生成AI活用実践研修

2日目

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T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、2日間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

サブ講師

濵田 龍之介

Ryunosuke Hamada

＜プロフィール＞
大学卒業後、アパレル企業にて店舗運営に従事。販売戦略から売上管理、スタッフの育成・マネジメントまでを担い、顧客起点でのビジネスとチームビルディングの基礎を築く。
その後IT業界へ転身し、国内SIerにて金融システムのサポート業務、総合商社にて社内ITヘルプデスクの立ち上げとメンバー育成を主導。
大手外資クラウドベンダーにて、カスタマーサービスとして顧客の課題解決に貢献。同チームのマネージャーを経て、サポート部門全体の人材開発を担うトレーニングチームへ異動。各種育成プログラムのプロジェクトマネージャーとして、組織的なスキルアップを推進。
2025年12月よりギブリーに参画。

得意言語／多言語経験

【領域】講師育成、研修カリキュラム開発、研修運営、業務改善

【言語】JavaScript

経験／実績など

【研修実績】
・大手外資クラウドベンダー(内製)
新卒研修運営、中途社員研修企画運営、リスキリング研修企画運営、Train The Trainer 研修講師、英語学習プログラム企画運営、マネージャー向け研修企画運営、社内認定資格運営

・2026年度 大手通信系SIer サブ講師

講師紹介

サブ講師

河合 謙一郎
Kenichiro Kawai

＜プロフィール＞
病院勤務（診療報酬請求・診療情報管理）を経てITエンジニアへ転身。
Java/Spring Bootを中心に要件定義から設計・実装・運用まで一貫して基幹システム開発に従事し、実務エンジニアとしての経験を積む。
その後研修インストラクター兼コンテンツクリエイターへ転向。自らの開発経験を軸にゼロからカリキュラム設計・教材開発を一手に担い、3年で250名超のエンジニア育成を実現。
並行して業務改善スペシャリストとしてGemini API×n8nを活用したQ&A自動化など、AI駆動の業務改善プロジェクトを主導し、定量的な成果を複数達成。

得意言語／多言語経験

【言語】Java, Python, JavaScript

【FW/ツール】Spring, React, n8n, Claude Code

【領域】講師・研修設計・カリキュラム開発, AI業務自動化

経験／実績など

【研修実績】
・研修インストラクター3年
3年で250名超のエンジニア育成、AI/Java/n8n eラーニングコース複数開発
・2026年度 大手メーカー系SIer メイン講師

【資格取得支援経験】
ITパスポート・基本情報技術者・AWS Certified Cloud Practitioner・AWS Certified Solutions Architect – Associate

講師紹介

サブ講師

河野 智

Satoru Kono

＜プロフィール＞
システム開発業務に約20年従事し、現在も現役エンジニアとして開発業務に携わる一方、IT分野の講師として主に新入社員研修を担当しています。
開発現場と教育現場の双方を経験していることを強みとし、実務に直結する知識や現場視点の実践的なアドバイスを講義に取り入れています。
【主な開発業務】
RPAパッケージ開発
需要予測システム
イーラーニングシステム
在庫管理システム
勤怠管理システム、シフト自動生成システム
医学論文検索システム
IoTデバイス開発(見守りシステム)

得意言語／多言語経験

【言語】C#、Java, Python,

【FW/ツール】.NET MVC, Spring, Codex, Claude Code

【領域】講師・研修設計・カリキュラム開発, AI業務自動化

経験／実績など

【研修実績】
・研修インストラクター15年
・大手メーカー系SIer/教育ベンダーオープンコース

【資格取得支援経験】
・Project Management Professional (PMP)
・SAP Certified Application Associate – Sales 2020
・ドットコムマスター　トリプルスター(NTT Communications)

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

研修のスケジュール　2日目：テスト・デバッグ・実践演習

TIME：7h

● 座学主体　● ハンズオン主体

■Session01：前回振り返りとDay 2概要［30min］ 前日の振り返りクイズ（5問）｜Copilot基本操作の確認｜Day 2ゴール共有｜テスト・デバッグ・総合演習の概要

■Session02：テストコード生成ハンズオン［130min］

- **JUnitテスト自動生成**　/testsコマンドでテストコード自動生成｜テストメソッドの自動命名確認｜正常系・異常系・境界値の網羅｜null入力・空文字・最大値最小値テスト
- **テスト実行と修正**　生成テストを実行して失敗ケースを特定｜テストが通るよう実装側を修正体験｜例外が正しくスローされるかのテスト実践｜テスト品質の確認と改善サイクル
- **デバッグ・リファクタリング実践**　/fixコマンドでバグ修正｜エラーメッセージから原因特定｜長いメソッド・マジックナンバー｜重複コードの共通化体験｜/explainでJavadoc自動生成

■Session03：総合演習 AI活用ミニ開発［150min］

- **演習課題**　簡易設計書（画面仕様書 or API仕様書）配布｜設計書→実装→テスト→レビューの一連の流れ｜Copilot Chat実装｜手作業との所要時間を計測して比較
- **成果共有・振り返り**　数名が画面共有で成果を発表｜「Copilotが役立った場面・邪魔だった場面」を共有｜2日間の学びを全体で振り返り｜配属後の活用イメージを言語化
- **配属後の活用ガイド**　新プロジェクト参画時のコード理解｜定型コード高速生成｜お客様環境での利用可否判断基準｜AIに任せる/自分で書く場面の整理

■Session04：学習リソース紹介・グループディスカッション［80min］

- **学習リソース**　GitHub Copilot公式ドキュメント｜VSCode公式ドキュメント｜JUnit 5ユーザーガイド｜社内の生成AI活用事例・ナレッジ｜今後の研修ロードマップ
- **グループディスカッション**　3～4名で「配属後に使いたい場面」を話し合う｜使えない現場での代替手段を考える｜セキュリティ上の判断フローを整理｜グループ代表者が発表

■Session05： 質疑応答・アンケート・閉会［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

詳細なゴールとゴールでないこと

最低限クリアすること

1. 生成AIの基本的な仕組み、危険性を理解する
2. VSCodeの使い方を理解する
3. GithubCopilotの使い方を理解する

できればクリアすること

1. ここまでの実装体験をAIに適用できる
2. AIの得意と不得意を理解し、使い分けられる
3. 基本的な機能をAI駆動で実装できる

目指していないこと

- 実務で使えるアプリを開発できる
- AI駆動開発について完全に理解する

おことわり

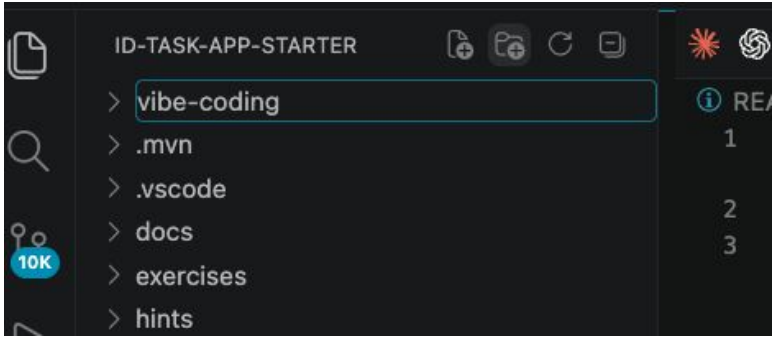
本資料には、

持ち帰って実施いただくことを想定した資料・ハンズオンが含まれています。

後日ご自身で実施・アップデートしていきましょう。

ディスカッション③

バイブコーディングの準備



・フォルダ直下に
　『vibe-coding』というフォルダを作成

・フォルダの中に『実装メモ.md』というマークダウンファイルを作成

・GithubCopilotに
　『実装メモ.mdに書かれているアイデアを元に、HTML単品ファイル形式でWebアプリを開発してください』と指示

・ファイルが生成されたら、HTMLファイルをダブルクリックして
　ブラウザで確認、修正してみましょう

Copyright © Givery, Inc. All rights reserved.

バイブコーディング×アイデア　の価値

2026.07.13 Sam altman（ChatGPTの生みの親）がXにて

i'd love to see interesting things people have built with 5.6 sol.
i will send the person who made the coolest thing a special gift from the openai archives.

5.6 SOLで人々が作った面白いものをぜひ見てみたいです。

一番クールなものを作った人に、OpenAIのアーカイブから特別なギフトを送ります。

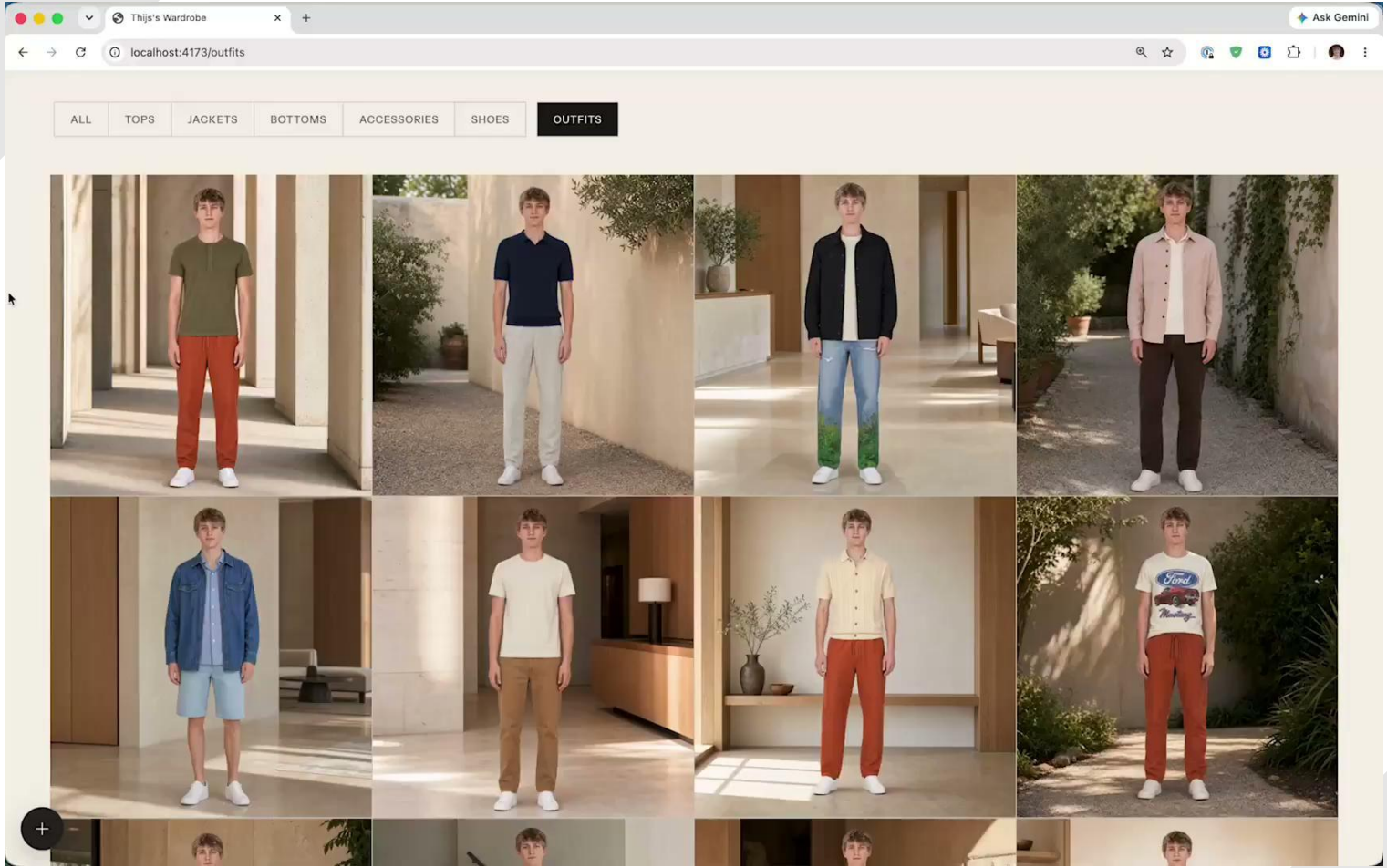
この翻訳を評価:

午前5:08 · 2026年7月13日 · **558.8万** 件の表示

テーブルでディカッション(30分)

ぼくがかんがえた
さいきょうのアイデアを
ことばにしてみましょう

ライバルたちの投稿
https://x.com/sama/status/2076398253332140410?s=20



https://x.com/cdngdev/status/2076812846793650485?s=20

本日の環境

はじめに

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

研修のスケジュール　2日目：テスト・デバッグ・実践演習

TIME：7h

● 座学主体　● ハンズオン主体

■Session01：前回振り返りとDay 2概要［30min］ 前日の振り返りクイズ（5問）｜Copilot基本操作の確認｜Day 2ゴール共有｜テスト・デバッグ・総合演習の概要

■Session02：テストコード生成ハンズオン［130min］

- **JUnitテスト自動生成**　/testsコマンドでテストコード自動生成｜テストメソッドの自動命名確認｜正常系・異常系・境界値の網羅｜null入力・空文字・最大値最小値テスト
- **テスト実行と修正**　生成テストを実行して失敗ケースを特定｜テストが通るよう実装側を修正体験｜例外が正しくスローされるかのテスト実践｜テスト品質の確認と改善サイクル
- **デバッグ・リファクタリング実践**　/fixコマンドでバグ修正｜エラーメッセージから原因特定｜長いメソッド・マジックナンバー｜重複コードの共通化体験｜/explainでJavadoc自動生成

■Session03：総合演習 AI活用ミニ開発［150min］

- **演習課題**　簡易設計書（画面仕様書 or API仕様書）配布｜設計書→実装→テスト→レビューの一連の流れ｜Copilot Chat実装｜手作業との所要時間を計測して比較
- **成果共有・振り返り**　数名が画面共有で成果を発表｜「Copilotが役立った場面・邪魔だった場面」を共有｜2日間の学びを全体で振り返り｜配属後の活用イメージを言語化
- **配属後の活用ガイド**　新プロジェクト参画時のコード理解｜定型コード高速生成｜お客様環境での利用可否判断基準｜AIに任せる/自分で書く場面の整理

■Session04：学習リソース紹介・グループディスカッション［80min］

- **学習リソース**　GitHub Copilot公式ドキュメント｜VSCode公式ドキュメント｜JUnit 5ユーザーガイド｜社内の生成AI活用事例・ナレッジ｜今後の研修ロードマップ
- **グループディスカッション**　3～4名で「配属後に使いたい場面」を話し合う｜使えない現場での代替手段を考える｜セキュリティ上の判断フローを整理｜グループ代表者が発表

■Session05： 質疑応答・アンケート・閉会［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

バイブコーディング体験

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）

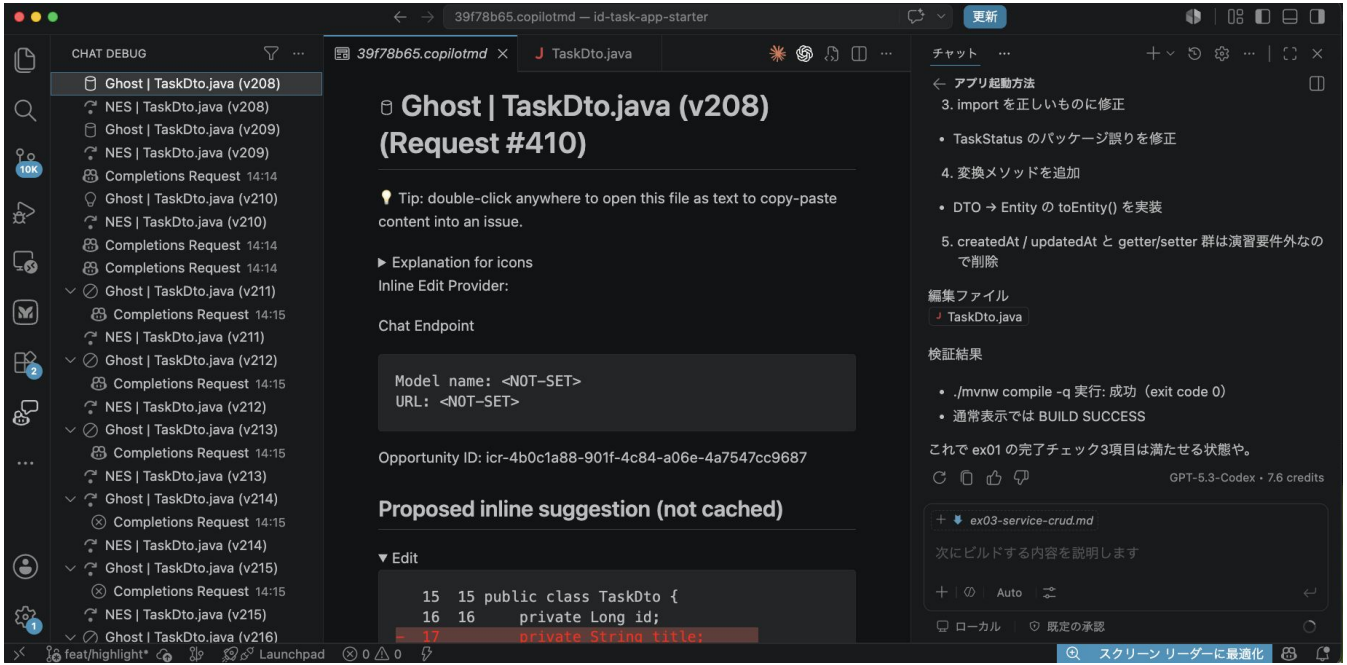
■Session05：Day1クロージング・明日の予告［30min］

*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

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

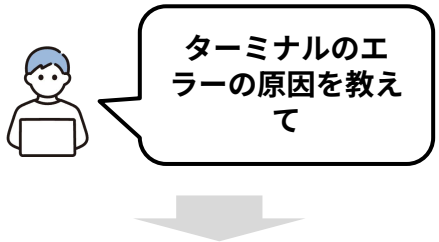
- 各種コマンド（@コマンド、#コマンド、/コマンド）の全体像のご説明
  - 『@コマンド』の説明と演習
  - 『#コマンド』の説明と演習
  - 『/コマンド』の説明と演習
- Custom Instructionsの説明と演習

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

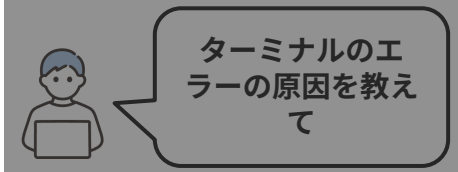
GitHub Copilotには3種類の基本コマンドがあり、それぞれ異なる用途で指示を最適化できます。

	@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！

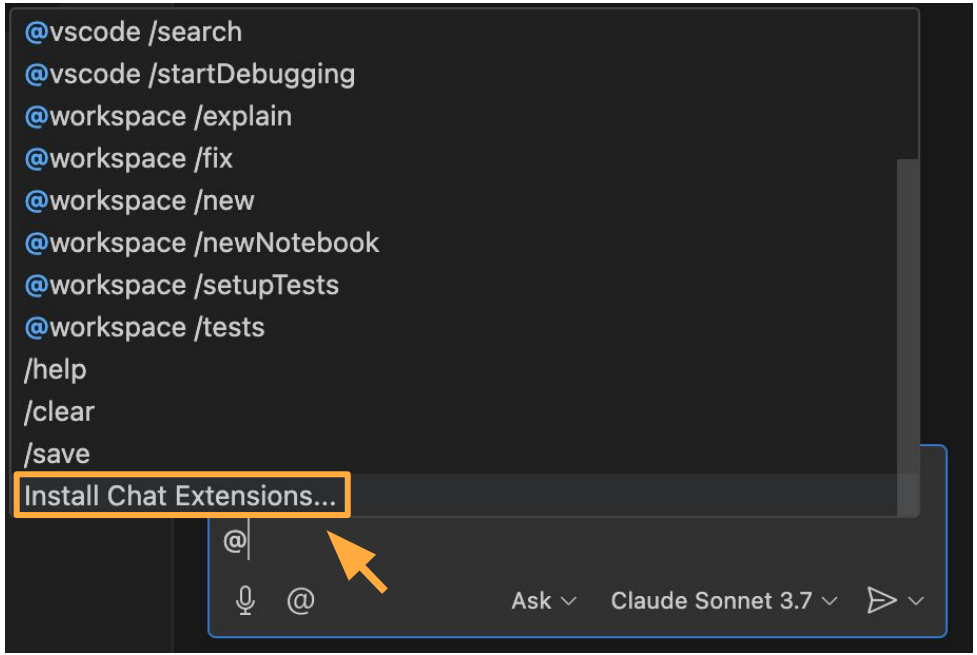
演習手順

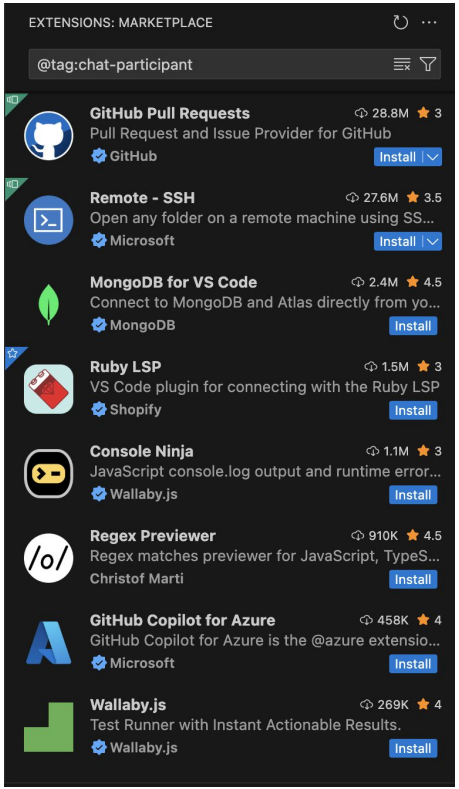
1. cd/src/work_exercise/fix をターミナルに入力
2. python exercise_bug_2.py を実行
3. エラーを確認
4. Copilotチャットで「@terminal /explain」を入力

GitHub Copilot の使い方 - @コマンドの補足

VScode

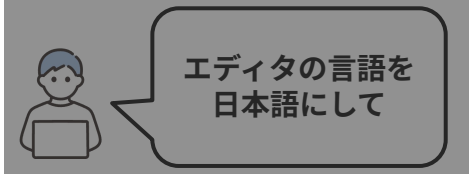
@コマンドには豊富な拡張機能があります。





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

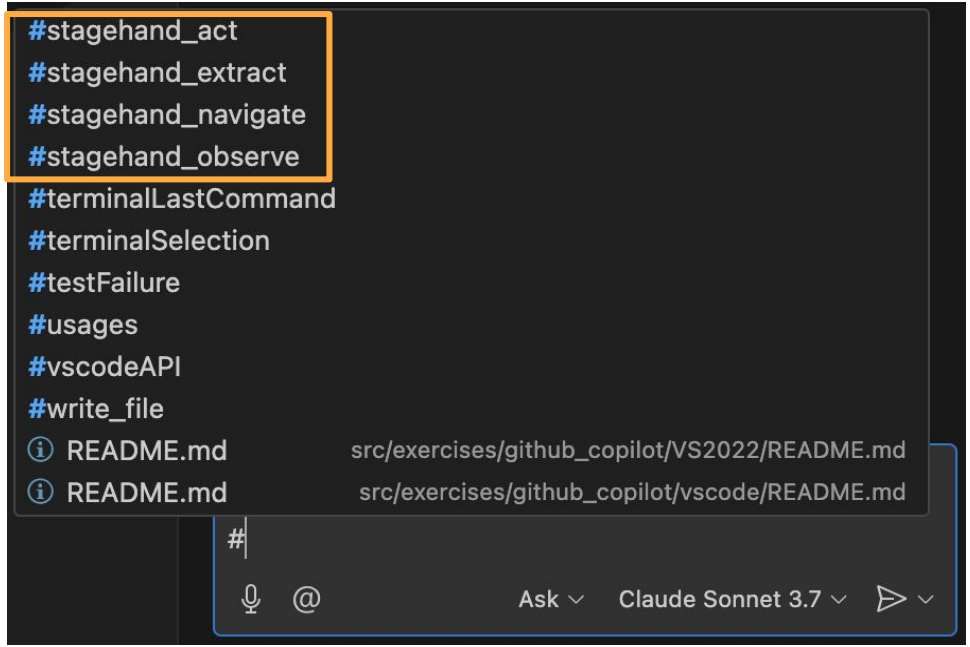
コマンド	説明
#changes	変更されたファイルの差分を取得
#codebase	ワークスペース全体を検索
#editFiles	指定したファイルを編集
#extensions	VS Code Marketplace で拡張機能を検索
#findTestFiles	テストファイル群を検索
#githubRepo	GitHub リポジトリで関連コードを検索
#new	空ディレクトリに新規ワークスペースの土台を作成
#openSimpleBrowser	Simple Browser でローカルサイトをプレビュー
#usages	シンボルの参照・定義・使用箇所を一覧取得

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

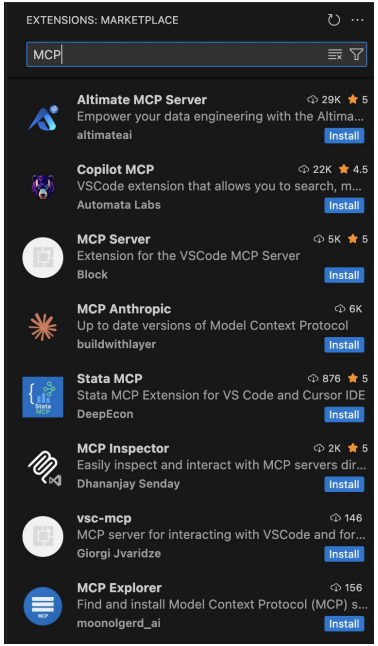
VScode

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。



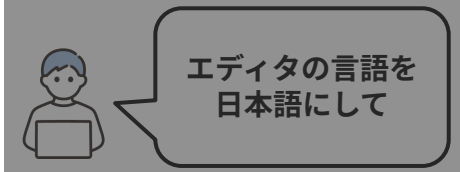
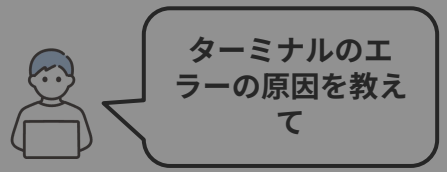
※1
MCP：Agentとツール間でコンテキストを標準化してやり取りする通信プロトコル

※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	–

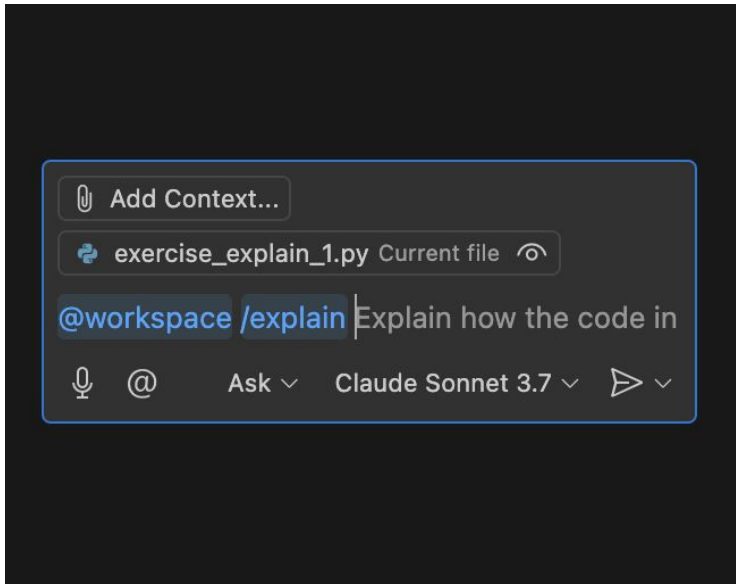
参考：GitHub「GitHub Docs」

GitHub Copilot の使い方 - /explain の使い方

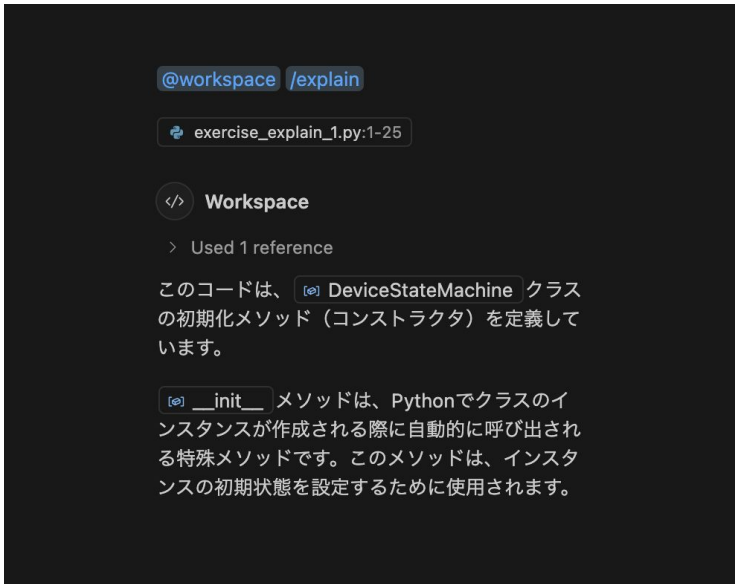
VScode

『/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。

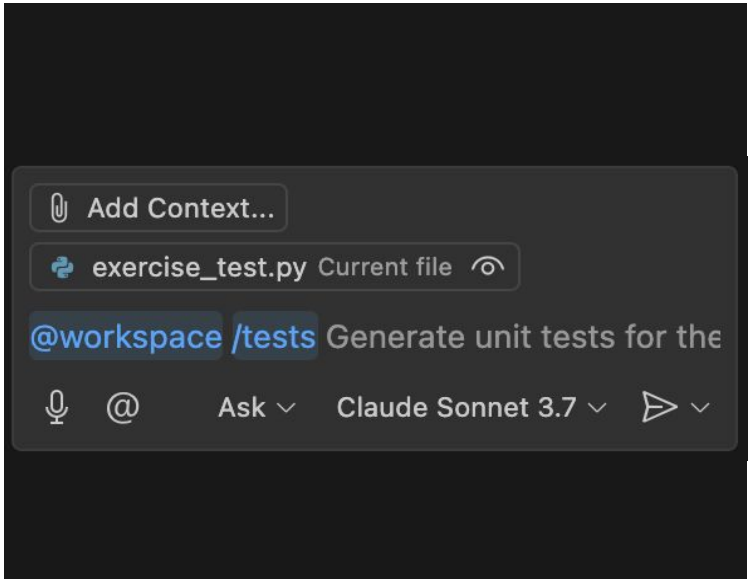
参考：GitHub「GitHub Docs」

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

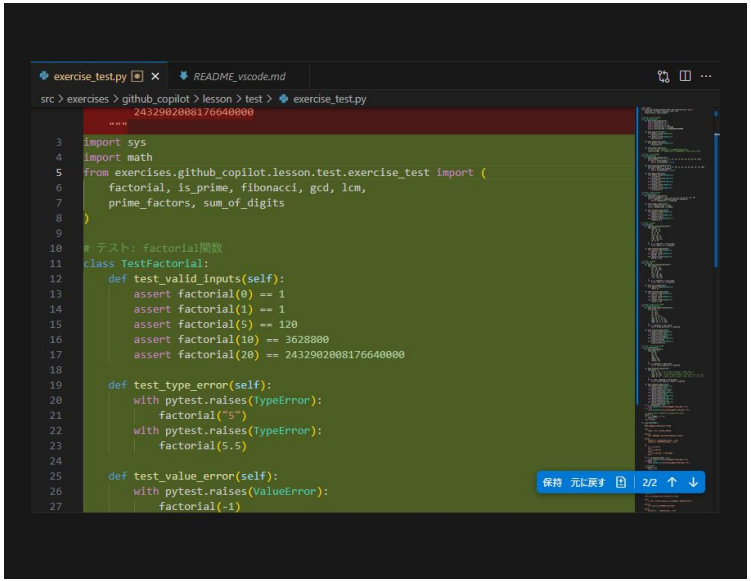
VScode

『/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



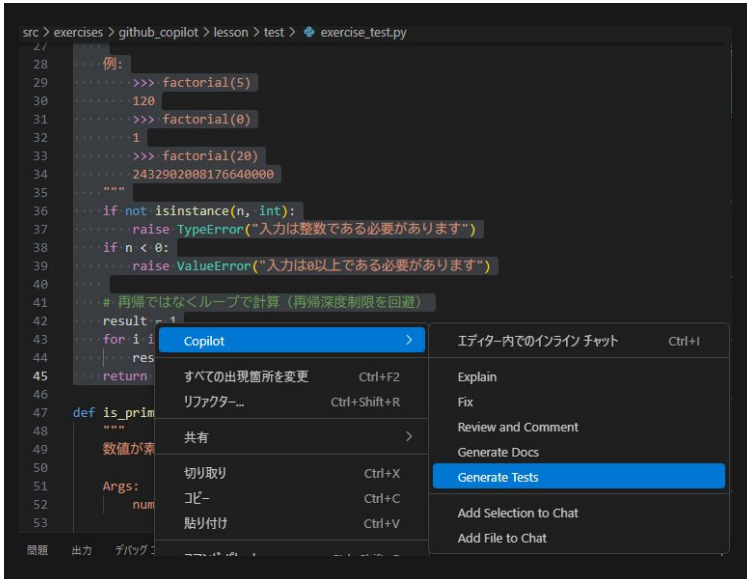
参考：GitHub「GitHub Docs」

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

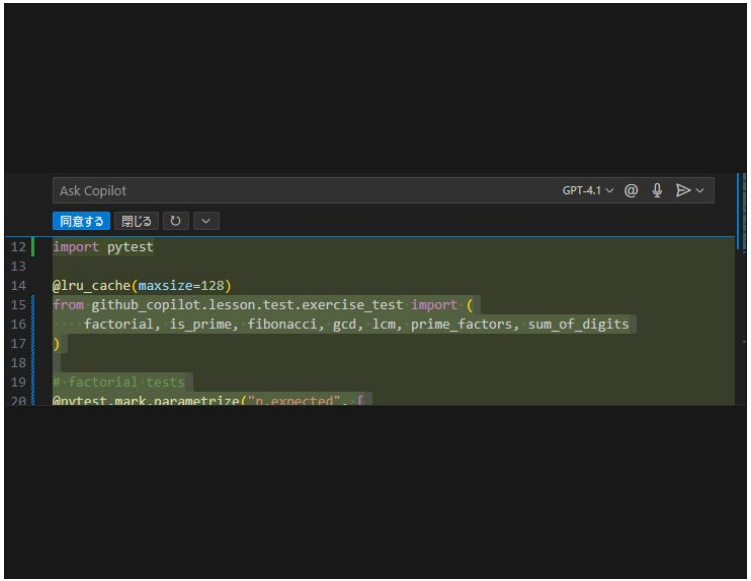
VScode

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」

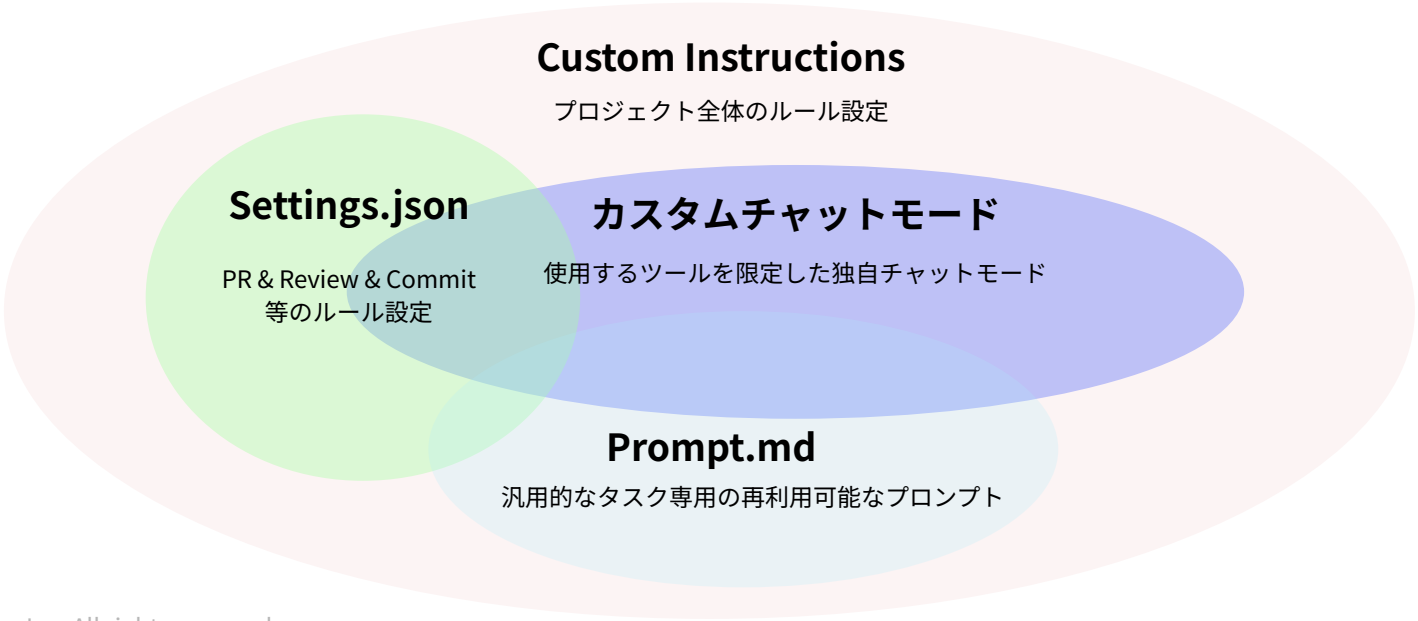
生成AI概論

- 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。

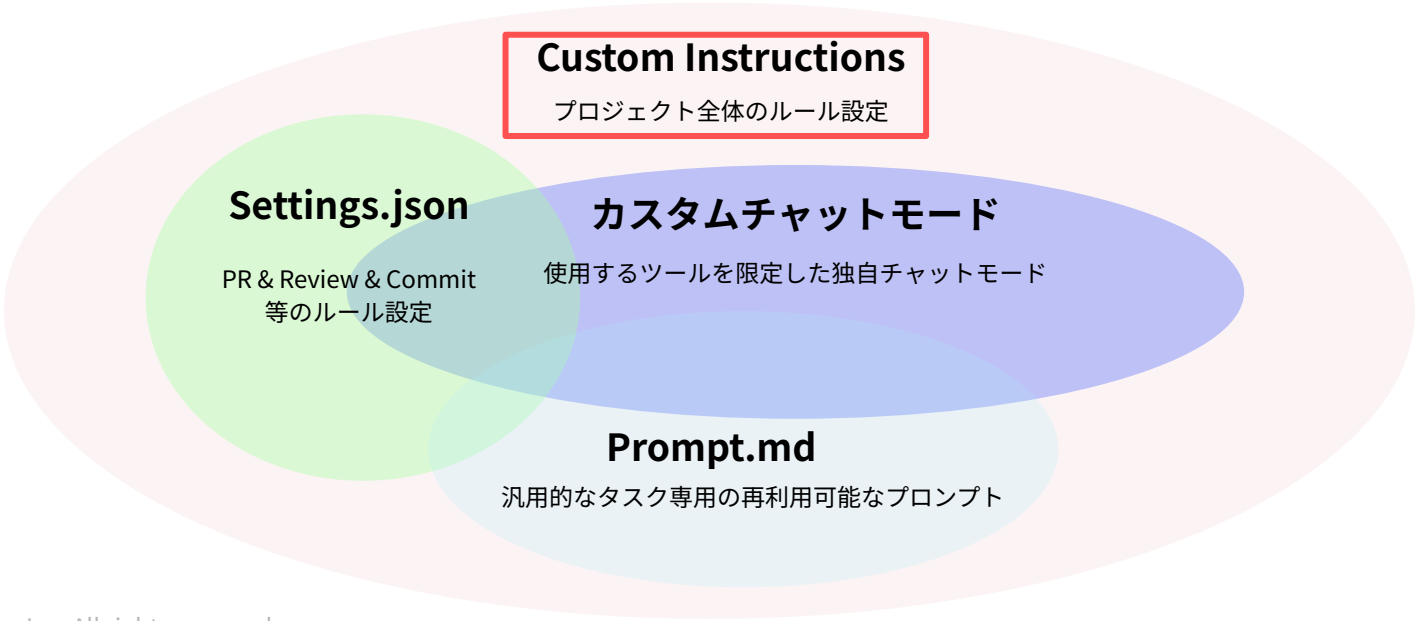
GitHub Copilot のカスタマイズ機能の全体像



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。**

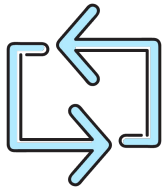
チーム内での設定

**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」

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

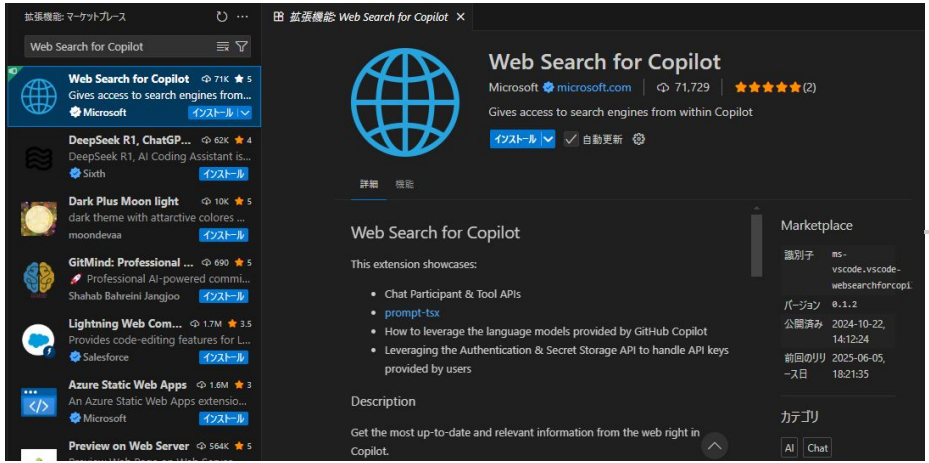
Github Copilot の活用Tips

- 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

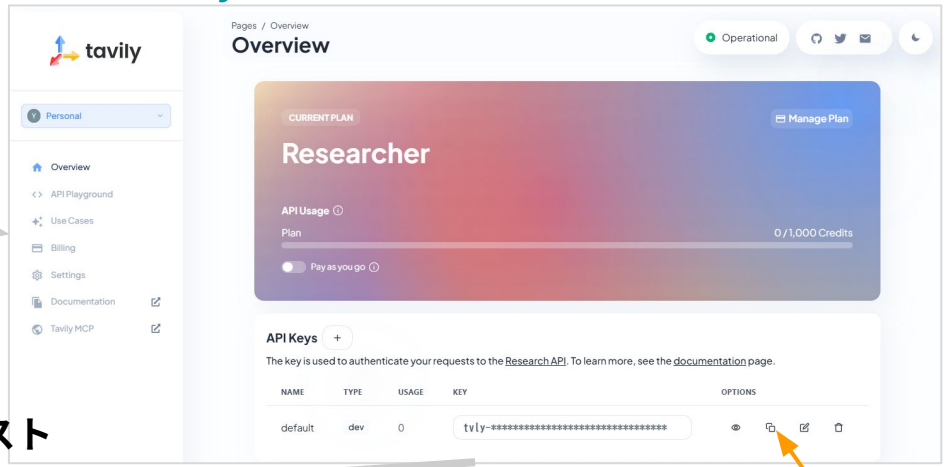
Github Copilot の活用Tips - 拡張機能：Web検索

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。

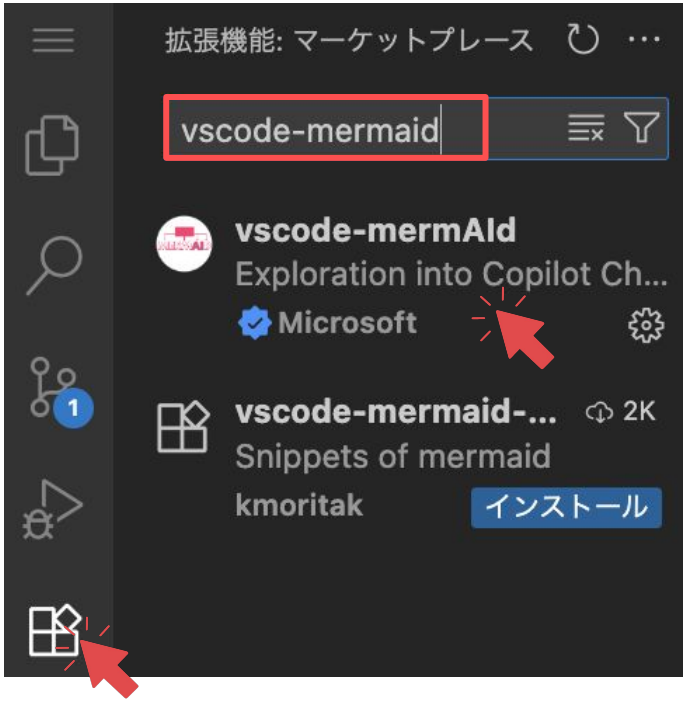
vscode-mermAId 使用イメージ

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

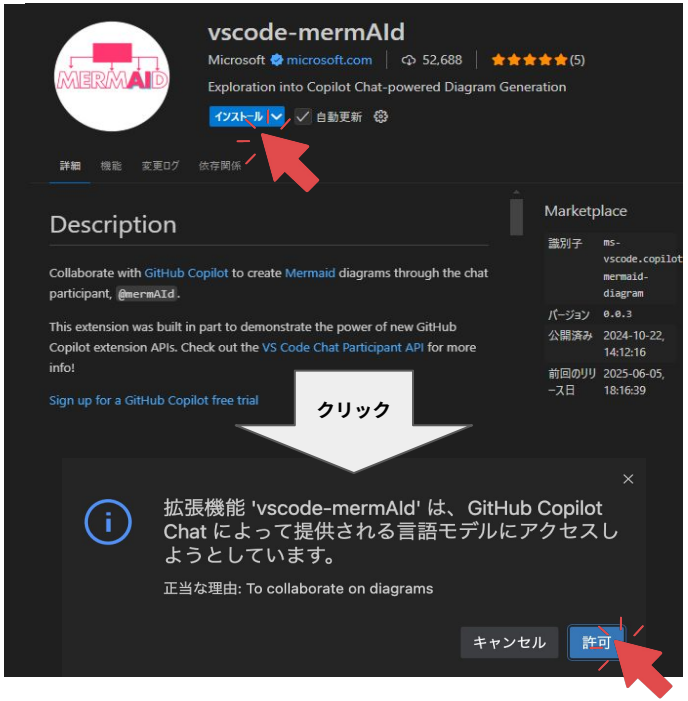
Github Copilot の活用Tips - 拡張機能：vscode-mermAId のインストール

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



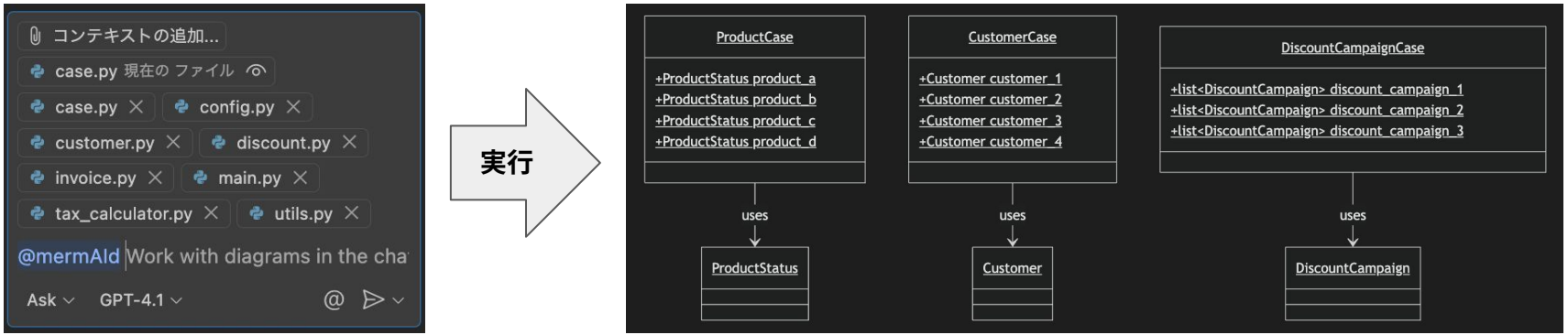
クリック



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



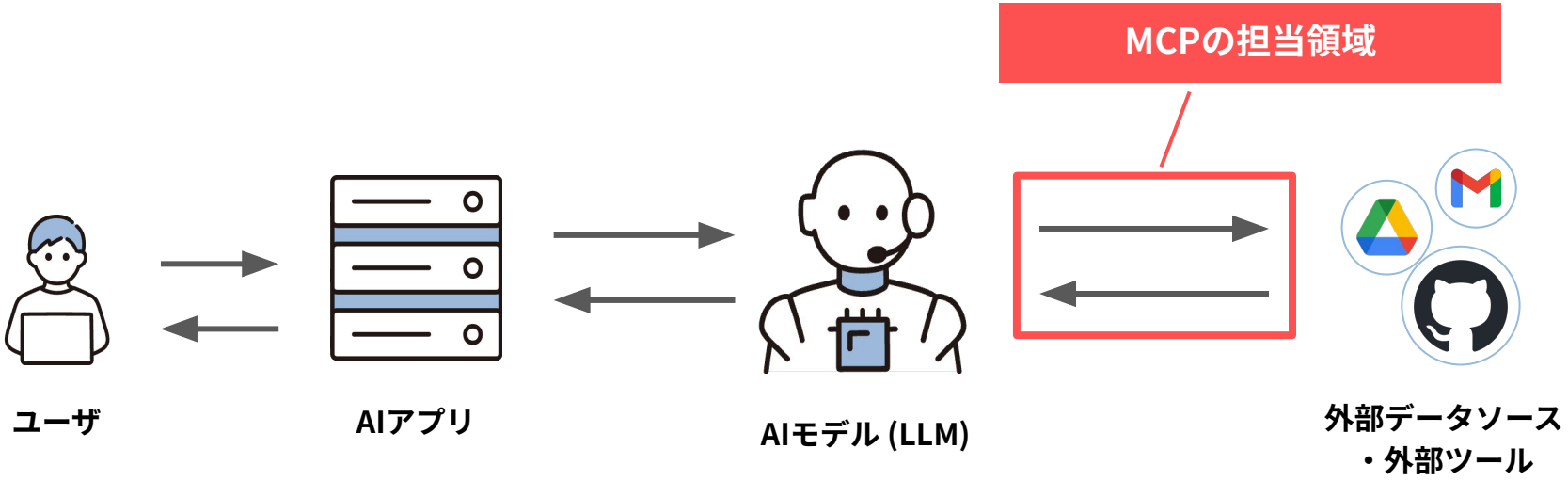
参考：GitHub - Microsoft「vscode-mermAId」

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

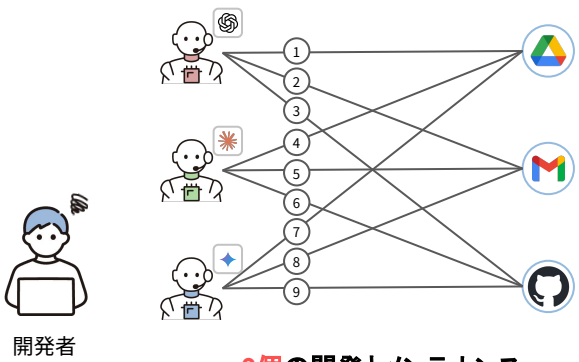
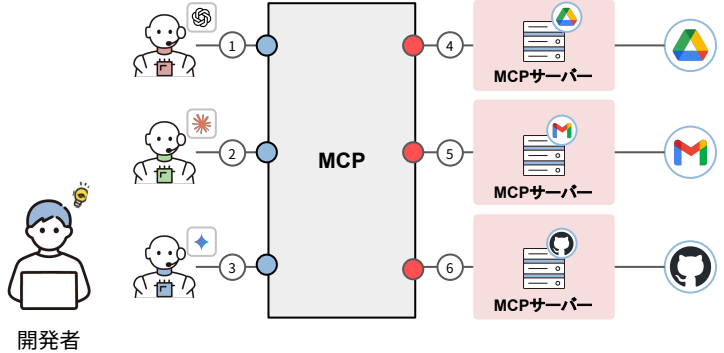
Model Context Protocol（MCP）は、2024年11月にAnthropic社が生み出した
AIモデルと外部のデータやツールとスムーズに連携するプロトコルのことです。



 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」

お昼休憩
12:30-13:30

研修のスケジュール　2日目：テスト・デバッグ・実践演習

TIME：7h

● 座学主体　● ハンズオン主体

■Session01：前回振り返りとDay 2概要［30min］ 前日の振り返りクイズ（5問）｜Copilot基本操作の確認｜Day 2ゴール共有｜テスト・デバッグ・総合演習の概要

■Session02：テストコード生成ハンズオン［130min］

- **JUnitテスト自動生成**　/testsコマンドでテストコード自動生成｜テストメソッドの自動命名確認｜正常系・異常系・境界値の網羅｜null入力・空文字・最大値最小値テスト
- **テスト実行と修正**　生成テストを実行して失敗ケースを特定｜テストが通るよう実装側を修正体験｜例外が正しくスローされるかのテスト実践｜テスト品質の確認と改善サイクル
- **デバッグ・リファクタリング実践**　/fixコマンドでバグ修正｜エラーメッセージから原因特定｜長いメソッド・マジックナンバー｜重複コードの共通化体験｜/explainでJavadoc自動生成

■Session03：総合演習 AI活用ミニ開発［150min］

- **演習課題**　簡易設計書（画面仕様書 or API仕様書）配布｜設計書→実装→テスト→レビューの一連の流れ｜Copilot Chat実装｜手作業との所要時間を計測して比較
- **成果共有・振り返り**　数名が画面共有で成果を発表｜「Copilotが役立った場面・邪魔だった場面」を共有｜2日間の学びを全体で振り返り｜配属後の活用イメージを言語化
- **配属後の活用ガイド**　新プロジェクト参画時のコード理解｜定型コード高速生成｜お客様環境での利用可否判断基準｜AIに任せる/自分で書く場面の整理

■Session04：学習リソース紹介・グループディスカッション［80min］

- **学習リソース**　GitHub Copilot公式ドキュメント｜VSCode公式ドキュメント｜JUnit 5ユーザーガイド｜社内の生成AI活用事例・ナレッジ｜今後の研修ロードマップ
- **グループディスカッション**　3～4名で「配属後に使いたい場面」を話し合う｜使えない現場での代替手段を考える｜セキュリティ上の判断フローを整理｜グループ代表者が発表

■Session05： 質疑応答・アンケート・閉会［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

HANDS-ON

GitHub Copilot 基礎演習

標準機能を使いこなす 5つのカスタマイズ

株式会社IDホールディングス FY26 新卒エンジニア研修

instructions ／ prompts ／ agents ／ skill ／ hooks

Givery

5つのカスタマイズ機能の全体像

Copilotには、文脈を仕込む形が5つあります。

GitHub Copilot　どれも「AIに渡す文脈」をプロジェクトに置く仕組み。効くタイミングと役割が違うだけです。

1	**instructions ｜ 常時効く規約**	書かなくても毎回効く土台
2	**prompts ｜ 呼び出す定型指示**	よく使う指示を /名前 で再利用
3	**agents ｜ 役割を持つモード**	レビュー役など人格を切り替え
4	**skill ｜ 手順の型**	決まった作業を手順書化して渡す
5	**hooks ｜ イベント連動**	保存や実行の節目で自動発火

この演習の前提｜演習3が終わった状態

動くアプリができたこの段階で、Copilotの機能を試します。

演習3まで終えると TaskService のCRUD5メソッドが実装され、/tasks 画面で初期データ10件の一覧・登録・編集・削除が動きます。findOverdue（期限切れ絞り込み）だけは演習7のため未実装です。

いま動くこと（演習3まで）

- ✓ 初期データ10件が /tasks で一覧表示される
- ✓ フォームから新規タスクを登録できる
- ✓ 各行の編集・更新ができる
- ✓ 各行の削除ができる
- ✓ 存在しないIDはエラー画面になる

この演習で足していくもの

- ＋ コーディング規約を instructions 化
- ＋ findOverdue実装を prompt 化
- ＋ レビュー役を agent モード化
- ＋ 機能追加の型を skill 化
- ＋ 保存時チェックを hook 化

① /create-instructions 常時効く規約（instructions）

規約は毎回書かず、**置いておけば自動**で効きます。

> 守らせたい規約を日本語で書くと、applyTo（適用範囲）付きの.instructions.md をCopilotが作ります。置いておくだけで、対象ファイルを触る依頼に規約が毎回自動で差し込まれ、プロンプトに書き添える手間が消えます。

活用シーン

- Service層の作法を常時強制する
- テストにだけテスト方針を効かせる
- アプリ固有の仕様を明文化する
- 規約を共有し出力を全員で揃える

特徴

- 日本語の説明でファイルを自動生成
- globで適用範囲を絞れる
- 呼び出し不要で常時効く
- promptsと違い常時効くのが差

解説

① /create-instructions サンプル

.github/instructions/service-layer.instructions.md

```
--- applyTo: "src/main/java/jp/idhd/taskapp/service/**/*.java" description: "Service層のコーディング規約" ---

# Service層の規約

- 依存はコンストラクタインジェクションで受け取る（フィールドインジェクション禁止）
- Optional は .get() を使わず orElseThrow で受ける
- 1件取得で不在のときは TaskNotFoundException を投げる（汎用の RuntimeException を投げない）
- コメントは日本語で、WHY だけ書く
```

先頭の --- で囲む部分がフロントマター。applyToのglobで適用範囲を絞れ、この例ではservice配下のJavaにだけ規約が効きます。下の本文が依頼のたび自動で渡されます。

ワーク

① /create-instructions プチハンズオン

ゴール **演習3で実装したTaskServiceの作法をカスタム命令にし、次の生成が規約に沿うか見比べる**

1. **考える**
   まず素の状態で『TaskServiceにfindByStatusを追加して』と頼み、コンストラクタ注入やOptionalの扱いを観察してメモする
2. **書く**
   /create-instructionsで、演習3で守った作法を3〜4個書く。applyToがservice配下に絞られているか確認して保存する
3. **実行して見る**
   同じ依頼をもう一度出し、出力が規約どおりに変わったかbeforeと並べて確認する
4. **アレンジ**
   applyToをsrc/test配下にした2枚目を作り、テスト生成にだけ『メソッド名は日本語』を効かせてみる

解説

① /create-instructions 答え合わせ

見本：このプロンプトを送信してみましょう

```
/create-instructions このプロジェクトのService層は、依存をコンストラクタインジェクションで受け取り、Optionalは.get()を使わずorElseThrowで受け、1件取得で見つからないときはTaskNotFoundExceptionを投げる。コメントは日本語。この規約をservice パッケージ配下に効くカスタム命令にしてください。
```

完成イメージ

以降『Serviceにメソッドを追加して』と頼むだけで、コンストラクタ注入・orElseThrow・TaskNotFoundExceptionを使ったコードが返ります。プロンプトに毎回規約を書き添えなくてよくなります。

② /create-prompt 呼び出す定型指示（prompts）

よく使う指示は、/コマンド一発で呼び出せます。

よく使う指示を1つのMarkdown（.prompt.md）に保存し、チャットから /名前 で呼び出せるようにします。
一度書けば同じゴールを短く再利用でき、リポジトリに置けばチームで共有できます。

活用シーン

- findOverdue実装を/コマンド化
- テスト生成を同じ方針で定型化
- レビュー観点を/reviewに固定
- コミット文面を定型で生成

特徴

- 呼んだときだけ動く再利用テンプレ
- model・toolsをファイルに固定
- ${input}で実行時に引数を渡す
- promptsに置けばチーム共有

解説

② /create-prompt サンプル

.github/prompts/implement-overdue.prompt.md

```
--- agent: 'agent' description: 'TaskService.findOverdue を設計書の受け入れ条件どおりに実装する' tools: ['search/codebase'] ---

あなたはSpring Bootに詳しいエンジニアです。`TaskService.findOverdue(LocalDate today)`を実装してください。

受け入れ条件（docs/設計書.md より）：

* dueDate が today より前のタスクだけを返す（today ちょうどは含めない）
* dueDate が null のタスクは含めない
* status が DONE のタスクは含めない

TaskRepository に `findByDueDateBefore(LocalDate date)` を追加して呼び出し、取得結果を上の条件で絞り込むこと。既存の命名規約に合わせてください。
```

> agentに'agent'を指定するとエージェントモードで実行。toolsにsearch/codebaseを渡すと既存構造を見てから書けます。/implement-overdueで呼び出せます。

ワーク

② /create-prompt プチハンズオン

ゴール **findOverdue実装を、その場のチャットでなく再利用できる/コマンドにして一発で走らせる**

1 **考える**
docs/設計書.mdを開き、findOverdueに必要な条件を自分で3つ抜き出す（today未満だけ/ null除外 / DONE除外）

2 **書く**
/create-promptで条件を渡しimplement-overdueを作らせ、本文の3条件が設計書と食い違わないか直す

3 **実行して見る**
チャットで/implement-overdueを実行。compileが通り、一覧の期限切れ行が赤くなれば成功

4 **アレンジ**
本文のtodayを${input:today}に変え、境界値（今日ちょうど）を差し替えて挙動を試す

解説

② /create-prompt 答え合わせ

見本：このプロンプトを送信してみましょう

```
/create-prompt docs/設計書.md の受け入れ条件に従って TaskService.findOverdue を実装するプロンプトを作ってください。
条件は「dueDateがtodayより前だけ・today当日は含めない」「dueDateがnullは除外」「statusがDONEは除外」の3つ。TaskRepositoryにfindByDueDateBeforeを追加して呼び出す方針で。
```

完成イメージ

> /implement-overdue と打つだけで findOverdue が実装され、TaskRepository に findByDueDateBefore が追加されます。compileが通り、一覧画面で期限切れの行が赤く表示されます。

③ /create-agent 役割を持つモード（agents）

役割ごとにモードを切り替えて対話します。

『役割・使ってよいツール・モデル・守る指示』を .agent.md 1枚にまとめ、1つのエージェントとして固定します。
チャット上部のドロップダウンで、レビュー役や設計役に切り替えて対話できます。

活用シーン

- 書き換えない指摘専任レビュー役
- 境界値まで書くテスト作成役
- read-only限定の調査役
- 設計方針を相談する設計役

特徴

- 役割・ツール・モデルを1枚に固定
- ドロップダウンで切り替え
- ツールを絞って暴走を防ぐ
- 選んだ時だけ効く

解説

③ /create-agent サンプル

.github/agents/reviewer.agent.md

```
--- name: Task Reviewer description: TaskService/Controller をnull安全と例外設計の観点で
レビューするtools: ['search/codebase', 'search/usages'] ---

# レビュー指示

あなたはこのタスク管理アプリのコードレビュアーです。次の観点だけ確認してください。

- Optional を .get() で受けていないか（orElseThrow を使うべき）
- 1件取得の不在時に TaskNotFoundException を投げているか
- Controller に業務ロジックが漏れていないか
- 命名がこのプロジェクトの既存クラスと揃っているか

コードは書き換えず、指摘と改善案だけを返してください。
```

> frontmatterで役割・使えるツール・モデルを固定します。toolsを読み取り系に絞ればコードを書き換えません。上部のドロップダウンで切り替えて使います。

ワーク

③ /create-agent プチハンズオン

ゴール **演習4のレビュー題材で、コードを書き換えない指摘専任エージェントを作り指摘の質を比べる**

1. **考える**
   良いレビュー役に何を禁止し何をさせたいか考える（書き換え禁止・指摘のみ・null安全と例外設計を見る）
2. **書く**
   /create-agentでレビュー役を作る。使えるツールを読み取り系だけに絞り、編集させない設定にする
3. **実行して見る**
   素のチャットとレビュー役モードの両方に同じ粗いコードを見せ、指摘の具体度を比べる
4. **アレンジ**
   テスト作成専用モードも作り、境界値まで書かせて役割ごとの出力差を見る

解説

③ /create-agent 答え合わせ

見本：このプロンプトを送信してみましょう

```
/create-agent このタスク管理アプリ用のコードレビュー専任エージェントを作ってください。
観点はOptionalを.get()で受けていないか、1件取得の不在時にTaskNotFoundExceptionを投げているか、
Controllerに業務ロジックが漏れていないか、命名が既存クラスと揃っているか。
使えるツールは検索系だけにして、コードは書き換えず指摘と改善案だけ返す役割にしてください。
```

完成イメージ

レビュー役に切り替えて粗いコードを見せると、コードを書き換えずに『.get()をorElseThrowへ』『不在時はTaskNotFoundExceptionを』といった指摘だけが返ります。

④ /create-skill 手順の型（skill）

決まった作業は、手順書（型）にして渡します。

> 特定作業のやり方を教える指示書フォルダです。SKILL.md に名前・説明・手順を書き、補助mdやスクリプトも同居できます。promptsと違い、説明文（description）を手がかりにCopilotが必要なときへ自分で使います。

活用シーン

- CRUD追加の型で多ファイル変更
- 不具合調査の手順を型化する
- ch03検索機能を型どおり実装
- 補助mdやスクリプトを同梱

特徴

- SKILL.mdに名前・説明・手順を書く
- descriptionを手がかりに自動発動
- 補助ファイルを同居できる
- promptsと違い自律的に動く

解説

④ /create-skill サンプル

.github/skills/task-crud-add/SKILL.md

```
--- name: task-crud-add description: このSpring Bootタスクアプリに CRUD系の機能を足すときに使う。
Controller/Service/Repository/DTOの層分けと JUnitテストの規約に沿わせる。 argument-hint: 追加したい機能
の説明（例： タイトルのキーワード検索） ---

# タスクアプリ CRUD追加の型

このリポジトリは Spring Boot 4系、パッケージは jp.idhd.taskapp。機能を足すときは次の順で作る。

1. Repository: メソッド名から問い合わせが生成される派生メソッドで足す（例 findByTitleContaining）
2. Service: @Service にコンストラクタ注入。不在時は TaskNotFoundException
3. Controller: 受け口だけ。業務ロジックを書かない
4. 返す型: エンティティを直接返さず DTO(record) に詰め替える
5. テスト: JUnit5 + Mockito + AssertJ、メソッド名は日本語

命名規約の詳細は ./naming.md を参照する。
```

SKILL.mdの名前とdescriptionで、Copilotがいつ使うスキルかを判断します。本文の手順どおりに複数ファイルを変更。補助ファイルも同フォルダに置けます。

ワーク

④ /create-skill プチハンズオン

ゴール **発展課題ch03を題材に、CRUD追加の型をスキル化して多ファイル変更を一気通貫で出させる**

1 **考える**
repository→service→controller→templateのどの順で何を変えるか、型に書く手順を自分で並べる

2 **書く**
/create-skillで手順を渡しSKILL.mdを作る。descriptionが『いつ使うスキルか』を的確に表すか確認する

3 **実行して見る**
『タイトルで検索する機能を追加して』と頼み、スキルが呼ばれ4ファイルが型どおり変わるか確認する

4 **アレンジ**
手順に『変更後は必ずcompileを通す』を足し、補助mdを同フォルダに置いて参照させてみる

解説

④ /create-skill 答え合わせ

見本：このプロンプトを送信してみましょう

```
/create-skill このSpring Bootタスクアプリに CRUD系の機能を足すときの型をスキルにしてください。
順番はRepositoryに派生メソッド追加→@ Serviceにコンストラクタ注入で実装→Controllerは受け口だけ→エンティティでなくDTO(record)を返す→JUnit5+Mockito+AssertJでメソッド名は日本語のテスト。
パッケージはjp.idhd.taskapp。いつ使うスキルかの説明も具体的に入れてください。
```

完成イメージ

『タイトルで検索する機能を追加して』の一言で、repository→service→ controller→template の4ファイルが型どおり変更され、検索フォーム付きの一覧が動きます。

⑤ /create-hook イベント連動（hooks）

操作の節目で、シェルを自動実行する強制ゲートです。

エージェントがコード生成やツール実行を行う節目（編集後・実行前・開始時など）に、自作のシェルコマンドを自動で差し込みます。AIへの『お願い』ではなくシェルの確実な実行なので、必ず動く強制ゲートになります。

活用シーン

- 編集直後にcompileを自動実行
- 保存後にテストを自動で回す
- 実行前に危険操作を止める
- 開始時に環境を整える

特徴

- お願いでなく確実に実行する
- 編集後・実行前など節目で発火
- JSONで条件とコマンドを指定
- instructionsと違い決定的に動く

解説

⑤ /create-hook サンプル

.github/hooks/compile-after-edit.json

```
{ "version": 1, "hooks": { "PostToolUse": [
{ "type": "command", "matcher":
"^(edit|create)$", "bash": "./mvnw -q
compile", "powershell": ".\\mvnw.cmd -q
compile", "timeoutSec": 120 } ] } }
```

発火イベント（編集後など）と実行するシェルコマンドをJSONで指定します。お願いでなく確実に走るので、壊れたコードをその場で止められます。

ワーク

⑤ /create-hook プチハンズオン

ゴール **ファイル編集直後に./mvnw compileが走るフックを作り、壊れたコードをその場で気づけるようにする**

1. **考える**
   どのタイミング（編集後/実行前/開始時）に何を走らせれば安全か考える。今回は編集後のcompile
2. **書く**
   /create-hookで『Javaを編集したらcompileを実行』を作る。生成JSONの発火条件とコマンドを確認する
3. **実行して見る**
   わざとコンパイルの通らないコードを出させ、フックが失敗を報告してくれるか確認する
4. **アレンジ**
   compileをtestに変え、演習6のバグ修正直後に自動でテストが回る流れを作ってみる

解説

⑤ /create-hook 答え合わせ

見本：このプロンプトを送信してみましょう

```
/create-hook Copilotが .java ファイルを編集または新規作成したあとに、
毎回 ./mvnw -q compile を自動実行するフックを作ってください。
Mac と Windows 両対応にしたいので bash は ./mvnw、powershell は .\mvnw.cmd を使う。
対象はファイル編集系のツールだけに絞ってください。
```

完成イメージ

Javaファイルを編集するたびに ./mvnw compile が自動で走り、コンパイルが通らないコードを出した瞬間に
エラーが表示されます。壊れたまま先へ進むのを防げます。

使い分けの早見｜迷ったらこれ

「いつ効かせたいか」で選ぶと迷いません。

機能	こう思ったら	具体例
instructions	常に・全部の会話に効かせたい	コーディング規約・言語・レビュー観点
prompts	同じ指示を何度も呼び出したい	テスト生成・コミット文・要約の定型
agents	役割を切り替えて対話したい	レビュアー・設計者・調査役モード
skill	決まった手順で作業させたい	新機能追加の型・不具合調査の型
hooks	操作の節目で自動で走らせたい	保存時の整形・実行前チェック

研修のスケジュール　2日目：テスト・デバッグ・実践演習

TIME：7h

● 座学主体　● ハンズオン主体

■Session01：前回振り返りとDay 2概要［30min］ 前日の振り返りクイズ（5問）｜Copilot基本操作の確認｜Day 2ゴール共有｜テスト・デバッグ・総合演習の概要

■Session02：テストコード生成ハンズオン［130min］

- **JUnitテスト自動生成**　/testsコマンドでテストコード自動生成｜テストメソッドの自動命名確認｜正常系・異常系・境界値の網羅｜null入力・空文字・最大値最小値テスト
- **テスト実行と修正**　生成テストを実行して失敗ケースを特定｜テストが通るよう実装側を修正体験｜例外が正しくスローされるかのテスト実践｜テスト品質の確認と改善サイクル
- **デバッグ・リファクタリング実践**　/fixコマンドでバグ修正｜エラーメッセージから原因特定｜長いメソッド・マジックナンバー｜重複コードの共通化体験｜/explainでJavadoc自動生成

■Session03：総合演習 AI活用ミニ開発［150min］

- **演習課題**　簡易設計書（画面仕様書 or API仕様書）配布｜設計書→実装→テスト→レビューの一連の流れ｜Copilot Chat実装｜手作業との所要時間を計測して比較
- **成果共有・振り返り**　数名が画面共有で成果を発表｜「Copilotが役立った場面・邪魔だった場面」を共有｜2日間の学びを全体で振り返り｜配属後の活用イメージを言語化
- **配属後の活用ガイド**　新プロジェクト参画時のコード理解｜定型コード高速生成｜お客様環境での利用可否判断基準｜AIに任せる/自分で書く場面の整理

■Session04：学習リソース紹介・グループディスカッション［80min］

- **学習リソース**　GitHub Copilot公式ドキュメント｜VSCode公式ドキュメント｜JUnit 5ユーザーガイド｜社内の生成AI活用事例・ナレッジ｜今後の研修ロードマップ
- **グループディスカッション**　3～4名で「配属後に使いたい場面」を話し合う｜使えない現場での代替手段を考える｜セキュリティ上の判断フローを整理｜グループ代表者が発表

■Session05： 質疑応答・アンケート・閉会［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

ディスカッション④

テーブルでディカッション(10分)

GithubCopilotを使いたい
使えそうな業務は？

配属先の業務の中でまず活用したいこと
もしその業務でAIが使えなかったらどうするか
セキュリティ的な懸念点はどこか

ディスカッション⑤

1人2分（合計10分）

プライベートも含め
生成AIをどのように
使ってみたいと思ったか

1人にずついくつか『こんなことをしたい』を言葉に出しましょう

まとめ

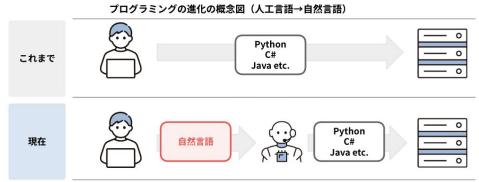
まとめ

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。



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

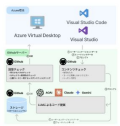
Custom Instructionsを活用することでチーム開発から個人開発まで、
様々なシーンで開発の生産性を向上させることができます。

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	• 受け入れられた・拒否/修正された補完 • 使用統計 • 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	• 👍/👎回答のユーザー評価 • ユーザーのコメント	未来の目的に必要な期間保存
③ プロンプト	• コード出力の内容 • チャットの入力内容 • 関連するコンテキスト	IDE 環境：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 環境：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

※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。

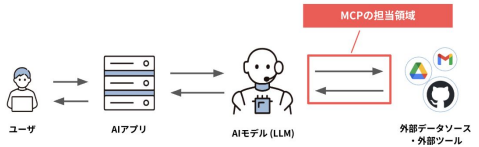
データフローの要約

- 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。

 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」

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

研修のスケジュール　2日目：テスト・デバッグ・実践演習

TIME：7h

● 座学主体　● ハンズオン主体

■Session01：前回振り返りとDay 2概要［30min］ 前日の振り返りクイズ（5問）｜Copilot基本操作の確認｜Day 2ゴール共有｜テスト・デバッグ・総合演習の概要

■Session02：テストコード生成ハンズオン［130min］

- **JUnitテスト自動生成**　/testsコマンドでテストコード自動生成｜テストメソッドの自動命名確認｜正常系・異常系・境界値の網羅｜null入力・空文字・最大値最小値テスト
- **テスト実行と修正**　生成テストを実行して失敗ケースを特定｜テストが通るよう実装側を修正体験｜例外が正しくスローされるかのテスト実践｜テスト品質の確認と改善サイクル
- **デバッグ・リファクタリング実践**　/fixコマンドでバグ修正｜エラーメッセージから原因特定｜長いメソッド・マジックナンバー｜重複コードの共通化体験｜/explainでJavadoc自動生成

■Session03：総合演習 AI活用ミニ開発［150min］

- **演習課題**　簡易設計書（画面仕様書 or API仕様書）配布｜設計書→実装→テスト→レビューの一連の流れ｜Copilot Chat実装｜手作業との所要時間を計測して比較
- **成果共有・振り返り**　数名が画面共有で成果を発表｜「Copilotが役立った場面・邪魔だった場面」を共有｜2日間の学びを全体で振り返り｜配属後の活用イメージを言語化
- **配属後の活用ガイド**　新プロジェクト参画時のコード理解｜定型コード高速生成｜お客様環境での利用可否判断基準｜AIに任せる/自分で書く場面の整理

■Session04：学習リソース紹介・グループディスカッション［80min］

- **学習リソース**　GitHub Copilot公式ドキュメント｜VSCode公式ドキュメント｜JUnit 5ユーザーガイド｜社内の生成AI活用事例・ナレッジ｜今後の研修ロードマップ
- **グループディスカッション**　3～4名で「配属後に使いたい場面」を話し合う｜使えない現場での代替手段を考える｜セキュリティ上の判断フローを整理｜グループ代表者が発表

■Session05： 質疑応答・アンケート・閉会［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

End of File