演習
  - 『#コマンド』の説明と演習
  - 『/コマンド』の説明と演習
- **Custom Instructionsの説明と演習**

GitHub Copilot の使い方 - Copilot のカスタマイズ：全体のイメージ

4つのカスタマイズ機能により、プロジェクト全体のルールから個別タスクまで、チームの開発スタイルに合わせたAIコーディング環境を構築できます。

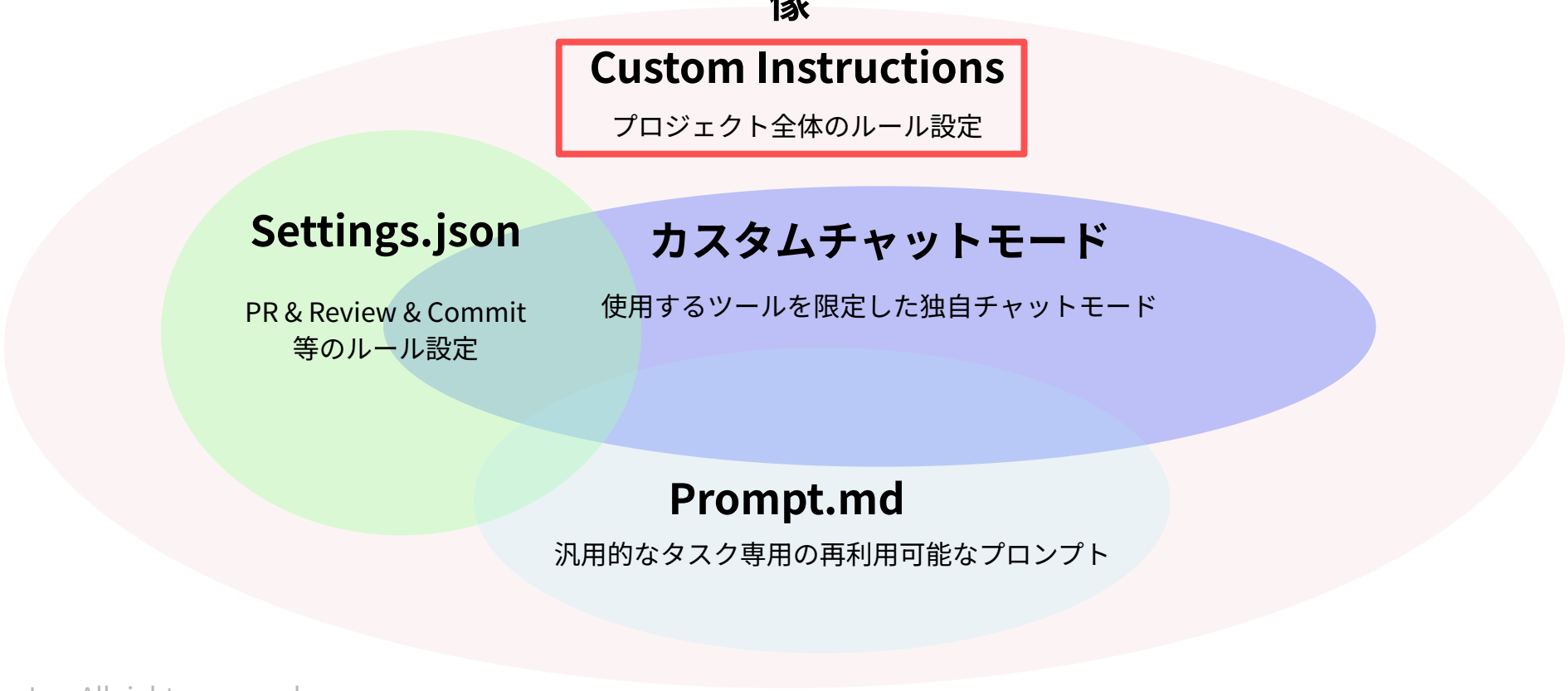
GitHub Copilot のカスタマイズ機能の全体像



GitHub Copilot の使い方 - Copilot のカスタマイズ：全体のイメージ

4つのカスタマイズ機能により、プロジェクト全体のルールから個別タスクまで、チームの開発スタイルに合わせたAIコーディング環境を構築できます。

GitHub Copilot のカスタマイズ機能の全体像



GitHub Copilot の使い方 - Custom Instructions

GitHub CopilotのCustom Instructionsとは
AIの提案をプロジェクトやチームのルールに合わせてカスタマイズできる機能です。

GitHub Copilot - Custom Instructions 公式ドキュメント説明

参考：[GitHub Docs「GitHub Copilot のリポジトリ カスタム命令を追加する」]

GitHub Copilot の使い方 - Custom Instructionsの活用シーン

**Custom Instructionsを活用することでチーム開発から個人開発まで、
様々なシーンで開発の生産性を向上させることができます。**

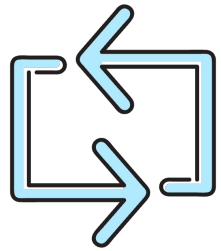
チーム内での設定

**チームのコーディング規約や
使用技術、プロジェクト固有の
情報を理解させる**

個人最適化

**個人の好み（応答言語、スタイル
など）を反映させる**

コンテキスト繰り返し利用



**繰り返し同じコンテキストを
チャットで説明する手間を省く**

GitHub Copilot の使い方 - Custom Instructionsの設定方法

リポジトリカスタム指示：.github/copilot-instructions.md を設定する

	説明
ファイル形式	• ルートに .github/copilot-instructions.md を作成（.github フォルダが無ければ新規作成）。
記述形式	• Markdown形式で、自然言語で指示を記述。 • 指示は短く、自己完結型の文章が望ましい。 • 読みやすさのために空白行で区切っても、段落としてまとめてもOK。
前提条件	• ファイルが存在する状態で、VS Code／Visual Studio などの 「カスタム指示を有効にする」 オプションを ON（既定は OFF）。 • ※設定画面に該当チェックボックスあり。
利用確認方法	• Copilot Chat で質問 → 応答下部 References に copilot-instructions.md が表示されていれば適用完了。

参考：GitHub「GitHub Docs」

演習：GitHub Copilot の使い方 - .github/copilot-instructions.md

次のシナリオを読み、`.github/copilot-instructions.md` に、どのような情報を書けば Copilot の提案精度を高められるかを実践してみましょう。

プロンプト例

`.github/copilot-instructions.md` を以下の参考資料を元にして作成してください。

参考資料：

- Zoom のチャットにテキストを貼り付けます。

GitHub Copilot の使い方 - 複数のCustom Instructions設定

用途別にCustom Instructionsファイルを作成して、Agentに参照させることができます

Custom Instructions の例

```
.github/
    instructions/
        general.instructions.md
        nextjs15app.instructions.md
        api_document.instructions.md
        progress.md
```

解説

- 複数 instructions.md を活用
  - フレームワーク別・言語別・API仕様別に用意
  - Agentが適切な情報を自動で読み込み、毎回の指示入力が不要
- progress.md で進捗を可視化
  - 完了タスク／残タスクを簡潔に更新
  - チャットウィンドウを替えても Agent が最新状況を継承
- 得られるメリット
  - チーム内外で同じドキュメントを共有でき、認識ズレを防止
  - 新メンバーもファイルを見るだけで背景を把握
  - 会話のたびに情報を引き継ぐため、開発効率と回答精度が向上

参考：Visual Stuido Code「April 2025 (version 1.100)」

研修のスケジュール　1日目：GitHub Copilot基礎と実装体験

TIME：7h

● 座学主体　● ハンズオン主体

■Session01：オリエンテーション［30min］ 研修ゴール共有｜自己紹介・2日間の概要説明｜ツール紹介（VSCode・GitHub Copilot）｜学習スタンスの共有

■Session02：生成AIリテラシーとガバナンス［100min］

- ● **ガバナンス基礎**　生成AIの仕組み｜4月研修の復習・接続｜機密情報・著作権・ハルシネーションの実例｜社内ガイドラインの要点確認｜やってよいこと・いけないことの判断基準
- ● **理解度確認テスト**　理解度確認テスト｜生成AI利用時のセキュリティリスク｜ガイドラインに基づく利用可否判断｜AI生成コードのレビュー義務確認｜テスト後に正解と解説をその場で共有
- ● **環境セットアップ**　VSCode起動・Copilot拡張の認証確認｜インラインチャット｜Copilot Chatの基本操作｜スラッシュコマンド｜FizzBuzzで最初の生成体験

■Session03：Javaコード生成ハンズオン［140min］

- ● **メソッド・DTOの生成**　メソッド名とJavadocからの実装自動生成｜getter/setter/toString/equals自動生成｜RecordクラスやLombokアノテーションの提案確認｜カスタム例外クラス
- ● **CRUD処理の生成**　DAOクラスのCRUDメソッド自動生成｜SQLクエリ生成支援｜Spring Data JPAリポジトリ生成｜try-catchブロックの自動追加｜例外処理パターンの提案比較
- ● **レビュー実践**　問題点発見｜セキュリティ観点（SQLインジェクション・ハードコード値）｜ロジック観点（NullPointerException・境界値）｜命名規約・コード規約との整合性確認

■Session04：ペアワーク・コードレビュー演習［120min］

- ● **ペアレビュー**　ペアレビュー｜セキュリティ・ロジック・命名の3観点｜Copilot Chatで修正｜インラインチャットで部分修正を体験｜複数パターン比較・採用判断基準の整理
- ● **Day 1振り返り**　本日のキーメッセージ3点を確認｜AIコードの最終責任は人間にあることを再確認｜プロンプトの書き方と品質の関係｜Day 2の予告（テスト生成・デバッグ・総合演習）

■Session05：Day1クロージング・明日の予告［30min］

＊上記スケジュールは変更となる場合がございます。

事前セットアップ：事前セットアップ：VS Code最新版 ｜ CopilotChat拡張機能 ｜ 各自の開発言語実行環境

Github Copilot の活用Tips

Github Copilot の活用Tips - ゴール

本パートのゴール	GitHub Copilotの様々な機能を把握し、 自らの継続学習をスタートするきっかけとする

Github Copilot の活用Tips

- 便利な拡張機能のご紹介
- MCPの概要とMCPサーバーのご紹介

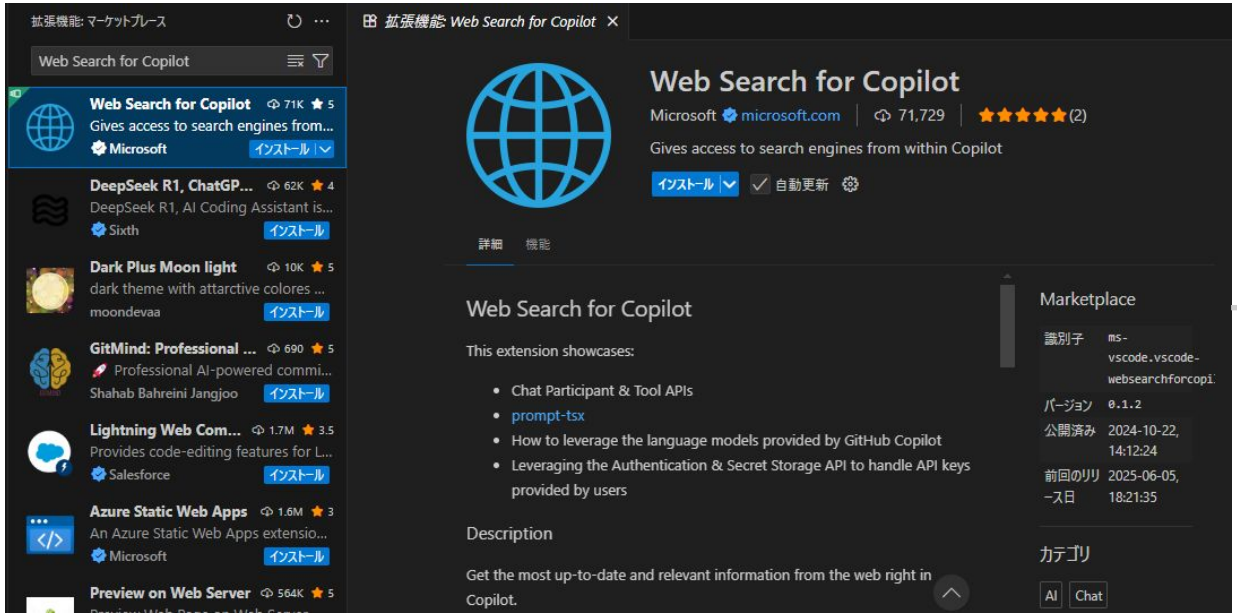
Github Copilot の活用Tips

- 便利な拡張機能のご紹介
- MCPの概要とMCPサーバーのご紹介

Github Copilot の活用Tips - 拡張機能：Web検索

VScode

Web検索によって、適切なソースから回答を得ることが可能



① エディタ左側の拡張機能検索から、Web Search for Copilot を検索してインストール

※モデルがClaude だと使用することができません

② @websearch ～を実行する

③ ポップアップが出たら許可

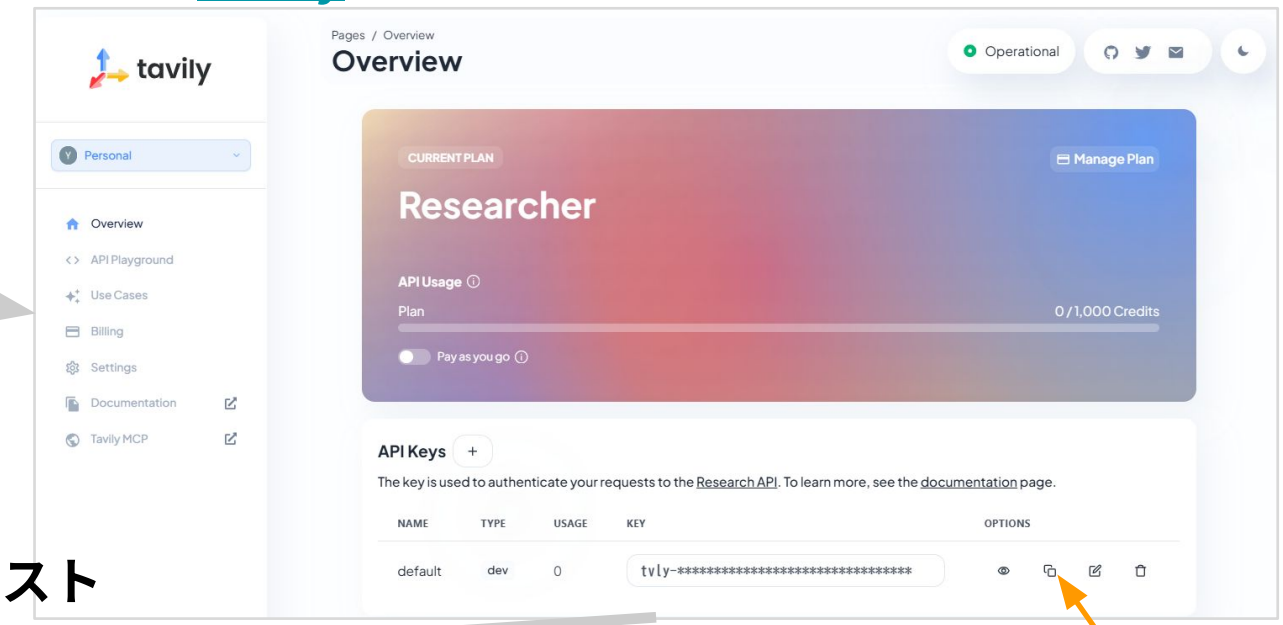
Github Copilot の活用Tips - 拡張機能：Web検索

VScode

TavilyのAPIキーを登録：初めて@websearchを使用する際にAPIキーの入力が求められるので、TavilyでAPIキーを作成しましょう。

④ 再びポップアップが出たら許可

⑤ Tavily でアカウント作成・APIを取得



⑥ APIキーをペースト

APIキーをコピー

Github Copilot の活用Tips - 拡張機能：Web検索

VScode

Websearch：Web検索して公式ドキュメントから回答させることも可能です。

Websearch

Github Copilot の活用Tips - 拡張機能：vscode-mermAId について

VS Code Mermaidは、コードやプロジェクト構造から図表を自動生成し、自然言語での編集や共有が可能なVS Code 拡張機能です。

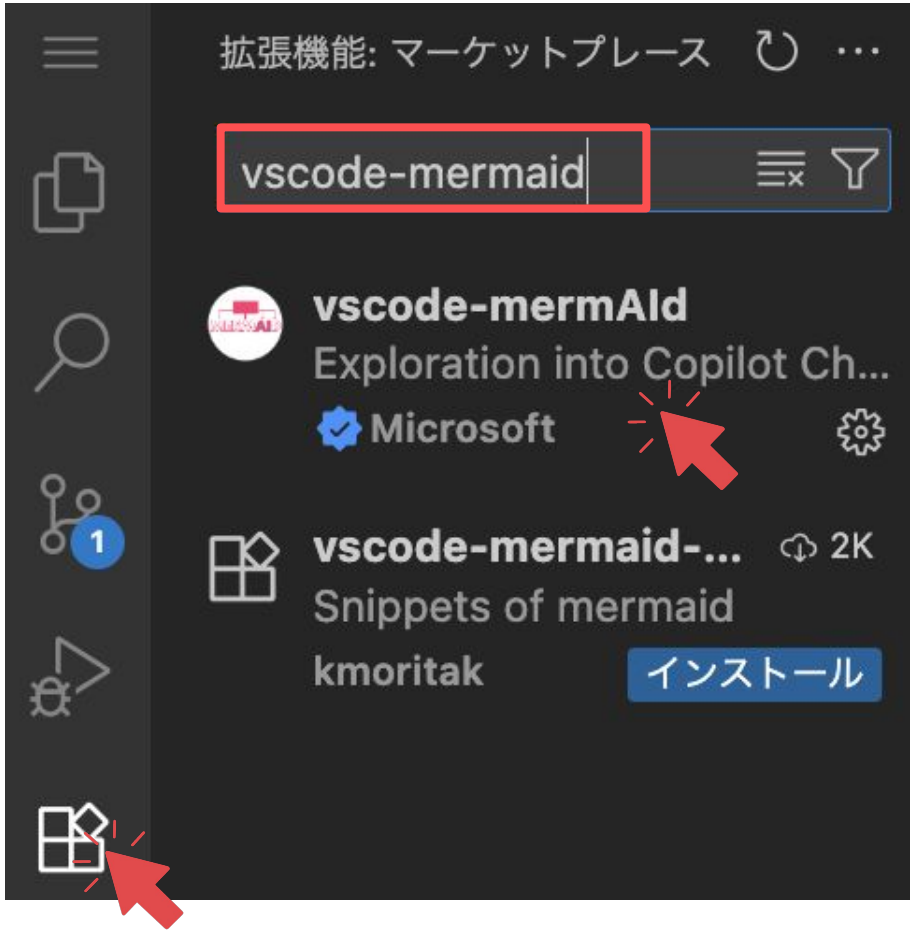
vscode-mermAId 使用イメージ

参考：[GitHub - Microsoft「vscode-mermAId」]

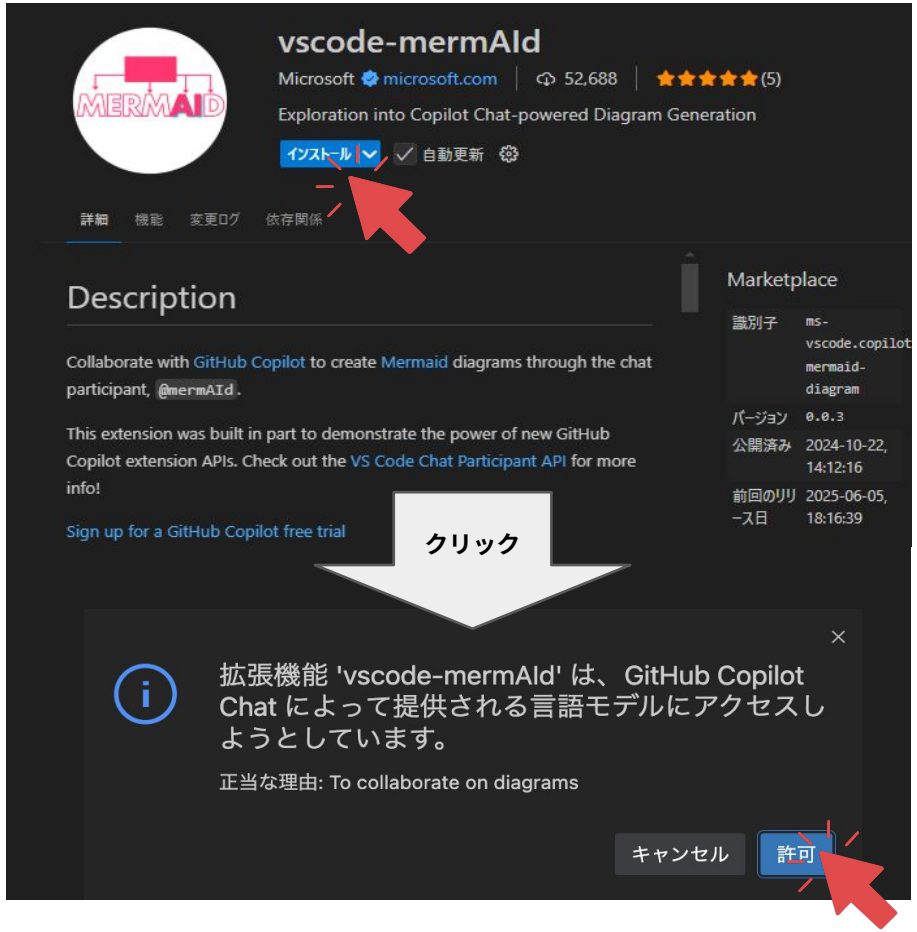
Github Copilot の活用Tips - 拡張機能：vscode-mermAId のインストール

VS Codeの拡張機能から、vscode-maermAId をインストールして使用することができます。

vscode-maermAIdのインストール手順



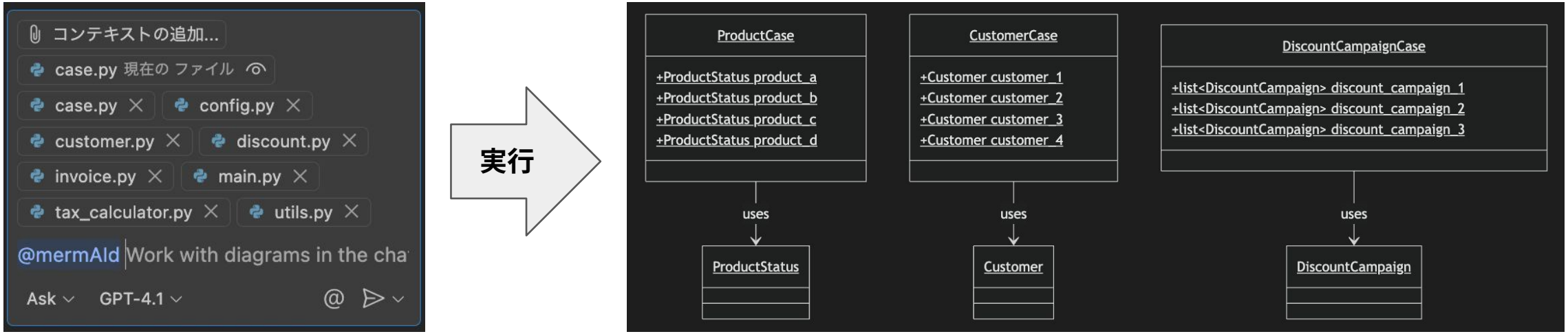
クリック



Github Copilot の活用Tips - 拡張機能： vscode-mermAId について

コードやプロジェクトの理解時間を短縮し、手動でのフローチャート作成やドキュメント更新作業を自動化して開発効率の向上が期待できます。

vscode-mermAId 使用イメージ



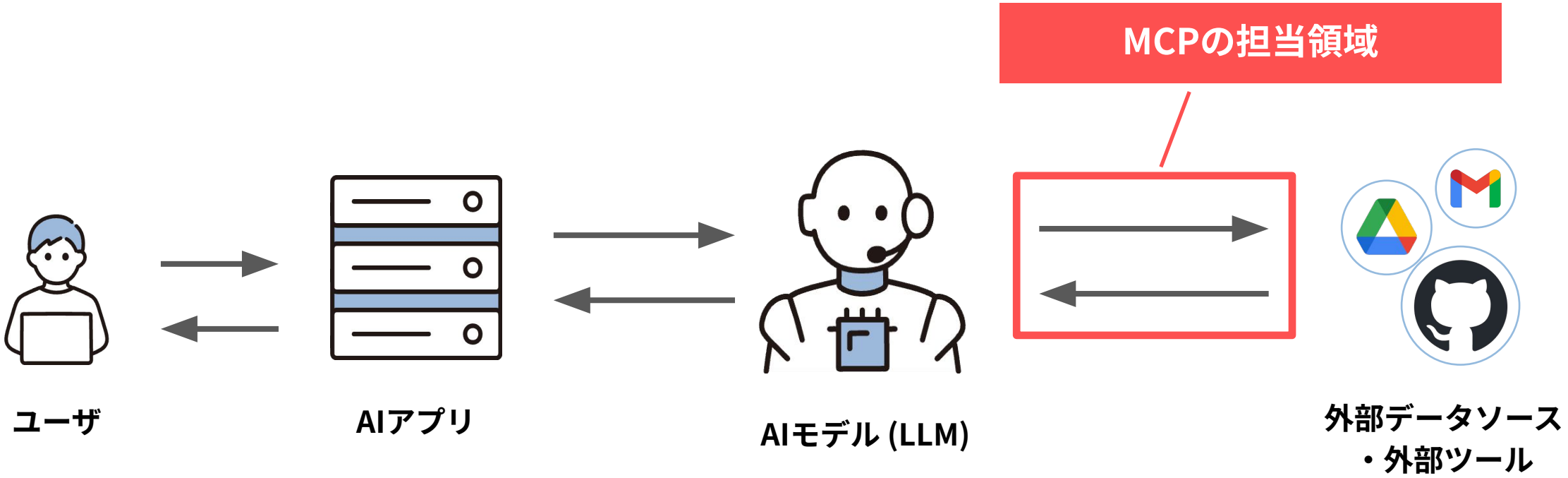
参考：[GitHub - Microsoft「vscode-mermAId」]

Github Copilot の活用Tips

- 便利な拡張機能のご紹介
- **MCPの概要とMCPサーバーのご紹介**

Github Copilot の活用Tips - MCPの概要とイメージ

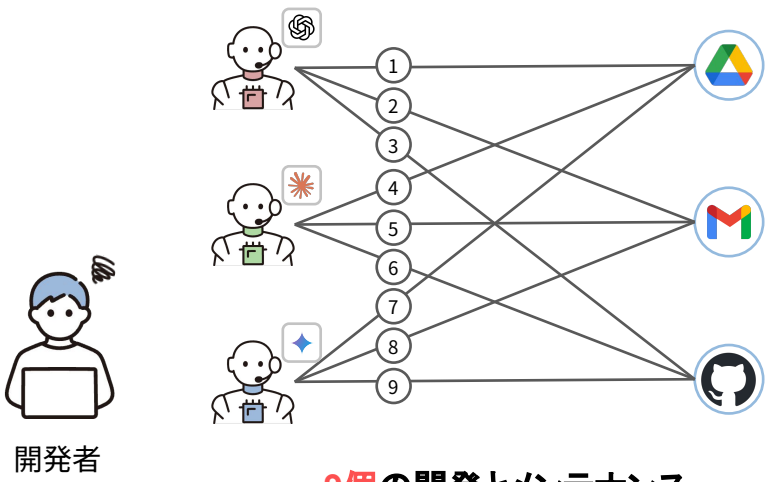
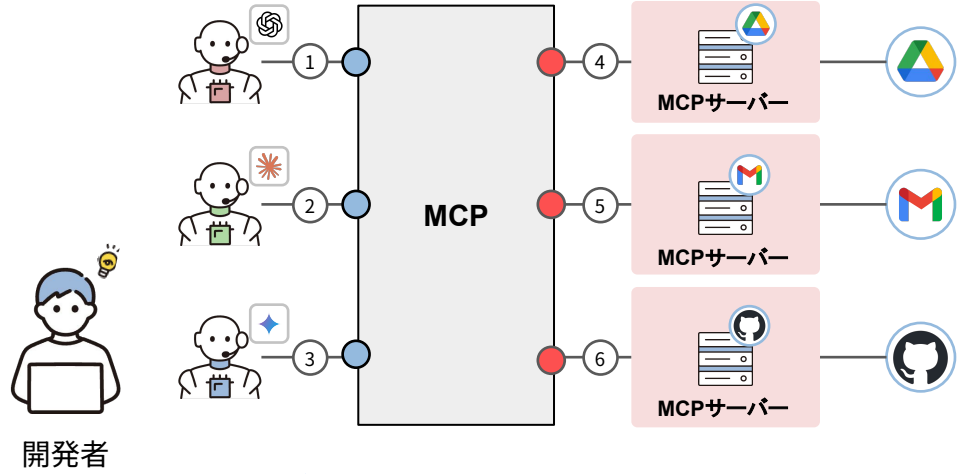
Model Context Protocol（MCP）は、2024年11月にAnthropic社が生み出した
AIモデルと外部のデータやツールとスムーズに連携するプロトコルのことです。



 参考： Anthropic「Introducing the Model Context Protocol」

Github Copilot の活用Tips - 従来型AI連携の課題とMCP導入による効果

MCPを活用することで開発及びメンテナンス対象のシステム数を大幅に削減することができます。

	Before	After
特徴	• ツールが増えるごとに**複雑かつ手間がかかる** • **メンテナンスが大変で効率が悪い**	• 規格に従えば簡単に手間なくAI連携が可能 • **開発コストを大幅に削減**できる
イメージ	 **9個の開発とメンテナンス**	 **6個の開発とメンテナンス**

Github Copilot の活用Tips - MCPオフィシャルサーバー一覧 - 機能と用途

各言語の公式 MCP-SDK が幅広いシナリオに対応しています。

言語別の主な公式MCP の機能と用途 一覧

SDK／サーバー名	主な機能	主な用途
Python SDK	Resources／Tools／Prompts登録、 CLI・SSE・Streamable HTTP対応	プロトタイピング、CLIツール、Webアプリ
TypeScript SDK	同上 ＋ 低レベルServer、stdio	Node.jsアプリ、VS Code拡張、サーバーレス開発
Java SDK	同上 ＋ Spring Bootスターター連携	エンタープライズサービス、マイクロサービス
Kotlin SDK	同上 ＋ iOS／Wasm対応	モバイル・ブラウザ・マルチプラットフォーム
C# SDK	同上 ＋ ASP.NET Core統合、DI・属性ベース定義	Windows／Azure Functions、Web API

参考：GitHub「Model Context Protocol」

まとめ

まとめ

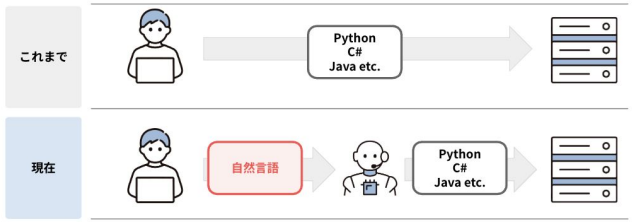
プログラミングの歴史的な変遷とAIコーディングに必要なスキルを理解し、適切なリスク対策を講じながら、生成AIコーディングによって開発効率を向上させていくことが重要です。

まとめ

生成AIネイティブ開発の概論 - プログラミング言語の変遷

生成AIの登場により、『人工言語から自然言語によるプログラミング』へと進化しつつあります。

プログラミングの進化の概念図（人工言語→自然言語）



GitHub Copilot 概論 - GitHub Copilot チャットの３つのモード

VScode

GitHub Copilotチャットでは、3つのモードを活用することで開発効率を最大化することができます。

	Askモード	Editモード	Agentモード
概要	対話型チャットで コード相談・質問解決	指示に基づいて 複数ファイルを一括編集	自律的にプロジェクト 全体のタスクを実行
@/ コマンド	○	×	×
使用 シーン	例:「Pythonについて教えてください」	例:「(複数ファイル選択) Google 形式の docstring を付けて」	例:「Flask で在庫管理のWebアプリケーションを作って」

 参考：GitHub Inc.「Copilot ask, edit, and agent modes: What they do and when to use them」

GitHub Copilot の使い方 - GitHub Copilot の3種類のコマンド

GitHub Copilotには3種類の基本コマンドがあり、それぞれ異なる用途で指示を最適化できます。

	@コマンド	#コマンド	/コマンド
概要	質問対象の スコープを指定する	Copilot に ツールの使用を指示する	特定の指示を ショートカットとして 利用する
使用例	エディタの言語を日本語にして @vscode エディタの言語を日本語にして	ターミナルのエラーの原因を教えて #terminalLastCommand エラーの原因を教えて	このコードを解説して @workspace /explain

 参考：GitHub「GitHub Docs」

GitHub Copilot の使い方 - Custom Instructionsの活用シーン

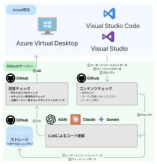
Custom Instructionsを活用することでチーム開発から個人開発まで、様々なシーンで開発の生産性を向上させることができます。

チーム内での設定	個人最適化	コンテキスト繰り返し利用
チームのコーディング規約や使用技術、プロジェクト固有の情報を理解させる	個人の好み（応答言語、スタイルなど）を反映させる	繰り返し同じコンテキストをチャットで説明する手間を省く

生成AIネイティブ開発におけるリスク対策 - GitHub Copilot のセキュリティ

暗号化 × 最小保持 × 厳格権限制御によって、Copilotのコード流出リスクを低減します。

データのフローおよびデータの概要、各保存期間、暗号化については下記の通りです。



送信されるデータの種類と保存期間

データの種類	データの概要	データの保存期間
①ユーザーエンゲージメントデータ	・仮名化されたID ・承認/却下された補完 ・エラーメッセージ ・システムログ ・製品使用状況メトリクス	2年間保存される
②フィードバックデータ	・AIの回答のユーザー評価 ・ユーザーのコメント	本来の目的に必要な期間保存
③プロンプト	・コードの入力内容 ・チャットの入力内容 ・関連するコンテキスト	IDE 経由：保存されない IDE以外：28 日間保持
④回答	・AI生成コード ・AIのチャット応答	IDE 経由：保存されない IDE以外：28 日間保持

出典：GitHub Trust Centerなどのドキュメントを元に作成

データフローの要約

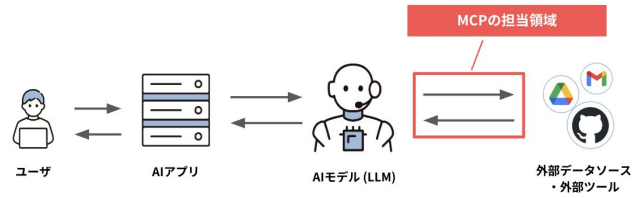
- すべての経路で TLS、保存時は FIPS 140-2 準拠で暗号化。
- ①②は品質向上のためのみ長期保管、③④はリアルタイム処理後に速やかに消去。
- アクセス権はロールベース＋多要素認証で厳格に制御。

※仮名化ID：ユーザーを特定できないようハッシュ化した内部識別子。利用傾向分析用で、個人情報は含まれない。
※承認／却下補完：Copilot 提案をそのまま挿入すれば「承認」、無視・削除・編集すれば「却下」と記録。モデル改善の指標になる操作ログ。

 参考：GitHub「GitHub Copilot Trust Center」

MCP概論 - MCPのイメージ

Model Context Protocol（MCP）は、2024年11月にAnthropic社が生み出したAIモデルと外部のデータやツールとスムーズに連携するプロトコルのことです。



 参考： Anthropic「Introducing the Model Context Protocol」

午後の演習の全体像

入門トラックで操作に慣れ、演習1から3で座学を手を動かして確かめます。

入門トラック

java-todo / タスク管理TODO

A-1 補完 [10min]

A-2 プロンプト [15min]

A-3 チャット [15min]

Copilotの入口3つを切り分けて体験

演習1 [55min]

プロンプト エンジニアリング

java-inventory / 機器点検

雑な依頼と整った依頼で
返答を比べる

演習2 [40min]

AIに問いを 立てる力

java-inventory / 機器点検

症状から仮説を立て、
検証できる問いにする

演習3 [95min]

エージェントモード 不具合調査

java-inventory / 機器点検

前半 [50min] ＋ 休憩 [10min]
＋ 後半 [45min]

座学とのつながり　演習1 → 3・4章　演習2 → 1・2章　演習3 → 5・6章

早く終わった方向けの任意メニュー

スキル演習1〜3（テスト生成・リファクタ・仕様から新機能実装）
補助演習1〜2（Python：ログ整形・請求データ検算）　チャレンジ1〜4

配布フォルダの歩き方

3つのフォルダを、開く順番で使い分けます。

exercise/

全13演習の完全手順書
まず開くフォルダ

→

入口は 00_進め方.md

何をするか・どこを触るか・どうなれば正解かが手順書だけで分かる
スライドが手元になくても、これだけで一人で進められます

hints/

参考プロンプトと解説
詰まったら開くフォルダ

→

5〜10分行き詰まってから開く

まず自分で仮説を立ててから見る
各手順書の末尾に、対応するhintsのファイル名を書いています

_reference_完成形/

答えそのもの
研修中は開かないフォルダ

→

確認は演習が終わってから

同名フォルダに完成コードが入っています
先に見ると、自分で問いを立てる練習にならないため復習用に使います

株式会社ギブリー (Givery, Inc.)
〒150-0036 東京都渋谷区南平台町15-13 帝都渋谷ビル8F

制作 : Givery 株式会社／株式会社エヌアイデイ

GitHub Copilot研修（AI駆動開発力強化）