（5）「AI 活用プロジェクト演習①PG 編」改訂・追加開発教材レベル 1

「AI活用プロジェクト演習①PG編」
PBL教材
レベル１

202X.XX.XX

チャットボット型／コード補完ツール

目次

1 はじめに

2 AIの概要と活用方法①　チャットボット型生成AI

3 ワーク①　チャットボット型生成AIでプログラミング

4 AIの概要と活用方法②　コード補完ツール

5 ワーク②　コード補完ツールの機能体験

6 ワーク③　AIの差を発表

7 まとめ

2

目次

1 はじめに

2 AIの概要と活用方法①　チャットボット型生成AI

3 ワーク①　チャットボット型生成AIでプログラミング

4 AIの概要と活用方法②　コード補完ツール

5 ワーク②　コード補完ツールの機能体験

6 ワーク③　AIの差を発表

7 まとめ

3

本日のゴール

生成AIはプログラミングの強力なサポートになることを知る

1 生成AIに効果的な指示を出せるようになる

2 生成AIを時短ツールとして活用するヒントを得る

3 生成AIは私たちのパートナーになり得ることを体感する

4

講義の進め方

ワークを用いて効率的な使い方を体験

AIの効率的な使い方を紹介

実際にワークで活用してみる

それぞれのAIの性能を比較する

5

AIを「道具」から「パートナー」へ

AIは従来のツールとは大きく異なり、
独自のコンテンツを生み出すことが可能

これまでのAI

単なる「道具」として、特定のタスクを自動化・効率化。
→検索エンジンの代替、簡単な翻訳



これからのAI

パターンや関連性の学習から、独自のコンテンツを生み出す。
→パートナーとして共に価値を創出する



6

パートナーとは？

人間とAIで得意なことを任せあえる関係



AIの得意なことを任せる
（コード生成やアイデアの提案）

人間にしかできないことを任せる
（要件定義や倫理・法的判断）



人間

AI

7

目次

1 はじめに

2 AIの概要と活用方法①　チャットボット型生成AI

3 ワーク①　チャットボット型生成AIでプログラミング

4 AIの概要と活用方法②　コード補完ツール

5 ワーク②　コード補完ツールの機能体験

6 ワーク③　AIの差を発表

7 まとめ

8

チャットボット型AIでプログラミングする流れ

プログラミングを生成AIと対話形式で行う



前提条件を伝える



コーディング



人の目で確認

（もし不備があった場合）



役割を伝える

プロンプトの3つの鉄則

プロンプトは3つの鉄則を守ることで、
より精度の高い回答を得ることができる。

具体的かつ明確に伝える

目的を明確にする

段階的に依頼する

鉄則1：具体的かつ明確に伝える

曖昧な依頼は避け、言語や処理を具体的に伝える

悪い例

「関数を作ってください」



曖昧な依頼

良い例

「Pythonで2つの数を受け取り、
その和を返す関数を作ってください」



明確な依頼

11

鉄則2：目的を明確にする

目的を細かく伝えると正確な回答が得られる

悪い例

「このコードのバグを直してください」



曖昧な目的

良い例

「このPythonコードのバグを直して、
リストのインデックス範囲外エラーが
起きないようにしてください」



明確な目的

12

鉄則3：段階的に依頼する

順を追った依頼で無駄を減らす

悪い例

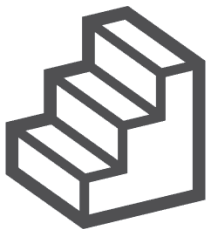
「ゲームを作って」



大きすぎる依頼

良い例

「まずタイトル画面の表示処理を書いてください。
その後プレイヤーの移動処理を依頼します」



段階的な依頼

13

プロンプトの3つの鉄則

3つの鉄則まとめ

具体的かつ明確に伝える

目的を明確にする

段階的に依頼する

14

3つの鉄則まとめ

これまでの鉄則を組み合わせる



AIが正確に意図を読み取れる

良い例

PythonでCSVファイルを読み込み、
売上データを月ごとに集計してください。

まずは集計処理だけを書いてください。
その後、matplotlibを使って
社内会議用の棒グラフにしたいと考えています。

3つの鉄則を使って実際にプロンプトを入力した結果

PythonでCSVファイルを読み込み、売上データを月ごとに集計してください。
まずは集計処理だけを書いてください。その後、matplotlibを使って☒社内会議用の棒グラフにしたいと考えています。

承知しました。まずは「CSVファイルを読み込んで、売上データを月ごとに集計する処理」だけをPythonで書きます。ここでは **pandas** を使います。

python　コードをコピーする

```python
import pandas as pd

# CSVファイルを読み込み
# 例: sales.csv に "date","amount" の列があると仮定
df = pd.read_csv("sales.csv", parse_dates=["date"])

# 月ごとに集計
monthly_sales = df.groupby(df["date"].dt.to_period("M"))["amount"].sum()

# 結果を確認
print(monthly_sales)
```

このコードでは：

- `parse_dates=["date"]` で日付を自動的に `datetime` 型に変換。
- `dt.to_period("M")` を使って「年月単位」に変換。
- `groupby` と `sum()` で月ごと合計を算出。

次のステップでは、この `monthly_sales` を `matplotlib` を使って棒グラフ化できます。

👉 CSVの列名（例: `date`, `amount`）は仮定ですが、実際のファイルに合わせて教えていただければ調整できます。列名を確認してから棒グラフ用コードを書きましょうか？

チャットボット型AI活用事例紹介

具体的な活用方法から生成AIでできることを知る



1.リサーチ＆情報収集

2.企画＆アイデア出し

3.文章作成＆要約＆校正

4.プログラミング学習

17

活用事例1：リサーチ＆情報収集のプロンプト例

リサーチ・情報収集の対象を具体的に伝えることで精度の高い解答が得られる



プロンプト例

「GitHub CopilotとAmazon Q Developerの違いを、
サポート言語と利用料金の観点から整理してください」

ポイント

・漠然とした要求

・**複数の視点**から情報を整理させることができる。

18

活用事例2：企画・アイデア出しのプロンプト例

企画・アイデア出しのパートナーとして、多角的に判断を行うための助けにする



プロンプト例

「ファッションECアプリを利用する10代女性ユーザーが
『もっとワクワクする』と思う機能を10個提案してください」

ポイント

・**多様な視点からのアイデア**を瞬時に得られる。

・煮詰まったときの**思考のヒント**になる。

19

活用事例3：文章作成・要約・校正

工数がかかる文章作成をスマートに解決



プロンプト例

「この会議議事録を200字以内に要約し、
結論と次のアクションだけを残してください」
[ここに文章を挿入]

ポイント

・**誰にでも伝わる文章**に書き換えられる

・長文のポイントを**瞬時に抽出**できる。

・**誤字脱字のチェック**や表現の提案も可能

20

活用事例4：プログラミング学習

AIと一緒にプログラミングの壁を乗り越える



プロンプト例

あなたはプログラミングの先生です。
Pythonのデコレーターについて、
初心者向けに簡単なコード例を交えて解説してください。

ポイント

- **専門用語**を分かりやすく説明してくれる
- **エラーの原因**を特定し、解決策を提案してくれる
- **コードの書き方や構造**についてアドバイスをくれる

21

効率的な使い方のコツ

あなたが成長するたびに、AIは「賢くなる」



プロンプトを具体的に書く

AIはプロンプトからしか情報を得られない。
「**何をしたいか**」を明確に。



ツールに任せすぎない

AIの提案はあくまで「候補」。
必ず**自分でコードレビュー**を行う。



ツールを先生として活用する

AIが提示したコードから、
「**新しい書き方**」や「**知らないライブラリ**」を学ぶ。

22

注意点とリスク

人間が安全面の責任を持つことで、AIは初めて活用できる



セキュリティと機密情報

非公開のコードや個人情報を入力しない



著作権とライセンス

AIが生成したコードの
ライセンス形態を把握しておく



倫理的な問題

AIに不適切なコードを生成させない

23

目次

1 はじめに

2 AIの概要と活用方法①　チャットボット型生成AI

3 ワーク①　チャットボット型生成AIでプログラミング

4 AIの概要と活用方法②　コード補完ツール

5 ワーク②　コード補完ツールの機能体験

6 ワーク③　AIの差を発表

7 まとめ

24

本日のワークの説明

人間と生成AIのプログラミングスキルを実際に比較する。

ワーク①　：チャットボット型生成AIと人間の比較

事前準備　：GitHub Copilotの導入

ワーク②　：GitHub Copilotと人間の比較

ワーク③　：AIのメリットや強みの考察・発表

25

本日のワークの説明

人間と生成AIのプログラミングスキルを実際に比較する。

ワーク①　：チャットボット型生成AIと人間の比較

事前準備　：GitHub Copilotの導入

ワーク②　：GitHub Copilotと人間の比較

ワーク③　：AIのメリットや強みの考察・発表

26

本日のワークの説明：問題の一覧

本日のワークでは多くの問題に挑戦する。どんどん試して欲しい

コード生成	バグ修正	リファクタリング
1-1 数当てゲーム	2-1 無限ループ修正	3-1 非効率コード最適化
1-2 複雑な問題	2-2 変数名参照ミス修正	3-2 ネスト構成最適化
1-3 メールフォーム作成	2-3 アルゴリズム修正	3-3 重複最適化

27

ワーク①：チャットボット型AIと人間の比較　問1　コード自動生成

コード自動生成を体験

問1の取り組み方
下記のコードをチャットボット型生成AIに生成させた上で、
事前課題で取り組んだ自分で書いたコードと比較してみてください。

28

ワーク①：チャットボット型生成AIと人間の比較　問1-1　数当てゲーム

問1-1(難易度：易しめ)
1～100の乱数を1つ生成する。ユーザーは数値を入力し、当たるまで繰り返す。

各回、大きい／小さいのヒントを表示し、当たったら試行回数とともに終了。

ワーク①：チャットボット型生成AIと人間の比較　問1-2　複雑な問題

問1-2(難易度：難しい)
高橋君はお酒を N杯飲みました。
i番目に飲んだお酒は、量がVml、アルコール度数がPi% です。
高橋君はアルコールの摂取量がXml を超えると酔っ払います。
高橋君が酔っ払ったのは何杯目のお酒を飲んでいるときですか。
ただし、N杯全てのお酒を飲んだあとでも酔っ払っていない場合は、
かわりに -1 を出力してください。

制約	入力	入力例	出力例
入力は全て整数 $1 \leq N \leq 10^3$ $0 \leq X \leq 10^6$ $1 \leq Vi \leq 10^3$ $0 \leq Pi \leq 100$	入力は以下の形式で 標準入力から与えられる。 N　X V1 P1 ・ ・ ・ VN PN	2 15 200 5 350 3	2

出典:https://atcoder.jp/contests/abc189/tasks/abc189_b

ご使用できるプログラミング言語上、問1-1や問1-2に取り組むことが難しい場合はこの問題から取り組んでください。

問1-3
名前・メールの入力フォームを作成をしてください。

要件
・ページ左上に「お問い合わせ」の文字を出す。
・その下に「お名前」の入力欄、「メールアドレス」の入力欄を用意する。
・その下に送信ボタンと入力内容をクリアするボタンを用意する。
・送信ボタンを押すと「送信ありがとうございました。」の表示される。

バグ修正機能を体験

課題実行手順

1.まず ①生成用プロンプトの{言語}を
自分の使用する言語にあったものに編集してからチャットボット型AIに貼り付けて、
あなたの言語で“バグ入りコード”を出してもらう。

2.生成されたコードを手元で実行し、バグを再現（ログ／挙動を確認）。

3.その同じコードを本文ごとチャットボット型AIに渡して、②修正用プロンプトでも
①生成用プロンプトと同様の編集をしてからチャットボット型AIに貼り付けて、修正を依頼。

4.修正コードを実行して直ったことを確認。ついでにテストや改善提案も活用。

言語指定の例

・{言語} → Python / JavaScript(Node) / C / Java / VBA(Excel)

①生成用プロンプト(これを編集してチャットボット型生成AIに貼り付けましょう)

{言語}であえてバグを含む最小プログラムを1ファイルで作ってください。
テーマは「無限ループのバグ再現」です。要件は次の通り：
- 外部ライブラリは使わない（標準のみ）。
- 目的：リトライ処理やカウンタ制御の不備によりループが停止しなくなる様子を再現。
- “動くが止まらない”状態を意図的に作る（例：カウンタ未更新／終了条件の論理ミス）。
- タイムアウト・強制break等の安全装置は入れない。

以上を満たし、作成してください。

②修正用プロンプト(これを編集してチャットボット型生成AIに貼り付けましょう)

次の{言語}コードは無限ループが発生します。原因を特定し、
品質を落とさずに修正してください。
要件：
- 原因の説明 → 修正方針 → 修正後の全文コード（1ファイル）の順で提示。
- 可能なら構造的に防ぐ書き方（例：forで回数を制御、最大リトライの明示）も提案。
- 実行手順と、期待されるログの変化例を示す。
- 簡単なテスト観点（成功即終了／連続失敗／境界値の最大試行回数）を箇条書き。
【対象コード】
<<<ここに生成されたコード全文>>>

①生成用プロンプト(これを編集してチャットボット型生成AIに貼り付けましょう)

{言語}であえてバグを含む最小プログラムを1ファイルで作ってください。
テーマは「変数名の参照ミス（未定義変数の使用）」です。
要件は次の通り：
- 目的：関数引数やループ対象と、実際に参照する変数名が食い違い、実行時にエラー（未定義/NameError/Null参照）や誤動作が起きる様子を再現。
- 一見もっともらしいが実行すると失敗する状態にする（例：for文で引数名ではなく未定義の配列名を参照）。
- 例外処理などでバグを隠さない（そのまま落ちる／誤結果が出る）。

以上を満たし、作成してください。

②修正用プロンプト(これを編集してチャットボット型生成AIに貼り付けましょう)

次の{言語}コードは「変数名の参照ミス」により失敗します。
原因を特定し、正しく動作するよう修正してください。
要件：
- 原因の説明（どの名前が不一致か、どの行で例外/誤動作が起きるか）
- 修正後の全文コード（1ファイル）を提示し、動作確認手順を記載
- 最小テスト観点（空配列／単一要素／複数要素／負値混在 等）を箇条書き
- 将来の再発防止策（変数名の規約／静的解析／リンタ活用）も提案
【対象コード】
<<<ここに生成されたコード全文>>>

①生成用プロンプト(これを編集してチャットボット型生成AIに貼り付けましょう)

{言語}であえてバグを含む最小プログラムを1ファイルで作ってください。
テーマは「アルゴリズムの組み立てミス（仕様と異なる実装）」です。
要件は次の通り：
- 目的：仕様は「重複を除き、**元の出現順**で要素を並べる」こと。しかし実装では集合やソートを使い、順序が壊れる/意図と異なる結果になる様子を再現。
- コードは“一見正しそう”だが仕様とズレる状態にする（小さい入力だと気づきにくいが、複雑な入力で破綻が見える）。
以上を満たし、作成してください。

②修正用プロンプト(これを編集してチャットボット型生成AIに貼り付けましょう)

次の{言語}コードは、アルゴリズムの組み立てが仕様と異なっています。
仕様：「重複を除き、元の出現順で要素を並べる」こと。
要件：
- どの入力で仕様不一致が顕在化するかを示し、原因（集合化やソート等による順序破壊）を説明。
- 仕様どおりに直した全文コード（1ファイル）を提示。
- 時間計算量・空間計算量のトレードオフを簡潔にコメント。
- 簡単なテスト群（例：既にユニーク／全要素重複／順序が重要なケース／大規模データのスモーク）を箇条書き。
【対象コード】
<<<ここに生成されたコード全文>>>

リファクタリング機能を体験

課題実行手順

1.まず ①生成用プロンプトの{言語} を
自分の使用する言語にあったものに編集してからチャットボット型AIに貼り付けて、
あなたの言語で“動作は正常だが読みにくいコード”を出してもらう。

2.生成されたコードを手元で実行し、動作が「正しいが読みにくい／冗長」 なことを確認。

3.その同じコードを本文ごとチャットボット型AIに渡して、②リファクタリング依頼用プロンプト
でも①生成用プロンプトと同様の編集をしてからチャットボット型生成AIに貼り付けて、改善を依頼。

4.改善されたコードを実行して、動作が変わっていないことを確認。

言語指定の例

・{言語} → Python / JavaScript(Node) / C / Java / VBA(Excel)

①生成用プロンプト(これを編集してチャットボット型生成AIに貼り付けましょう)

{言語}で「冗長で非効率なコード」をあえて作ってください。
要件：
- テーマ：配列やリストの最大値を求める処理。
- 基本構文だけで書く（for文やif文を多用）。
- 既に存在する組み込み関数やライブラリ（例：max関数など）は使わない。
- 変数名も分かりにくく、ネストも深く、読みにくいコードにしてください。
- 実行すると一応正しい最大値は得られるが、改善の余地が大きい状態に。

ワーク①：チャットボット型生成AIと人間の比較　問3-1　非効率コード最適化　修正用プロンプト

②改善用プロンプト（これを編集してチャットボット型生成AIに貼り付けましょう）

次の{言語}コードは、動作は正しいものの冗長で読みづらいです。リファクタリングしてください。
要件：
- 冗長さやネストを減らし、標準関数を活用。
- 変数名をわかりやすく改善。
- 修正後の全文コードを提示。
- 改善点を箇条書きで整理。
- 簡単なテストケース（例: 正常、負数混在、要素1つ）も提示。

【対象コード】
<<<ここに先ほど生成されたコード全文を貼る>>>

41

ワーク①：チャットボット型生成AIと人間の比較　問3-2　ネスト構成最適化　生成用プロンプト

①生成用プロンプト（これを編集してチャットボット型生成AIに貼り付けましょう）

{言語}で「FizzBuzz問題」を、わざとネストが深く読みにくい形で書いてください。
要件：
- forループを使って1〜nまで表示。
- ifの入れ子を多用して条件が複雑に見えるように。
- コードは正しく動作するが、とても読みにくいこと。

42

ワーク①：チャットボット型生成AIと人間の比較　問3-2　ネスト構成最適化　修正用プロンプト

②改善用プロンプト（これを編集してチャットボット型生成AIに貼り付けましょう）

以下の{言語}コードはFizzBuzzとして動きますが、条件分岐が深く読みにくいです。
リファクタリングしてください。
要件：
- ネストを減らし、条件式をシンプルに。
- 冗長なコードを整理。
- 修正後の全文コードを提示。
- 改善点を箇条書きで整理。
- テスト観点（例: 15, 3, 5, 7など）を提示。

【対象コード】
<<<ここに先ほど生成されたコード全文を貼る>>>

ワーク①：チャットボット型生成AIと人間の比較　問3-3　重複最適化　生成用プロンプト

①生成用プロンプト（これを編集してチャットボット型生成AIに貼り付けましょう）

{言語}で「ユーザー情報を表示する関数」をわざと重複が多い形で作ってください。
要件：
- 引数: name, age, email, isActive (ブール値)。
- Active/Inactiveを判定して表示するが、表示処理のコードを重複させる。
- printやConsole.WriteLineなど出力は2パターンに分かれているが、中身はほぼ同じ。
- 冗長で保守性が悪い状態にする。

ワーク①：チャットボット型生成AIと人間の比較　問3-3　重複最適化　修正用プロンプト

②改善用プロンプト(これを編集してチャットボット型生成AIに貼り付けましょう)

次の{言語}コードは重複が多く、保守性が低いです。リファクタリングしてください。
要件：
- 重複部分を関数化してまとめる。
- 変数名や関数名をわかりやすく改善。
- 修正後の全文コードを提示。
- 改善点を箇条書きで整理。
- 簡単なテスト観点（ActiveとInactive両方のケース）を提示。

【対象コード】
<<<ここに先ほど生成されたコード全文を貼る>>>

ワーク①：AIの強みを考察・発表

各問から判明したAIのメリットや強みの考察・発表

問	手動と比較した際のチャットボット型生成AIの強み・メリットは？
1.コード生成	
2.バグ修正	
3.リファクタリング	
まとめ	

目次

1 はじめに

2 AIの概要と活用方法①　チャットボット型生成AI

3 ワーク①　チャットボット型生成AIでプログラミング

4 AIの概要と活用方法②　コード補完ツール

5 ワーク②　コード補完ツールの機能体験

6 ワーク③　AIの差を発表

7 まとめ

コード補完ツールの概要

AIはあなたの開発をどう変えるのか？



コード補完ツールとは？

・AIが文脈を読んでコードの候補を自動で提案してくれるツール

なぜ必要？

・開発速度の向上、学習コストの削減、メンタルモデルの向上

今回紹介する機能一覧

1. 基本的なコード補完
2. チャット機能を用いたコード生成
3. チャットモードの切り替え
4. コンテキストの使い方
5. メンション機能の使い方
6. コマンド機能の使い方



「思考を補助する相棒」として活用

途中まで書いたコード

```python
# FizzBuzzを実装
def fizzbuzz(n):
    for i in range(1, n+1):
        if i % 3 == 0 and i % 5 == 0:
            # ここで止めると…
```

補完機能を使ったコード

```python
# FizzBuzzを実装
def fizzbuzz(n):
    for i in range(1, n+1):
        if i % 3 == 0 and i % 5 == 0:
            print("FizzBuzz")
        elif i % 3 == 0:
            print("Fizz")
        elif i % 5 == 0:
            print("Buzz")
        else:
            print(i)
# 1から100までのFizzBuzzを表示
fizzbuzz(100)
```

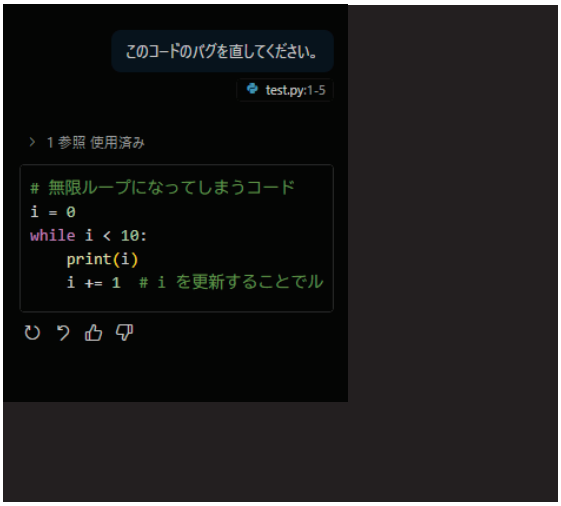
デモ2：チャット機能を用いたコード生成

「対話型ペアプログラマ」として活用

バグっているコード

```
# 無限ループになってしまうコード
i = 0
while i < 10:
    print(i)
    # i の更新を忘れている！
```

チャット機能を用いて修正



51

デモ3：チャットモードの切り替え

シーンに応じた使い分けがカギ

	Ask	Edit	Agent
目的/用途	コードを変更せずに情報を得たい	コードの一部を修正・改善	プロジェクト全体にわたる変更
使い方	コードを選択して質問	コードの変更を具体的に指示	プロジェクト全体に関わる指示
適したシーン	設計 アルゴリズム選択	リファクタリング 小さな修正	新機能の追加 設計変更

52

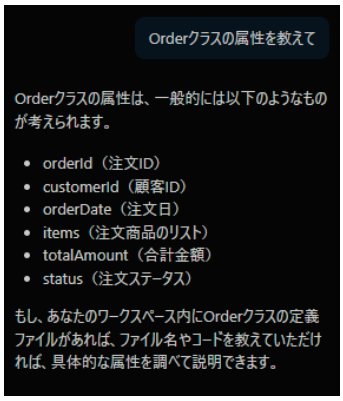
デモ4：コンテキストの追加方法

必要な情報を与えるほど精度が向上

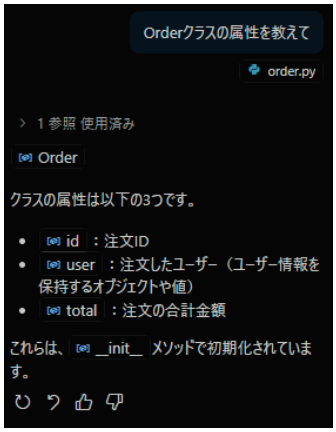
Orderクラスを作成

```python
# models/order.py
class Order:
    def __init__(self, id, user, total):
        self.id = id
        self.user = user
        self.total = total
```

コンテキストなし



コンテキストあり



53

チャット機能の切り替え方法

水色の四角の中からチャットモード切替やコンテキストの追加が可能



54

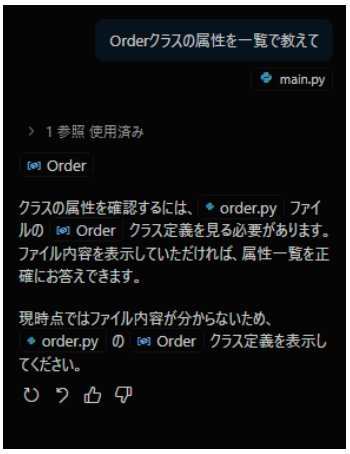
デモ5：メンション機能の活用

対象を指定することで無駄のない応答が得られる

Orderクラスを作成

```python
# models/order.py
class Order:
    def __init__(self, id, user, total):
        self.id = id
        self.user = user
        self.total = total
```

メンションなし



メンションあり



55

デモ6：コマンド機能の活用

定型作業を効率化

間違ったコードを修正したい

```python
# main.py
def find_max(nums):
    max_val = 0
    for n in nums:
        if n > max_val:
            max_val = n
    return max_val


print(find_max([-5, -2, -10]))
# 本当は -2 を返すべきだが 0 を返してしまう
```

各コマンドを使うことで
プロンプトを省略

/fix

/tests

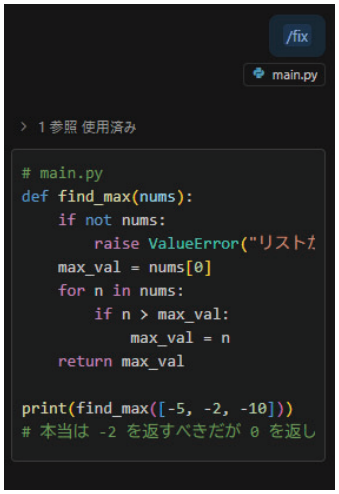
/explain

56

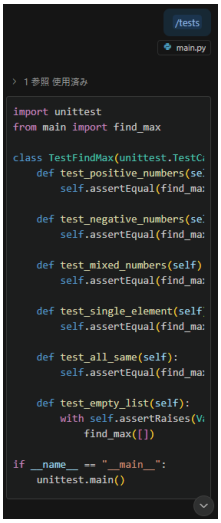
デモ6：各コマンド機能の活用例

プロンプトを省略可能

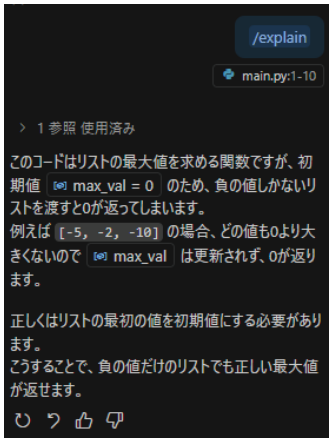
/fixでコード修正



/testsでテスト生成



/explainでバグの原因の説明



57

目次

1 はじめに

2 AIの概要と活用方法① チャットボット型生成AI

3 ワーク① チャットボット型生成AIでプログラミング

4 AIの概要と活用方法② コード補完ツール

5 **ワーク② コード補完ツールの機能体験**

6 ワーク③ AIの差を発表

7 まとめ

58

ワーク②：事前準備　GitHub Copilotの導入-1

GitHub CopilotをVSCで無料で使えるようにしましょう

1.GitHubアカウントを作成する。https://github.com/
2. VSCを開き、左下のアイコンをクリック。



3.「バックアップと同期の設定」>画面上部のサインインをクリック。
　ブラウザからサインインを行う。

59

ワーク②：事前準備　GitHub Copilotの導入-2

GitHub CopilotをVSCで無料で使えるようにしましょう

4.拡張機能の検索欄を開き、「GitHub」で検索をする。
5.「GitHub Copilot」の「インストール」をクリック。



6.導入完了！次からのワークを行って、補完機能が出るかテストしましょう。

60

ワーク②：コード補完ツールの機能体験

コード補完機能を使ってみる

課題内容

「FizzBuzzプログラム」を書いてみましょう。
・1から100までの整数を順番に出力する。
・3の倍数のときは「Fizz」、5の倍数のときは「Buzz」、
　両方のときは「FizzBuzz」と出力する。
・上記以外のときは数字をそのまま出力する。

課題手順

1.まずは自分でコードを書き始める。
2.補完機能の提案が出てきたら、
　内容を確認し、問題なければTabキーで提案を受け入れる。
3.それを繰り返して上記の課題を作成

61

ワーク②：チャット機能を用いたコード生成

チャット機能を用いてコードを改善する

課題内容

7の倍数で ‘Bazz’ を出力するように修正する。

課題手順

1.エディタのサイドバーまたはインラインで Copilotチャットを開く。
2. 「このコードを1行ずつ出力するようにしてください」と入力し、
　提案を受け取る。
3. 「7の倍数で ‘Bazz’ を出すように修正してください、
　複合の際は全て繋げてください。」と依頼する。
4. 「このFizzBuzzを関数化してください」と依頼する。
5.提案されたコードを適用(生成されたコードの右上に適用ボタンが出現)し、
　実行して正しく動作するか確認する。

62

ワーク②：チャットモード切替

チャットモード切替を行い、機能の違いを体感する

ASK

ワーク①②で使ったFizzbuzzを開いた状態で、
チャットのモードを「Ask」に切り替えチャット欄に質問をする。
質問例：このコードの計算量はどの程度ですか？

Edit

チャットのモードを「Edit」に切り替え修正を依頼する。
依頼例：このFizzBuzzを、配列に格納して返す関数 fizzbuzz(n) に書き換えてください。

Agent

チャットのモードを「Agent」に切り替え、複数の依頼をまとめて依頼する。
依頼例：このFizzBuzzをリファクタリングしてください。
- テストコードを追加
- コメントを日本語で入れる
- 複数ファイル構成に分割する（mainとlibに分ける）

63

ワーク②：コンテキストの追加方法1

コンテキスト機能を用いた際の出力結果の差を理解する

1.以下の内容のファイル「rules.txt」を作成する。

アプリのルール：
- ユーザーはタスクを追加できる
- 完了済みに変更できる
- タスクは期限を持てる

64

ワーク②：コンテキストの追加方法2

2.チャットモードを「Edit」にし、以下のプロンプトをチャットに入力する。

簡単なToDoアプリのサンプルコードを書いてください。
言語は {自分の使える言語} を指定します。
機能はタスクの追加だけで十分です。

3.チャットモードを「Ask」にし、以下の内容をチャットに入力する。

このアプリのルールは何ですか？

4.「コンテキストの追加」ボタンからrules.txtを選び追加する。

5.再度3.と同じ内容の質問をしてみる。

6.それぞれの質問への回答内容を比較し、
後の質問がrules.txtの内容を前提に回答していることを確認する。

65

ワーク②：コマンド機能を活用する

コマンド機能を使い、プロンプトを省略する

1.チャットモードを「Edit」にし、
下記のプロンプトを使い、Fizzbuzzプログラムを作成する。

FizzBuzzプログラムを作成してください。
仕様は以下の通りです：
- 3の倍数 → "Fizz"
- 5の倍数 → "Buzz"
- 3と5両方の倍数 → "FizzBuzz"
- それ以外の数は数字を文字列でそのまま出力
- 1から100までを順番に出力
- 条件分岐は排他的にせず、文字列を積み上げて判定してください
言語は {任意の言語に編集} でお願いします。

2.チャットモードを「Ask」に変更し、/doc,/tests,/fixとそれぞれ入力する。

3.それぞれの出力内容を確認する。

66

各問から判明したAIのメリットや強みの考察・発表

	手動と比較した際のコード補完ツールの強み・メリットは？
コード補完	
チャット	
チャットモード	
コンテキスト	
コマンド	
まとめ	

1 はじめに
2 AIの概要と活用方法①　チャットボット型生成AI
3 ワーク①　チャットボット型生成AIでプログラミング
4 AIの概要と活用方法②　コード補完ツール
5 ワーク②　コード補完ツールの機能体験
6 ワーク③　AIの差を発表
7 まとめ

ワーク③：AIの差を考察・発表

ワーク①②から判明したAIのメリットや強みの考察・発表

	各AIを比較した際の強み・メリットは？
チャットボット	
コード補完	

69

目次

1 はじめに

2 AIの概要と活用方法①　チャットボット型生成AI

3 ワーク①　チャットボット型生成AIでプログラミング

4 AIの概要と活用方法②　コード補完ツール

5 ワーク②　コード補完ツールの機能体験

6 ワーク③　AIの差を発表

7 まとめ

70

本日の振り返り

チャットボット型生成AI



・プロンプトの質がアウトプットを左右する

・役割、目的、条件の3つの鉄則で、より良い回答を得られる

コード補完ツール

・開発の強力なアシスタントになる

・適切なプロンプトやコマンドで、作業を劇的に効率化できる

71

（6）「AI 活用プロジェクト演習①PG 編」改訂・追加開発教材レベル 2

「AI活用プロジェクト演習①PG編」
PBL教材
レベル２

202X.XX.XX

コード補完ツール／AIエディタ

目次

1 はじめに

2 活用事例

3 GitHub Copilotの概要

4 ワーク①　GitHub Copilot活用

5 AIエディターの概要

6 ワーク②　AIエディター活用

7 ワーク③　実践開発課題

8 ワーク④　AIの差を考察・発表

9 まとめ

2

目次

1 はじめに

2 活用事例

3 GitHub Copilotの概要

4 ワーク① GitHub Copilot活用

5 AIエディターの概要

6 ワーク② AIエディター活用

7 ワーク③ 実践開発課題

8 ワーク④ AIの差を考察・発表

9 まとめ

本日のゴール

講義で到達する目標について

1 コード補完ツールを活用できるようになる

2 AIエディターの活用方法を理解する

3 実務で生成AIを活用するイメージを掴む

講義の進め方

本日の講義の流れを説明



Part 1
知識編

- コード補完ツール
- GitHub Copilot
- AIエディター



Part 2
実践編

- 多数の問題演習
- 実務想定のWebアプリ開発
- 各AIのメリット等を発表

5

AIの過去・現在・未来

AIは加速度的に進化を続けている



6

AIエディターへの進化の流れ

生成AIはこの数年で劇的な進化を遂げ、
「便利な道具」から「共同開発者」となった。

補完

文法やシンボルに基づく
コード補完

提案

コメントや文脈を理解
関数やロジックを生成

協働

プロジェクト全体を理解
コード修正・資料制作

目次

1 はじめに

2 活用事例

3 GitHub Copilotの概要

4 ワーク① GitHub Copilot活用

5 AIエディターの概要

6 ワーク② AIエディター活用

7 ワーク③ 実践開発課題

8 ワーク④ AIの差を考察・発表

9 まとめ

学習での活用

検索エンジンでは難しかった、現在の状況・質問者のレベルに合わせた説明を受けられる

質問内容	AIの回答
なぜこのプログラムはエラーが出るの？	原因と修正方法を説明
九九表を出力するプログラムを出力して欲しい	完成したサンプルコードを提示
この問題を理解するには何からどのように勉強すべき？	学習計画を設計

9

個人開発での活用

設計が完了した後の工程を短時間で完了

活用場面	活用結果
プロトタイプ作成	要件を書くだけで作成完了
デバッグ	エラーチェック漏れを防ぐ
エラー解消	エラー原因を即座に特定、改善案の提案まで確認

10

チーム開発での活用

メンバー間で足並みを揃えた状態で業務が進む

活用場面	活用結果
コードルール共有	AIでコード作成時、 自動的に策定したルールで作成
業務引継ぎ	コメントを記述することで 可読性が向上
コードレビュー	人間の確認前に生成AIで レビューすることで工数削減

11

企業利用での活用

既に企業でも生成AIが導入されており、
多数の生産性が向上した事例が発表されている。

LINEヤフー

約7,000名のエンジニアに 「GitHub Copilot for Business」導入	生産性が10～30％向上、 1日のコーディング時間1～2時間削減

富士通

システム開発業務で 「GitHub Copilot」を導入	作業時間が20％超削減 リファクタリングでは75％削減

参考①：LINEヤフーが「GitHub Copilot for Business」を導入、1人あたりのコーディング時間が削減
参考②：富士通が挑む「GitHub_Copilot」での開発効率化

12

目次

1 はじめに

2 活用事例

3 GitHub Copilotの概要

4 ワーク① GitHub Copilot活用

5 AIエディターの概要

6 ワーク② AIエディター活用

7 ワーク③ 実践開発課題

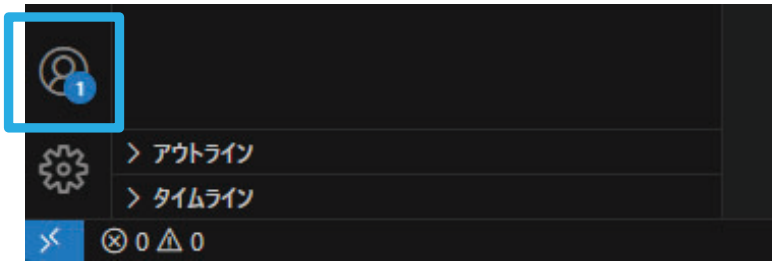
8 ワーク④ AIの差を考察・発表

9 まとめ

GitHub Copilotの導入-1

まずはGitHub CopilotをVSCで無料で使えるようする

1.GitHubアカウントを作成する。https://github.com/
2. VSCを開き、左下のアイコンをクリック。



3.「バックアップと同期の設定」>画面上部のサインインをクリック。
ブラウザからサインインを行う。

GitHub Copilotの導入-2

まずはGitHub CopilotをVSCで無料で使えるようにしましょう

4.拡張機能の検索欄を開き、「Github」で検索をする。
5.「GitHub Copilot」の「インストール」をクリック。



6.導入完了！次からのワークを行って、補完機能が出るかテストしましょう。

15

GitHub Copilotとは何か

GitHub CopilotはAIを用いたコード補完ツールの王者

概要

・開発元：GitHub（Microsoft傘下）
・OpenAIモデルを活用

特徴

・コード補完を超えて関数や処理全体を提案する
・世界で最も利用されているAIコード補完ツール

16

基本的なコード補完

「思考を補助する相棒」として活用

途中まで書いたコード

```python
# FizzBuzzを実装
def fizzbuzz(n):
    for i in range(1, n+1):
        if i % 3 == 0 and i % 5 == 0:
            # ここで止めると…
```

補完機能を使ったコード

```python
# FizzBuzzを実装
def fizzbuzz(n):
    for i in range(1, n+1):
        if i % 3 == 0 and i % 5 == 0:
            print("FizzBuzz")
        elif i % 3 == 0:
            print("Fizz")
        elif i % 5 == 0:
            print("Buzz")
        else:
            print(i)
# 1から100までのFizzBuzzを表示
fizzbuzz(100)
```

17

チャット機能の切り替え方法

水色の四角の中から
チャットモード切替・モデルの変更・コンテキストの追加が可能



18

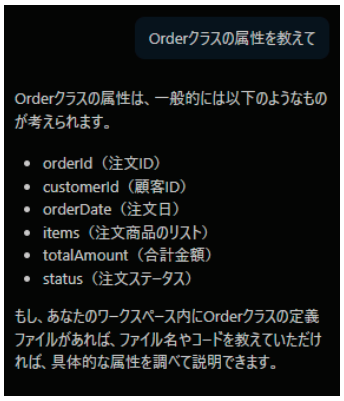
コンテキストの追加方法

必要な情報を与えるほど精度が向上

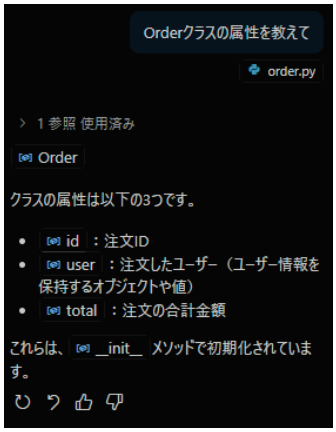
Orderクラスを作成

```python
# models/order.py
class Order:
    def __init__(self, id, user, total):
        self.id = id
        self.user = user
        self.total = total
```

コンテキストなし



コンテキストあり



19

チャットモードの切り替え

シーンに応じた使い分けがカギ

	Ask	Edit	Agent
目的/用途	コードを変更せずに情報を得たい	コードの一部を修正・改善	プロジェクト全体にわたる変更
使い方	コードを選択して質問	コードの変更を具体的に指示	プロジェクト全体に関わる指示
適したシーン	設計 アルゴリズム選択	リファクタリング 小さな修正	新機能の追加 設計変更

20

コマンドとは

チャットに加え、様々なコマンドを活用することで性能を引き出せる

チャットでの命令

チャット機能でプロンプトを書くと、 GitHub Copilotがコードを提案する

特殊コマンド

特殊なコマンドで、自然言語でプロンプトを記入しなくても、
プロジェクト全体や外部情報を対象にできる

21

コマンドやメンションについて

プロンプトの省略を行う方法を3つ紹介

1.ハッシュコマンド

2.メンション

3.スラッシュコマンド

22

ハッシュコマンドについて



例　：#fetch , #codebase

特徴：Copilot Chatに特別な指令を出し、
プロジェクト全体や外部情報を利用できる

自然言語と組み合わせることで、複雑な指示も可能

役割に合わせたコマンドを入力することで、プロンプトの省略が可能

コマンド	役割
#codebase	プロジェクト全体のコードを参照
#fetch	外部リソースを取得して利用
#explain	指定したコードやファイルの機能を解説
#tests	テストコードを生成

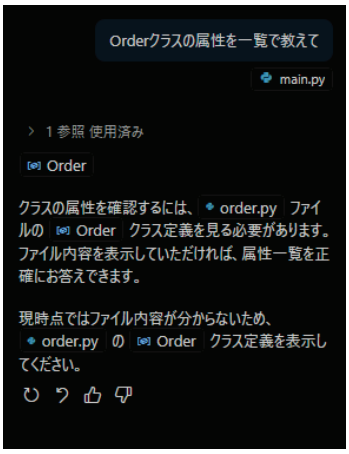
2.メンション機能の紹介

対象を指定することで無駄のない応答が得られる

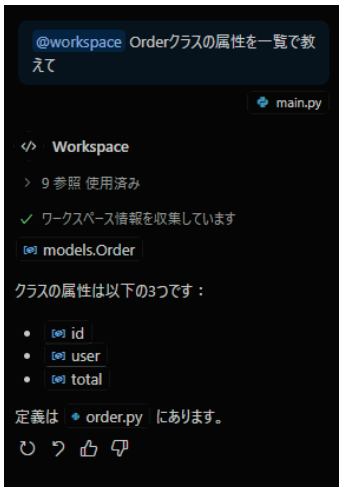
Orderクラスを作成

```python
# models/order.py
class Order:
    def __init__(self, id, user, total):
        self.id = id
        self.user = user
        self.total = total
```

メンションなし



メンションあり



2.メンションの具体例

シンプルなコンテキスト機能の設定を省略可能

コマンド例	役割
@workspace	プロジェクト全体を対象にする
@file	現在のファイルに限定して指示する
@symbol	特定の関数・クラスなどを対象にする

定型作業を効率化

間違ったコードを修正したい

```python
# main.py
def find_max(nums):
    max_val = 0
    for n in nums:
        if n > max_val:
            max_val = n
    return max_val

print(find_max([-5, -2, -10]))
# 本当は -2 を返すべきだが 0 を返してしまう
```

各コマンドを使うことで プロンプトを省略

/fix

/tests

/explain

スラッシュコマンドのみで簡単なプロンプトの作成が可能

コマンド例	役割
/explain	選択したコードを解説
/tests	テストコードを生成
/doc	ドキュメントやコメントを生成
/fix	バグを修正
/refactor	コードをリファクタリング

GitHub Copilot活用のメリット

現場の状況に幅広く対応することが可能

メリット

・プロンプトを省略する方法が豊富

・ドキュメントやテストケースの作成がスムーズに行える

・対応IDEが幅広い

・GitHubとの親和性が高い

29

GitHub Copilot活用のデメリット

扱い方を誤ると、むしろ効率を落としてしまう

デメリット

・社内規約や外部コードとの依存関係の反映が難しい

・必要以上にコードが作成され、保守が行いにくくなる

30

目次

1 はじめに

2 活用事例

3 GitHub Copilotの概要

4 ワーク① GitHub Copilot活用

5 AIエディターの概要

6 ワーク② AIエディター活用

7 ワーク③ 実践開発課題

8 ワーク④ AIの差を考察・発表

9 まとめ

ワーク①：コード補完ツールの強みを考察・発表

各問から判明したコード補完ツールの強み・メリットを考察

	手動と比較した際のコード補完型AIの強み・メリットは？
問1.開発(補完機能)	
問2.コード修正	
問3.言語変換	
問4.アルゴリズム問題	
問5.実務活用想定	
まとめ	

ワーク①の説明

本日のワークでは多くの問題に挑戦する。どんどん試して欲しい

問1.開発(補完機能)
- 1-1　データ抽出
- 1-2　指定文字列探し
- 1-3　Todoアプリの作成

問2.コード修正
- 2-1　無限ループ修正
- 2-2　算術バグ
- 2-3　リファクタリング体験

問3.言語変換
- 3-1　FizzBuzz
- 3-2　ハノイの塔

各問の中で1問は必ず取り組む

33

ワーク①の説明

本日のワークでは多くの問題に挑戦する。どんどん試して欲しい

問4.アルゴリズム問題
- 4-1　乗算問題
- 4-2　階段作り
- 4-3　排他的論理和

問5.実務活用想定
- 5-1　社内クイズ作成

どちらかを選択して取り組む

34

ワーク①：目次

問1.開発(補完機能)
- 1-1　データ抽出
- 1-2　指定文字列探し
- 1-3　Todoアプリの作成

問2.コード修正
- 2-1　無限ループ修正
- 2-2　算術バグ
- 2-3　リファクタリング体験

問3.言語変換
- 3-1　FizzBuzz
- 3-2　ハノイの塔

問4.アルゴリズム問題
- 4-1　乗算問題
- 4-2　階段作り
- 4-3　排他的論理和

問5.実務活用想定
- 5-1　社内クイズ作成

ワーク①：問1　開発課題

開発(コード生成)を体験

課題の取り組み方

次ページ以降のコードを以下の機能を使用して作成
・GitHub Copilotのコード補完機能
・Ask、Editモードのチャット機能

ワーク①：問1-1　開発課題　データ抽出

問題文

配列から20歳以上のユーザーを抽出する関数を作成する

例
入力：
```
[
  {name: "Alice", age: 25},
  {name: "Bob", age: 17},
  {name: "Charlie", age: 30}
]
```
出力：
```
[
  {name: "Alice", age: 25},
  {name: "Charlie", age: 30}
]
```


ワーク①：問1-2　開発課題　指定文字列探し　問題文

問題文

縦Hマス×横Wマスのマス目があり、各マスに1つずつ英小文字が書き込まれています。
上からi行目かつ左からj列目のマスを(i,j)で表します。
マス目に書き込まれている英小文字はH個の長さWの文字列S1,S2,…SHによって与えられ、
Siのj文字目が、(i,j)に書き込まれた英小文字を表します。
マス目の中に、s, n, u, k, e が
この順に(縦・横・ななめのいずれかの方向に)連続して並んでいる場所がただ１つ存在します。
そのような場所を見つけ、そのマスの位置を出力の形式に従って出力してください。
ただし、s, n, u, k, e が この順に(縦・横・ななめのいずれかの方向に) 連続して並んでいる場所
とは、5つのマスの組(A1,A2,A3,A4,A5)であって、次をすべてみたすものをさします。

・A1,A2,A3,A4,A5 に書き込まれた英小文字はそれぞれ s, n, u, k, e である。
・1≤i≤4について,AiとAi+1は頂点または辺を共有している。
・A1,A2,A3,A4,A5の中心はこの順に一直線上に等間隔で並んでいる。

出典:https://atcoder.jp/contests/abc302/tasks/abc302_b

ワーク①：問1-2　開発課題　指定文字列探し　制約・入力・出力

制約

・5≤H≤100
・5≤W≤100

・H,Wは整数
・Siは英小文字のみからなる長さWの文字列
・与えられるマス目の中に条件をみたす場所がただ1つ存在する

入力

入力は以下の形式で標準入力から与えられる。
H W
S1
S2
・
・
・
SH

出力

次の形式にしたがって、 5行出力せよ。
条件をみたす場所のうち s, n, u, k, e が書かれたマスがそれぞれ(R1,C1),(R2,C2)…,(R5,C5)であるとき、
i行目にはRiとCiをこの順に空白区切りで出力せよ。
すなわち、以下のように出力せよ。
R1 C1
R2 C2
R3 C3
R4 C4
R5 C5

ワーク①：問1-2　開発課題　指定文字列探し　入出力例1

入力例1

6 6
vgxgpu
amkxks
zhkbpp
hykink
esnuke
zplvfj

出力例1

5 2
5 3
5 4
5 5
5 6
この時、
(A1,A2,A3,A4,A5)=((5,2),(5,3),(5,4),(5,5),(5,6))
とすると、それぞれのマスに書き込まれた英小文字は s, n, u, k, e であり、
1 ≤i≤4について、AiとAi+1は辺を共有しており、
各マスの中心は一直線上に存在するため、
条件をみたしています。

ワーク①：問1-2　開発課題　指定文字列探し　入出力例2

入力例2

```
5 5
ezzzz
zkzzz
ezuzs
zzznz
zzzzs
```

出力例2

```
5 5
4 4
3 3
2 2
1 1
```

(A1,A2,A3,A4,A5)=((5,5),(4,4),(3,3),(2,2),(1,1))
が条件をみたしています。
例えば、
(A1,A2,A3,A4,A5)=((3,5),(4,4),(3,3),(2,2),(1,1))は、
1,2つめの条件をみたしていますが、
マスの中心が一直線上に存在しないため、
3つめの条件をみたしていません。

ワーク①：問1-2　開発課題　指定文字列探し　入出力例3

入力例3

```
10 10
kseeusenuk
usesenesnn
kskekeeses
nesnusnkkn
snenuuenke
kukknkeuss
neunnennue
sknuessuku
nksneekknk
neeeuknenk
```

出力例3

```
9 3
8 3
7 3
6 3
5 3
```

ワーク①：問1-3　Todoアプリの作成

問題文

軽量なToDoアプリを作成してください。

要件
・テキスト入力→「追加」ボタン or Enter で1件追加
・空文字や空白のみは追加不可
・各行にチェックボックス、削除ボタンを用意
・チェックで行に「完了」スタイル（取り消し線 or 薄色）を付与
・クリックで対象の行を削除

ワーク①：目次

問1.開発(補完機能)
1-1　データ抽出
1-2　指定文字列探し
1-3　Todoアプリの作成

問2.コード修正
2-1　無限ループ修正
2-2　算術バグ
2-3　リファクタリング体験

問3.言語変換
3-1　FizzBuzz
3-2　ハノイの塔

問4.アルゴリズム問題
4-1　乗算問題
4-2　階段作り
4-3　排他的論理和

問5.実務活用想定
5-1　社内クイズ作成

ワーク①：問2-1,2-2　コード修正

バグ修正機能を体験

課題の取り組み方

1.課題実行用のファイルを作成し、生成用プロンプトの{言語}を
自分の使用する言語にあったものに編集してからEditモードの
GitHub Copilotのチャットに貼り付けて、あなたの言語で“バグ入りコード”を出してもらう。

2.生成されたコードを実行し、バグを再現（ログ／挙動を確認）。

3.対象のファイルをコンテキストに追加した状態で、/fixとチャットに入力。

4.修正コードを実行して直ったことを確認。

言語指定の例

・{言語} → Python / JavaScript(Node) / C / Java / VBA(Excel)

45

ワーク①：問2-1　コード修正　無限ループ修正

生成用プロンプト
（これを編集してコード補完ツールに貼り付けましょう）

{言語}で下記の要件のコードを生成してください。
要件:
- 与えられた数値までの値を全て加算する関数を作ってください。
- 関数: spin_until(n)
  - バグ: ループ条件またはインクリメント欠落により実質無限ループになる初期実装にする。
  - ただし実行が停止しないと扱いづらいので、関数内だけで反復回数を監視し、一定回数を超えたら意図的に例外を投げる「簡易ウォッチドッグ」を入れる（try/catchは使わない）。これにより初回実行は必ず例外で停止する。
  - 正しい姿は 0..n の合計などを返す処理を想定。
- 他に安全な関数を1〜2個含める（例: clamp、format_result）。これらは初回から正常動作。
- 実行部（ファイル末尾）:
  - 最初にspin_until(100000) 等を呼び、初回は必ずウォッチドッグ例外で停止する構成にする。
  - その下に安全な関数の呼び出しを数行並べる（spin_untilを修正した後にだけ到達する）。
- 初期状態は try/catch を入れず、未捕捉例外で1回だけ落ちる。spin_until を1箇所修正すれば最後まで完走するように作る。
- 過度なコメントは不要。コードは読みやすく。

46

ワーク①：問2-2　コード修正　算術バグ

生成用プロンプト
（これを編集してコード補完ツールに貼り付けましょう）

{言語}で下記の要件のコードを生成してください。

要件:
- 与えられた引数の割り算を行う関数を作ってください。
- 関数: safe_div(a, b)
  - バグ: b==0 を考慮せず a/b を実行し、初回実行で未捕捉のゼロ除算例外が必ず発生する初期実装にする。
  - 正しい姿の例: b==0 のとき「0で割ることはできません」を返す等。※修正は1箇所で済む前提。
- 他に安全な関数を1〜2個含める（例: normalize: 合計0なら全0、clamp: min(max(x,lo),hi)）。これらは初回から正常。
- 実行部（ファイル末尾）:
  - 最初に print/safe_div(10, 0) を呼び、初回は必ず未捕捉例外で停止。
  - その下に安全関数の呼び出しを数行。safe_div を1箇所修正すれば最後まで完走する。
- 初期状態では try/catch を使わない。過度なコメントは不要。

47

ワーク①：問2-3　コード修正

リファクタリング機能を体験

課題の取り組み方

1.課題実行用のファイルを作成し、生成用プロンプトの{言語}を
自分の使用する言語にあったものに編集してからEditモードの
GitHub Copilotのチャットに貼り付けて、
あなたの言語で“動作は正常だが読みにくいコード”を出してもらう。

2.生成されたコードを確認・実行し、動作が「正しいが読みにくい／冗長」 なことを確認。

3.対象のファイルをコンテキストに追加した状態で、/optimizeとチャットに入力。
@editor 指示文　の形式で修正内容を追記することも可能。

4.改善されたコードを実行して、動作が変わっていないことを確認。

48

ワーク①：問2-3　コード修正　リファクタリング体験

生成用プロンプト
（これを編集してコード補完ツールに貼り付けましょう）

{言語}で「1900年～2024年までの各年に対して、うるう年ラベルを判定して出力」するプログラムを作成。（単一ファイル、外部ライブラリ禁止、即実行可能）
要件:
・「うるう年」・「通常」・「通常（世紀）」※100　例：1900→“通常“、2000→"通常（世紀）"、1996→"うるう年"、1999→"通常" ・1900～2024を1行ずつ出力。
・冗長な一時変数を4つ以上（同じ数値の段階的コピー、謎のフラグなど）。
・到達できないコードを1～2か所（return後、絶対に通らないelseなど）。
・到達できないコードは“通常“のように、一目見た際に到達できないことが不明にする
・同じ判定（%400, %100, %4やラベル結合）を少なくとも3か所で重複
・マジックナンバー（1900,2024,400,100,4）を直書きし、定数化しない。
・出力としての判定は最終的に正しいこと。
・コメント文を記載しない

ワーク①：目次

問1.開発(補完機能)
1-1　データ抽出
1-2　指定文字列探し
1-3　Todoアプリの作成

問2.コード修正
2-1　無限ループ修正
2-2　算術バグ
2-3　リファクタリング体験

問3.言語変換
3-1　FizzBuzz
3-2　ハノイの塔

問4.アルゴリズム問題
4-1　乗算問題
4-2　階段作り
4-3　排他的論理和

問5.実務活用想定
5-1　社内クイズ作成

ワーク①：問3　言語変換

言語変換の精度を体験

課題の取り組み方

1.課題実行用のファイルを作成し、生成用プロンプトの{言語}を
自分の使用する言語にあったものに編集してからEditモードの
GitHub Copilotのチャットに貼り付けて、サンプルコードを出してもらう。

2.チャットモードをAgentに変更し、
「@editor #selection を{ターゲット言語}関数として移植して」
の{ターゲット言語}を任意の言語名に編集した上で上記を入力する。

3. 実行結果を確認、環境がなければチャットモードをAskに変更し、
「実行結果を確認したい」と入力することで疑似的に実行結果が確認可能。
変換前の言語の実行結果と比較し、等しくない場合は/fixや/optimizeをチャットに入力。

4.実行結果に相違ないことを確認したら終了。

51

ワーク①：問3-1　言語変換　FizzBuzz

生成用プロンプト
（これを編集してコード補完ツールに貼り付けましょう）

{言語}で、単一ファイルのプログラムを作ってください。

[要件]
- FizzBuzzを作成する。
- 1..100 を順に出力。
- 3の倍数は "Fizz"、5の倍数は "Buzz"、両方の倍数は "FizzBuzz"、それ以外は数値。
- 1行に1つ出力（空行なし、末尾の余計な空白なし）。

52

ワーク①：問3-2　言語変換　ハノイの塔

生成用プロンプト
（これを編集してコード補完ツールに貼り付けましょう）

{言語}で、単一ファイルのプログラムを作ってください。

[要件]
- ハノイの塔を作成する。
- 関数: solve_hanoi(n) など任意の関数名で可。
- 最初、柱Aに円盤が全てあり、それを柱Cに移動させる。経由できる柱として柱Bを用意する。
- n=3のデモを実行し、各手を "move from A to C" のような形式で1行ずつ出力。
- 盤面表示やアニメは禁止。テキストで各手の出力のみ。

ワーク①：目次

問1.開発(補完機能)
1-1　データ抽出
1-2　指定文字列探し
1-3　Todoアプリの作成

問2.コード修正
2-1　無限ループ修正
2-2　算術バグ
2-3　リファクタリング体験

問3.言語変換
3-1　FizzBuzz
3-2　ハノイの塔

問4.アルゴリズム問題
4-1　乗算問題
4-2　階段作り
4-3　排他的論理和

問5.実務活用想定
5-1　社内クイズ作成

ワーク①：問4　アルゴリズム問題

難しい問題にコード補完機能を用いて挑戦

課題の取り組み方

1.次のスライドから出題される問題に、コード補完機能のみを用いて解答を作成する。

2.解答が正しいかは、出典として載せているURLのサイトにアクセスし、
ページ最下部の入力欄にコードを貼り付けることで確認可能。

3. 自分で取り組むことが難しいと判断した場合は、
問題に関する各内容をチャットモード「Edit」で入力し、解答を作成してもらう。
ページ最下部の入力欄にコードを貼り付けることで解答が正しいか確認をする。

4.その後、/explainを実行し、コードに関する説明を読み、理解を深める。

ワーク①：問4-1　アルゴリズム問題　乗算問題　問題文

問題文

N個の整数A1,…,ANが与えられます。
A1×…×ANを求めてください。
ただし、結果が10^{18}を超える場合は、代わりに -1 を出力してください。

出典:https://atcoder.jp/contests/abc169/tasks/abc169_b

ワーク①：問4-1　アルゴリズム問題　乗算問題　制約・入力・出力

制約

・$2 \leq N \leq 10^5$
・$0 \leq Ai \leq 10^{18}$
・入力は全て整数である。

入力

入力は以下の形式で標準入力から与えられる。
N
A1 … AN

出力

値A1×…×ANを整数として出力せよ。
ただし、この値が10^{18}を超える場合は、
代わりに -1 を出力せよ。

ワーク①：問4-1　アルゴリズム問題　乗算問題　入出力例1

入力例1

2
1000000000 1000000000

出力例1

1000000000000000000

1000000000×1000000000
=1000000000000000000 です。

ワーク①：問4-1　アルゴリズム問題　乗算問題　入出力例2

入力例2

```
3
101 9901 999999000001
```

出力例2

```
-1
```

101×9901×999999000001
=1000000000000000001 ですが、
これは10^{18}を超えるので、代わりに -1 を出力します。

59

ワーク①：問4-1　アルゴリズム問題　乗算問題　入出力例3

入力例3

```
31
4 1 5 9 2 6 5 3 5 8 9 7 9 3 2 3 8 4 6 2 6 4 3 3 8 3 2 7 9 5 0
```

出力例3

```
0
```

60

ワーク①：問4-2　アルゴリズム問題　階段作り　問題文

問題文

左右一列にN個のマスが並んでおり、左からi番目のマスの高さはHiです。
あなたは各マスについて1度ずつ次のいずれかの操作を行います。
・マスの高さを1低くする。
・何もしない。
操作をうまく行うことでマスの高さを左から右に向かって単調非減少にできるか求めてください。

出典:https://atcoder.jp/contests/abc136/tasks/abc136_c

ワーク①：問4-2　アルゴリズム問題　階段作り　制約・入力・出力

制約

・入力は全て整数である。
・$1 \leq N \leq 10^5$
・$1 \leq Hi \leq 10^9$

入力

入力は以下の形式で標準入力から与えられる。
N
H1 H2 … HN

出力

マスの高さを左から右に向かって
単調非減少にできるなら Yes、
そうでないなら No を出力せよ。

ワーク①：問4-2　アルゴリズム問題　階段作り　入出力例1

入力例1

```
5
1 2 1 1 3
```

出力例1

```
Yes
```

左から2番目のマスのみ高さを1低くすることで
目的を達成できます。

ワーク①：問4-2　アルゴリズム問題　階段作り　入出力例2

入力例2

```
4
1 3 2 1
```

出力例2

```
No
```

入力例3

```
5
1 2 3 4 5
```

出力例3

```
Yes
```

入力例4

```
1
1000000000
```

出力例4

```
Yes
```

ワーク①：問4-3　アルゴリズム問題　排他的論理和　問題文

問題文

6種類の文字、(,), [,], <, > からなる文字列 が与えられます。
ここで、文字列Tは、以下の条件を満たすときカラフル括弧列と呼ばれます。

以下の操作を何回か（ 0回でも良い）繰り返すことで、Tを空文字列にできる。
・Tの（連続する）部分文字列であって、
　(), [], <> のいずれかであるようなものが存在するとき、そのうちの1つを選んで削除する。
・削除された部分文字列がTの先頭または末尾であるとき、残りの文字列を新たにTとする。
・そうでないとき、削除された前後の文字列を1つに連結し、新たにTとする。

Sがカラフル括弧列であるか判定してください。

出典:https://atcoder.jp/contests/abc121/tasks/abc121_d

ワーク①：問4-3　アルゴリズム問題　排他的論理和　制約・入力・出力

制約

・S は長さ1以上2×10^5以下の文字列
・Sは (,), [,], <, > のみからなる。

入力

入力は以下の形式で標準入力から与えられる。
S

出力

S がカラフル括弧列ならば Yes を、
そうでないならば No を出力せよ。

ワーク①：問4-3　アルゴリズム問題　排他的論理和　入出力例1

入力例1	出力例1
([])<>()	Yes S=([])<>() に対して、次のように操作を繰り返すことで 空文字列にすることができます。 ・([])<>() の2文字目から3文字目までをとった部分文字列 [] を削除し、 前後を連結する。文字列は新たに ()<>() となる。 ・()<>() の1文字目から2文字目までをとった部分文字列 () を削除する。 文字列は新たに <>() となる。 ・<>() の1文字目から2文字目までをとった部分文字列 <> を削除する。 文字列は新たに () となる。 ・() の1文字目から2文字目までをとった部分文字列 () を削除する。 文字列は空文字列となる。 よって、S=([])<>() はカラフル括弧列であるため、Yes を出力します。

ワーク①：問4-3　アルゴリズム問題　排他的論理和　入出力例2

入力例2	出力例2
([<)]>	No S=([<)]> は (), [], <> を部分列として含まないため、 1回目の操作を行うことができず、 特にSはカラフル括弧列ではありません。 よって、No を出力します。

入力例3	出力例3
())	No

ワーク①：目次

問1.開発(補完機能)
1-1　データ抽出
1-2　指定文字列探し
1-3　Todoアプリの作成

問2.コード修正
2-1　無限ループ修正
2-2　算術バグ
2-3　リファクタリング体験

問3.言語変換
3-1　FizzBuzz
3-2　ハノイの塔

問4.アルゴリズム問題
4-1　乗算問題
4-2　階段作り
4-3　排他的論理和

問5.実務活用想定
5-1　社内クイズ作成

ワーク①：問5　実務活用想定　社内クイズ作成

実務に使えるような成果物をAIと共に作成

課題の取り組み方

1. GitHub Copilotのコード補完機能、チャット機能を活用し
   html,css,javascriptで、「4択問題を出題できるWebアプリ」を作成する。

2.問題内容は、受講者の皆さんが新入社員の方に、
「自社で勤める上で覚えてほしい」内容にする。問題数は指定なし。

3.アプリ上で解答終了後、リザルト画面の出力機能を追加する。
リザルト画面には正解数と正答率を出力。

4.全問正解した場合、リザルト画面に秘密のパスワードを出力。
それを担当者(上司など)に伝えることで、内容を理解したと伝えられる形式にする。

目次

1 はじめに

2 活用事例

3 GitHub Copilotの概要

4 ワーク① GitHub Copilot活用

5 AIエディターの概要

6 ワーク② AIエディター活用

7 ワーク③ 実践開発課題

8 ワーク④ AIの差を考察・発表

9 まとめ

AIエディターとは何か

生成AIだけでプロジェクト全体を作成してくれる

特徴

・コード補完を超えて開発全体を支援する

・生成、修正、解説、ドキュメント作成まで対応する

・人とAIが共同作業する開発環境である

代表的なツール

**AIエディターごとに得意としている内容が違うので、
使い分けることで出来る範囲が広がる**

Cursor	VSCodeベース、チャットしながら開発可能
Replit Ghostwriter	ブラウザ上で開発からデプロイまで支援
JetBrains AIアシスタント	IntelliJやPyCharmなどに統合

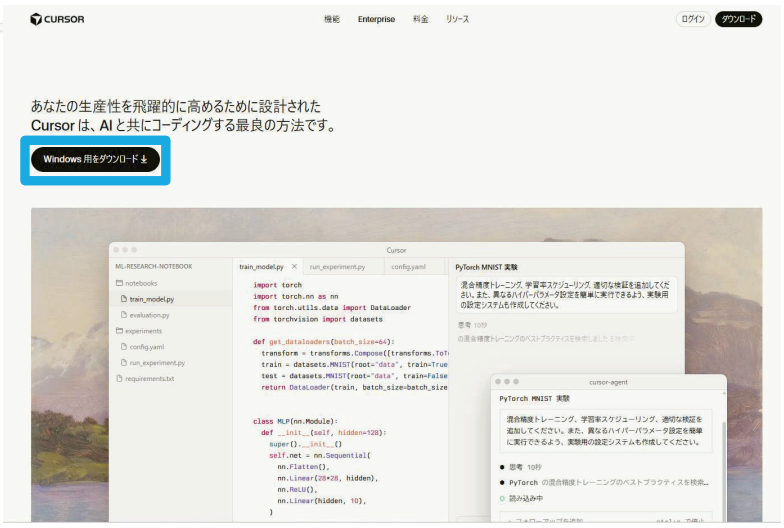
73

事前準備　Cursorの導入方法-1

Cursorを検索

まずCursorと検索して、インストールします。



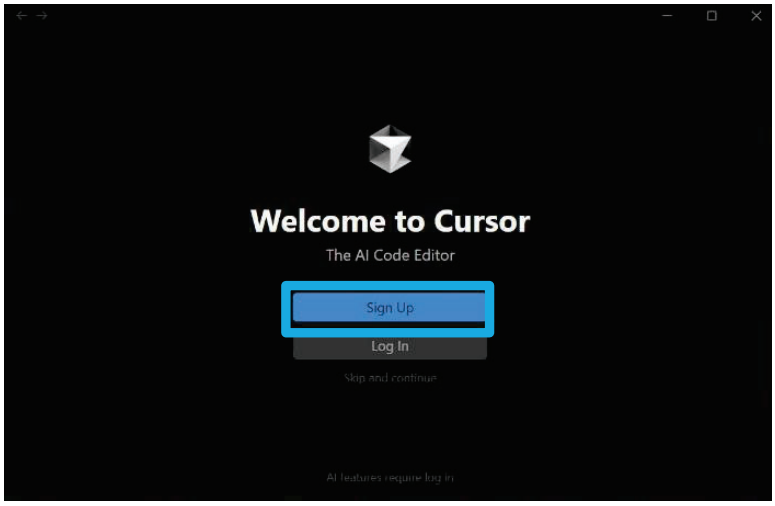


74

事前準備　Cursorの導入方法-2

CursorのSignUp

CursorSetup-x64-x.x.xx.exeをインストール後実行する。
全てそのままの設定で次へ→インストール完了まで行いSignUpを行う。



75

事前準備　Cursorの導入方法-3

アカウントを作成

ブラウザでサインイン画面が開かれるのでSign upを選択。
アカウント作成後、ログインを行う。

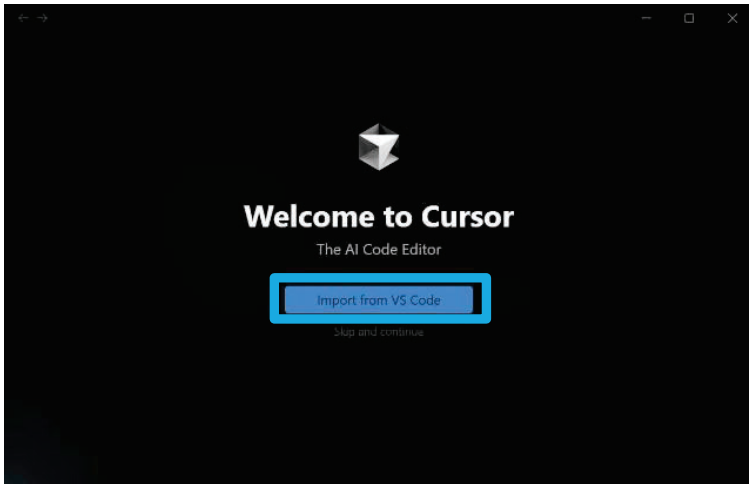


76

事前準備　Cursorの導入方法-4

VSCodeの設定を引継ぎ

VSCodeをインストールしている場合
「import from VS Code」を押下することで、VSCodeの設定を引継可能。

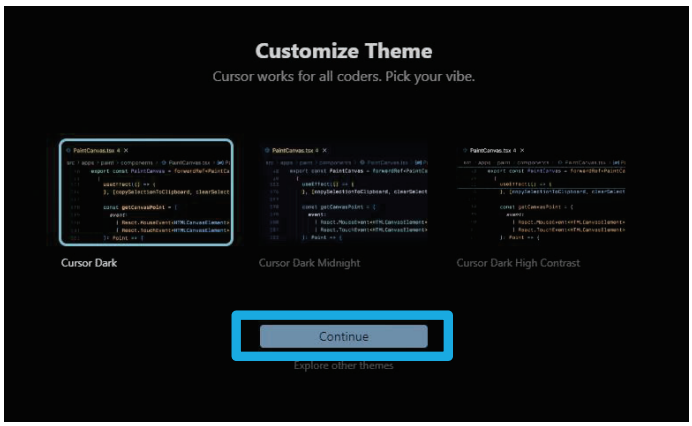


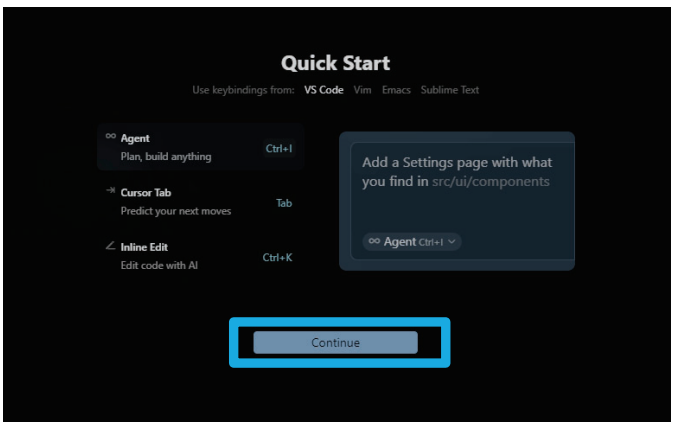
77

事前準備　Cursorの導入方法-5

Cursor初期設定

設定変更せずContinueを選択。

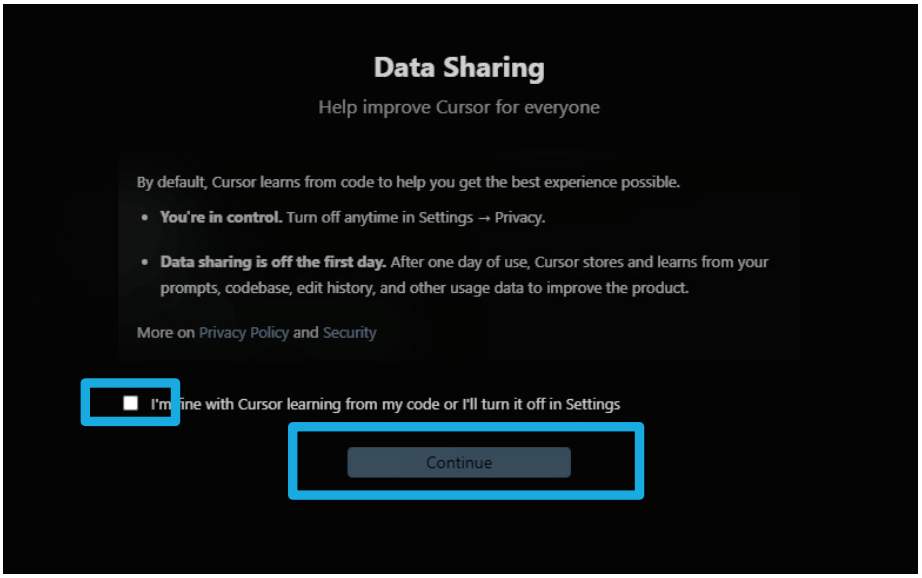




78

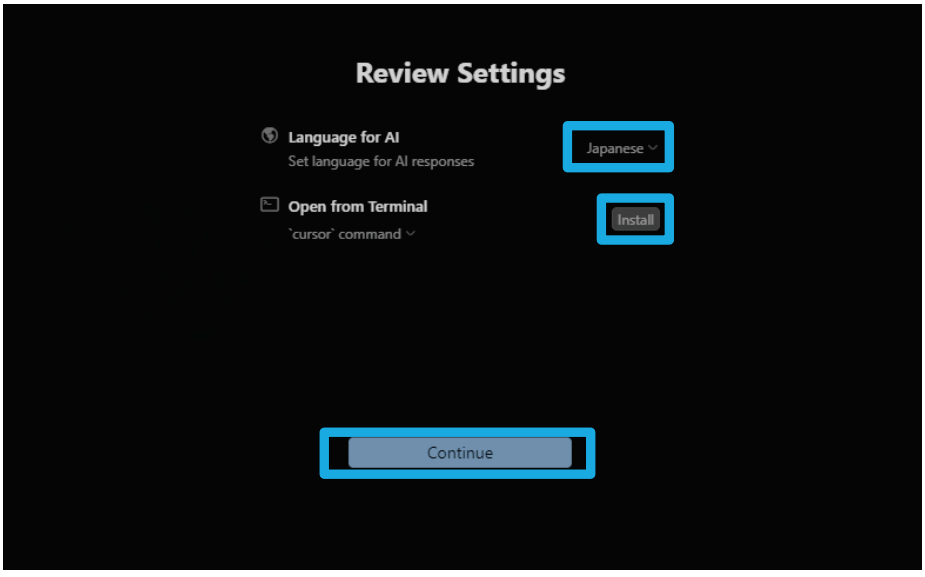
Cursor初期設定

チェックを入れてContinueを選択。



言語設定

LanguageをJapaneseに変更、Installをクリックした後Continueを選択。

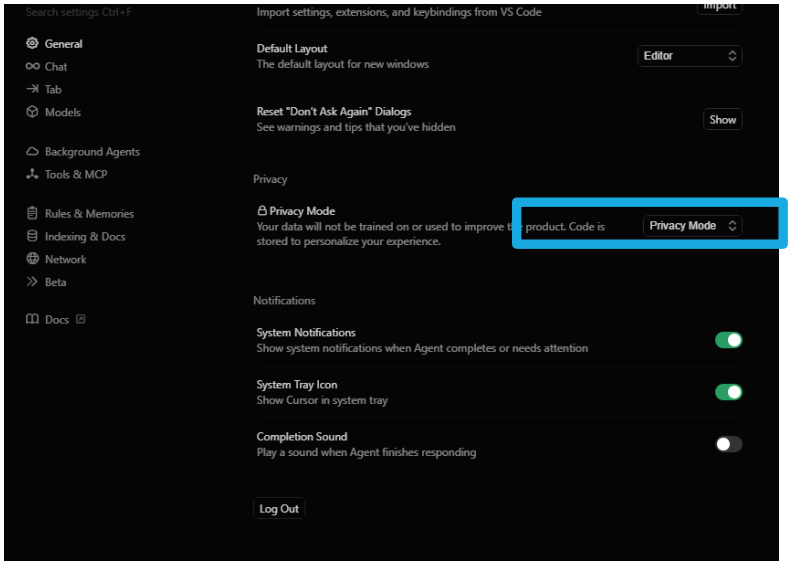


事前準備　Cursorの導入方法-8

学習設定の変更

Cursor起動後に右上の歯車アイコンをクリック。
Generalの中のPrivacyの設定を
Privacy Modeに変更。

これを行わずに利用すると、
入力した内容が
Cursorの学習に使われる可能性が
あるため注意が必要。



81

事前準備　Cursorの導入方法-9

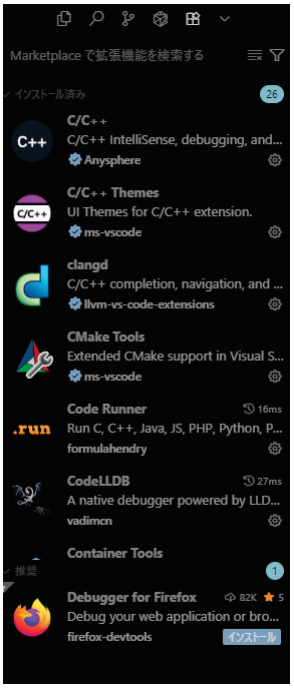
拡張機能の実装確認

1.画面右上の一番左のアイコン
「プライマリサイドバーを切り替える」をクリック

2.サイドバーを表示後、拡張機能の追加画面を開き、
自分がVSCで追加している拡張機能を何か検索。

3.既にインストールされていることを確認。

実装完了！お疲れ様でした！

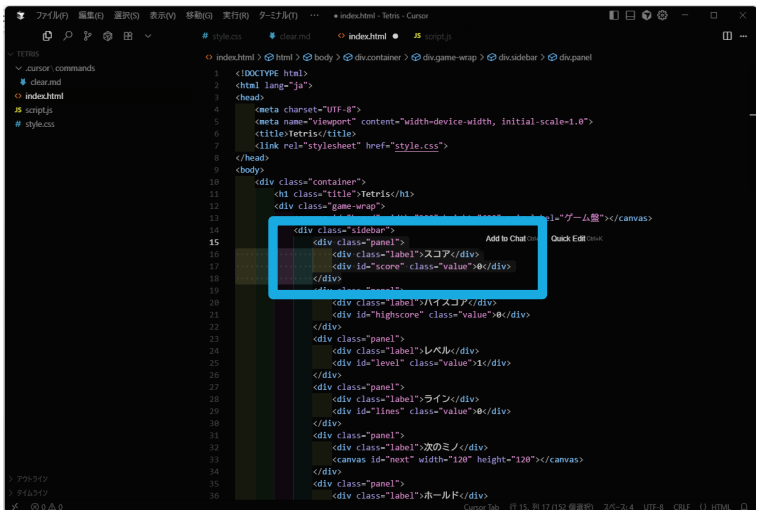


82

Cursorの使用方法：新規チャットの作成

範囲選択した上でチャットを開くことが可能

Ctrl＋L　→　新規チャットを開く
範囲を選択して新規チャットを開くことで
「この部分では何をしている？」のように質問できる

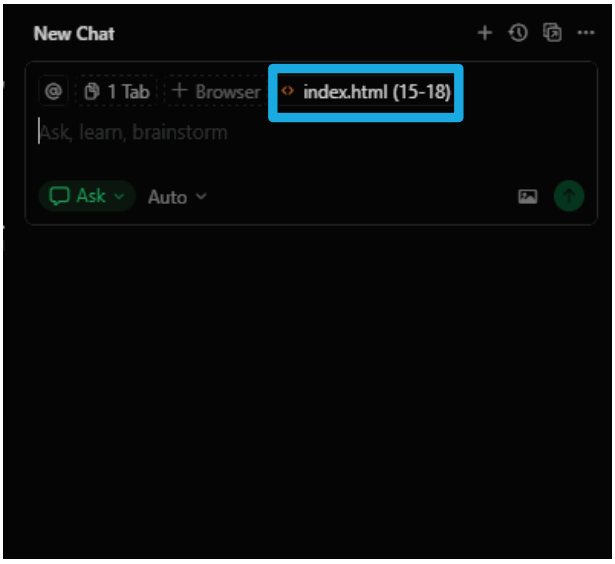


83

Cursorの使用方法：コンテキスト機能

前提条件を知らせた上でプロンプトを入力可能

選択部分が読み込まれていることが確認可能

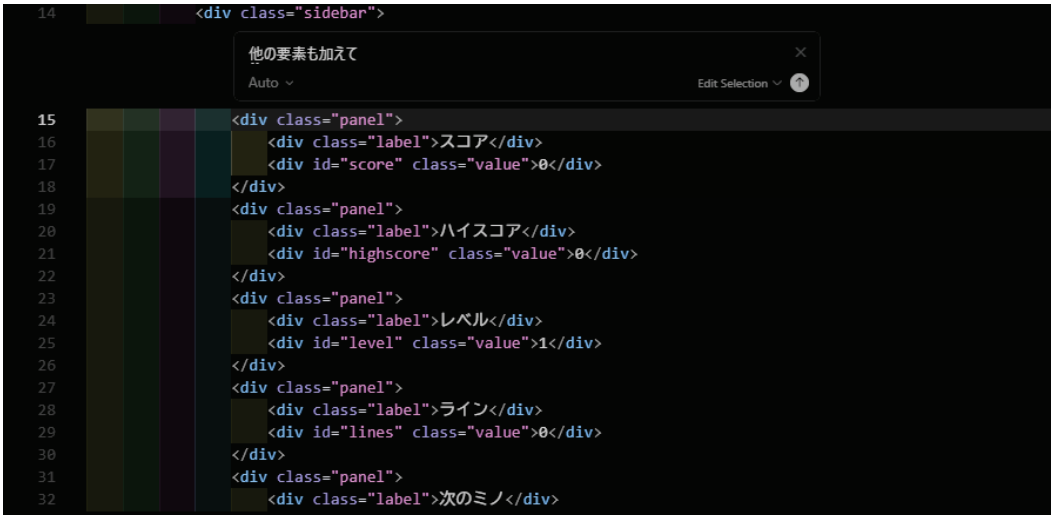


84

Cursorの使用方法：インライン操作

チャットを開かずにプロンプトの入力が可能

Ctrl＋K　→　インライン編集



85

Cursorの使用方法：コード提案の受入

変換前のコードと比較しながら1つずつ受け入れていく

プロンプトの反映されたコードが出力
Keepを押下することで提案コードを確定



86

Cursorの使用方法：モードの活用

対応してもらう内容に応じて使い分けることが重要

機能名	特徴
Composer	タスクごとに別の依頼を行う必要があるが、生成物の精度が高い 無料版では利用不可
Chat(Ask)	質問や小さい規模の依頼を安全に行うことが可能
Chat(Agent)	自動で生成されたコードを承認するだけで開発が可能 大きな依頼にすると生成物の精度が悪くなりやすい

87

Cursorの使用方法：メンション操作の活用

@の後に参照したいものを指定することで、その内容を踏まえた生成物が作成可能

コマンド例	出力結果
@<ファイル名>	指定したファイルをの内容を参照した上で出力
@<ファイル名>:<シンボル>	指定したシンボルにのみ関係する内容が出力
@<ドキュメント名>	指定したドキュメントの内容を考慮したコードが出力

88

AIエディターの強み

コード補完を超え、プロンプトから全てを作成可能

強み

・試作品を数分で作成可能

・エラーの内容を貼るだけで修正が可能

・個人でもチーム並みの開発スピードが実現可能

89

AIエディターの弱み

情報の正確性を保証するのが大変

弱み

・一度に大量にコードが作成されるため、
コード確認に工数がかかる

・同じ指示でも提案内容が変わることがある

・依存してしまい、利用者のスキルが落ちる

90

使用上の注意や心構えについて

AIの結果を盲目的に信じるのではなく、一度疑う姿勢が大切



提案はレビューして判断



コメントや指示を
具体的に書いて精度を高める



機密情報や重要コードが
含まれていないか確認



AIの提案を
「理解し取捨選択」する姿勢を持つ

91

目次

1 はじめに

2 活用事例

3 GitHub Copilotの概要

4 ワーク① GitHub Copilot活用

5 AIエディターの概要

6 ワーク② AIエディター活用

7 ワーク③ 実践開発課題

8 ワーク④ AIの差を考察・発表

9 まとめ

92

ワーク②：AIエディターの強みを考察・発表

各問から判明したAIエディターの強み・メリットを考察

	手動と比較した際のAIエディターの強み・メリットは？
テトリス 作成	
テトリス 機能拡張	
ワークフロー 作成・機能拡張	

93

ワーク②の説明

「AIと共に開発を行う」体験をしてほしい

問1.テトリス

1-1　テトリス作成

1-2　テトリス機能拡張

問2.ワークフロー作成

2-1　ログインフォーム作成

2-2　管理者アカウント作成

2-3　ワークフロー機能作成

2-4　オリジナル機能追加

94

ワーク②：問1 テトリス

問1.テトリス

1-1　テトリス作成

1-2　テトリス機能拡張

問2.ワークフロー作成

2-1　ログインフォーム作成

2-2　管理者アカウント作成

2-3　ワークフロー機能作成

2-4　ワークフロー機能追加

ワーク②：問1-1　テトリス作成

Cursorを使用したコード作成を体験

課題の取り組み方

1.自身の端末の適当な場所に、「Cursor work」という名前のフォルダを作成し
その中にhtmlファイルを作成。

2.左上の「ファイル」から先ほど作成したhtmlファイルを開く。

3.チャット機能を開き、左下のモードがAgentになっていることを確認。

4.「テトリスを作成したい」と入力し、作業が完了するまでしばらく待つ。

5.作業完了後、htmlファイルを開き、テトリスが実装されているか確認します。

「AIとの共同開発」を体験

課題の取り組み方

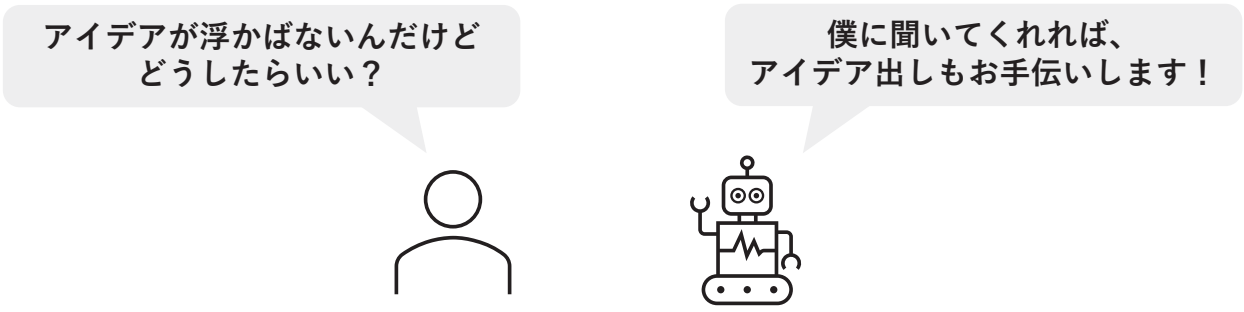
1.講師の作成したテトリスを参考にしつつ、先ほどのテトリスを動かして、
「実装した方がユーザの体験が向上する機能」について考える。

2.自分が先ほど作成したテトリスに、1.で考えた機能を実装。

3.実装した結果エラーが起きる等、想定した動作をしない場合も、
自らプロンプトを考え、修正を行う。

4.次のシートに、実装した機能とその結果をまとめる。

アイデア出しも生成AIに依頼可能



ワーク②：問1-2　テトリス機能拡張　結果記入用紙1

実装したい機能と結果1

実装したい機能	想定通りに実装できたか

99

ワーク②：問1-2　テトリス機能拡張　結果記入用紙2

実装したい機能と結果2

実装したい機能	想定通りに実装できたか

100

実装したい機能と結果3

実装したい機能	想定通りに実装できたか

問1.テトリス
1-1　テトリス作成
1-2　テトリス機能拡張

問2.ワークフロー作成
2-1　ログインフォーム作成
2-2　管理者アカウント作成
2-3　ワークフロー機能作成
2-4　ワークフロー機能追加

ワーク②：問2　ワークフロー作成

実務想定課題で生成AIの性能を体験

課題の取り組み方

1. html,css,javascriptを使い、ローカルサーバーでログイン情報を管理し、
   ログインするとプロフィール編集ができるようなWebアプリを作成。

2. アカウント作成・管理を行える管理者権限を
   アカウントに付与できる機能を実装。

3. 一般アカウントからワークフローの申請を行い、
   管理者アカウントから承認を行うことができる形式のワークフロー機能を実装。

4.ワークフローの実装後、機能の追加を自由に行う。

ワーク②：問2-1　ログインフォーム作成

問1.テトリス
1-1　テトリス作成
1-2　テトリス機能拡張

問2.ワークフロー作成
2-1　ログインフォーム作成
2-2　管理者アカウント作成
2-3　ワークフロー機能作成
2-4　ワークフロー機能追加

ワーク②：問2-1　ログインフォーム作成：ログイン画面



105

ワーク②：問2-1　ログインフォーム作成：プロフィール編集



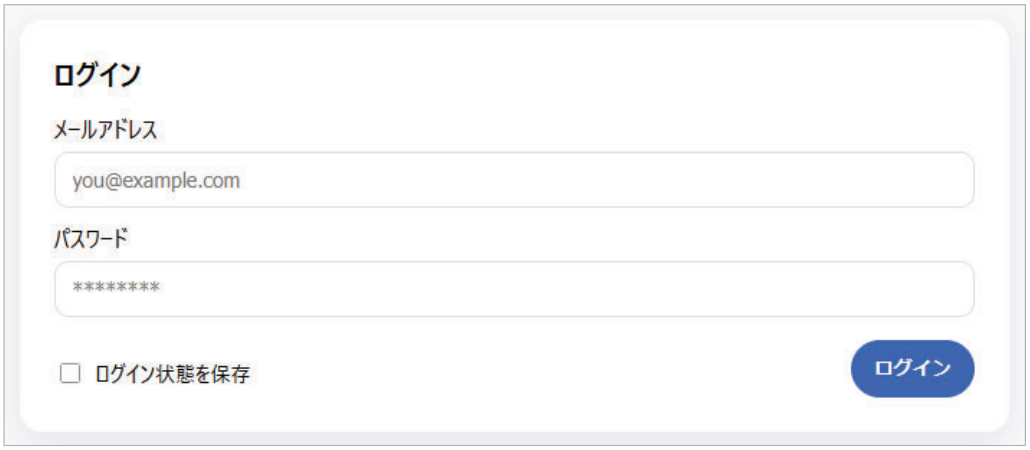
106

ワーク②：問2-2　管理者アカウント作成

問1.テトリス
1-1　テトリス作成
1-2　テトリス機能拡張

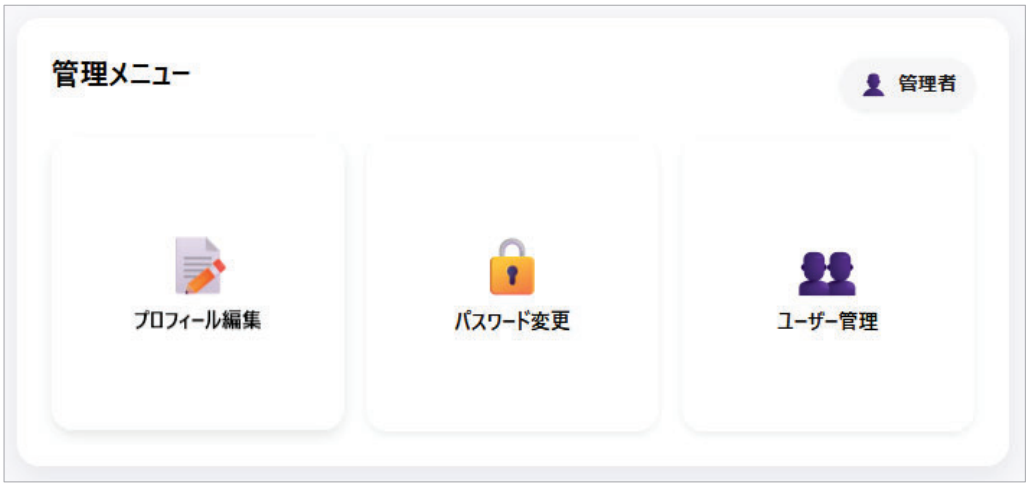
問2.ワークフロー作成
2-1　ログインフォーム作成
2-2　管理者アカウント作成
2-3　ワークフロー機能作成
2-4　ワークフロー機能追加

ワーク②：問2-2　管理者アカウント作成：ログイン画面



ワーク②：問2-2　管理者アカウント作成：メニュー画面





109

ワーク②：問2-2　管理者アカウント作成：プロフィール編集



110

ワーク②：問2-2　管理者アカウント作成：パスワード編集



111

ワーク②：問2-2　管理者アカウント作成：ユーザー管理画面



112

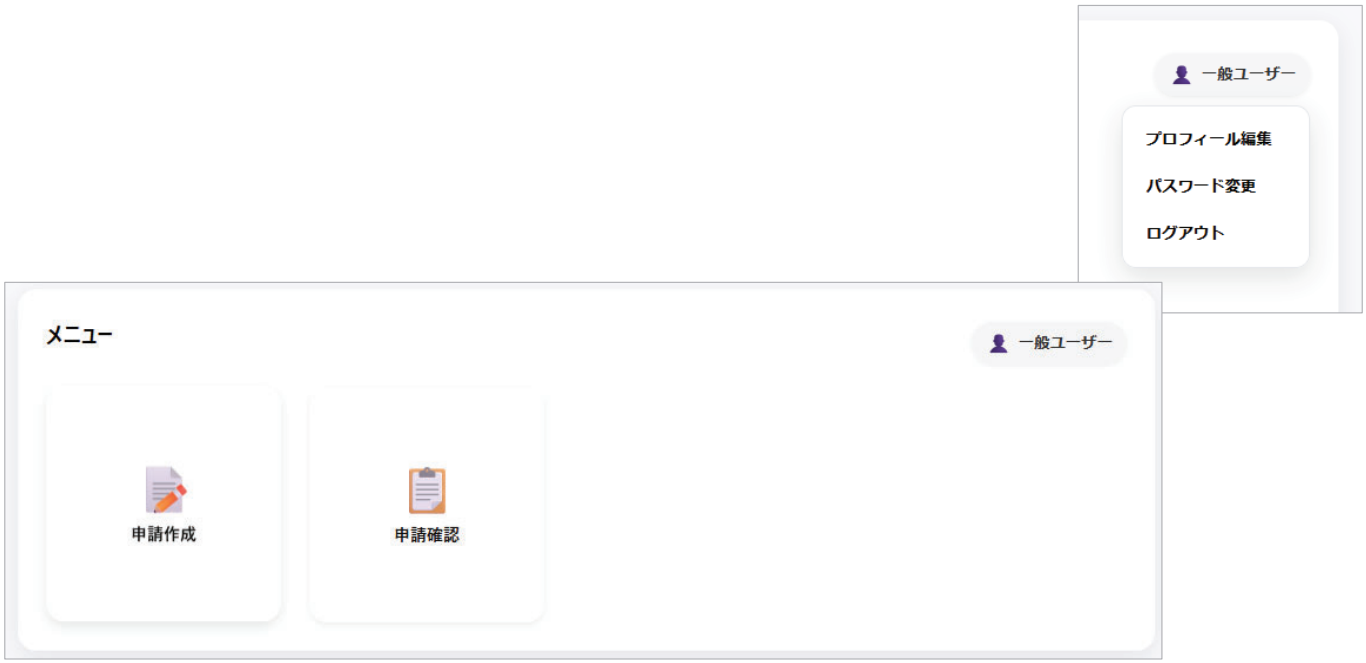
ワーク②：問2-3　管理者アカウント作成

問1.テトリス
1-1　テトリス作成
1-2　テトリス機能拡張

問2.ワークフロー作成
2-1　ログインフォーム作成
2-2　管理者アカウント作成
2-3　ワークフロー機能作成
2-4　ワークフロー機能追加

113

ワーク②：問2-3　ワークフロー機能作成：メニュー画面



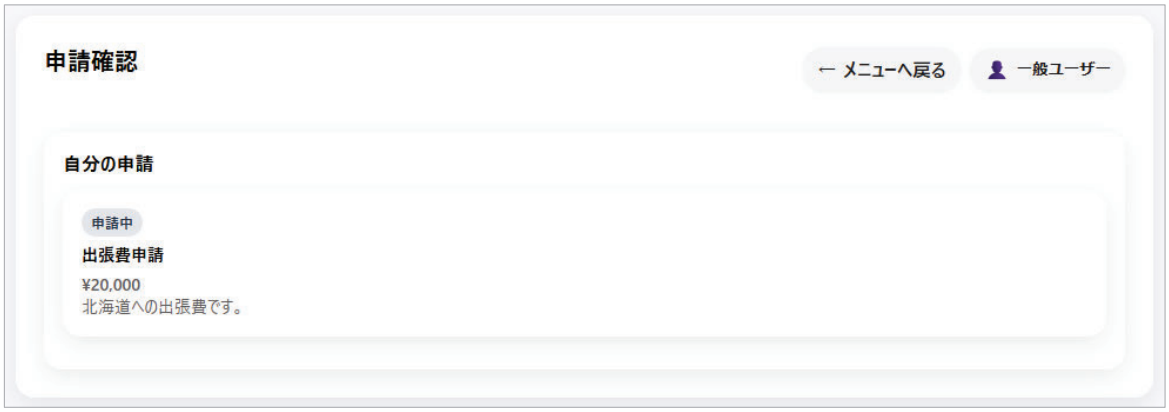
114

ワーク②：問2-3　ワークフロー機能作成：申請作成



115

ワーク②：問2-3　ワークフロー機能作成：申請確認（管理者承認前、申請者視点）



116

ワーク②：問2-3　ワークフロー機能作成：申請確認（管理者承認前、管理者視点）



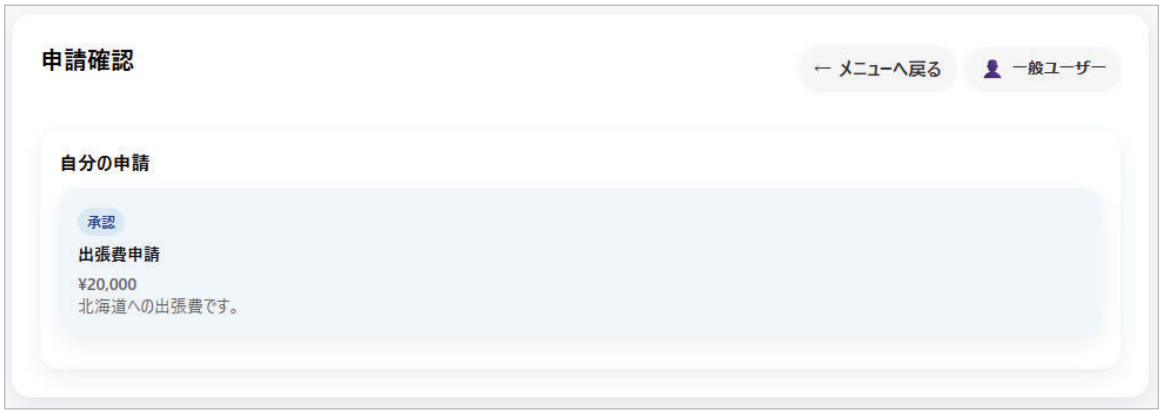
117

ワーク②：問2-3　ワークフロー機能作成：申請確認（管理者承認後、管理者視点）



118

ワーク②：問2-3　ワークフロー機能作成：申請確認（管理者承認後、一般ユーザー視点）



ワーク②：問2-4　ワークフロー機能追加

問1.テトリス
1-1　テトリス作成
1-2　テトリス機能拡張

問2.ワークフロー作成
2-1　ログインフォーム作成
2-2　管理者アカウント作成
2-3　ワークフロー機能作成
2-4　ワークフロー機能追加

ワーク②：問2-4　ワークフロー機能追加

オリジナルの機能を考えて自由に追加

例えば……

・ユーザーの種類に経理担当ユーザーを追加

・申請時に消耗品・旅費交通費など科目を選択

・管理者ユーザーをシステム管理者と承認担当者に分離

・より複雑なワークフローの経路選択

121

ワーク②：問2　サンプルサイト　利用情報

講師が作成したサンプルを紹介

URL　　：https://app.syscari.com/

ログイン情報
管理者　　：admin@example.com / adminpass
一般ユーザー：test@example.com / password123

122

ワーク②：問2-4　ワークフロー機能追加

実装したい機能と結果1

実装したい機能	想定通りに実装できたか

123

ワーク②：問2-4　ワークフロー機能追加

実装したい機能と結果2

実装したい機能	想定通りに実装できたか

124

実装したい機能と結果3

実装したい機能	想定通りに実装できたか

1 はじめに
2 活用事例
3 GitHub Copilotの概要
4 ワーク①　GitHub Copilot活用
5 AIエディターの概要
6 ワーク②　AIエディター活用
7 ワーク③　実践開発課題
8 ワーク④　AIの差を考察・発表
9 まとめ

ワーク③：実践開発課題

生成AIの機能を活用し、実践的な開発に取り組む

	課題の取り組み方	時間配分
手順1	実務で活用可能な成果物を個人で作成 ※cursor・ GitHub copilotを活用	60分
手順2	次のスライドにグループ内での共有内容を記入	5分
手順3	グループ内で1人3分程度で共有	15分

	題材例
例1	勤怠管理システム
例2	顧客データを記載したCSVファイル読込、特徴を抽出して提示

127

ワーク③：実践開発課題　発表内容の説明

発表内容の紹介

成果物の概要	作成した成果物に関する概要の記載
想定活用シーン	作成の際に想定した成果物が実務の中で活用できるシーン
AIの活用内容	成果物の作成の際どこにGitHub Copilot・Cursorを用いたか
効率化できた点	GitHub Copilot・Cursorを活用した結果、どのように開発を効率化できたか
リスク課題点	GitHub Copilot・Cursorを活用した際のリスクや課題点
学び	今回の課題を通して学びに繋がった点

128

ワーク③：実践開発課題　発表内容メモ

作成した成果物の発表内容を記入

成果物の概要	
想定活用 シーン	
AIの活用内容	
効率化 できた点	
リスク 課題点	
学び	

目次

1 はじめに

2 活用事例

3 GitHub Copilotの概要

4 ワーク①　GitHub Copilot活用

5 AIエディターの概要

6 ワーク②　AIエディター活用

7 ワーク③　実践開発課題

8 ワーク④　AIの差を考察・発表

9 まとめ

ワーク④：AIの差を考察・発表

生成AIから感じた強み、メリットを共有し、全体へ発表

	課題の取り組み方	時間配分
手順1	各生成AIの強みを**グループ内で**共有 共有　：各グループメンバー全員 内容　：ワーク①と②で記入した考察	15分
手順2	各生成AIの強みを**全体に向けて**発表 発表者：各グループの代表者 内容　：手順1で発表された考察のまとめ	各グループ 1～2分

131

ワーク④：AIの差を考察・発表

ワーク①②から判明した各AIの強み・メリットの考察・発表

	各AIを比較した際の強み・メリットは？
コード補完ツール	
AIエディター	

132

目次

1 はじめに

2 活用事例

3 GitHub Copilotの概要

4 ワーク① GitHub Copilot活用

5 AIエディターの概要

6 ワーク② AIエディター活用

7 ワーク③ 実践開発課題

8 ワーク④ AIの差を考察・発表

9 まとめ

コード補完ツールのまとめ①：コード補完ツールの強み

大規模な開発案件でも、チーム内で同じ条件でサポートしてくれる



事前にコードルールを共有した上で
効率的に開発が可能



メンションやコマンド機能の扱い方を
習得することで、プロンプトを省略可能

コード補完ツールのまとめ②：適した開発規模

大規模開発のエンジニアそれぞれの「パートナー」

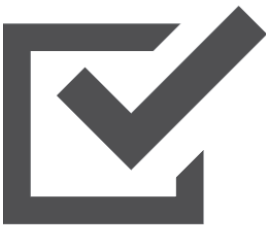


大規模なチームでの開発向け

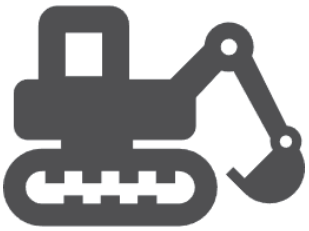
135

AIエディターのまとめ①：AIエディターの強み

開発を1から任せることが可能



プロンプトを入力するだけで、
開発業務を全て対応することが可能

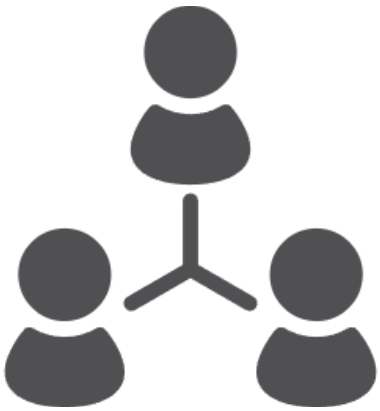


多数のファイルを一瞬で作成できるため、
プロジェクト全体の改修や設計変更が得意

136

AIエディターのまとめ②：適した開発規模

個人開発から少数精鋭までの「即戦力」



個人~少人数での開発向け

137

AI活用に必要なリテラシー

AIを扱う人間が、自らのスキルを高める意識を持とう



生成AIの提案をそのまま採用せず、
必ず人間がレビューする



レビューを正確に行うために、
・AIの提案の取捨選択を正しく行うスキル
・生成物の質が高いプロンプトを入力するスキル
両方を伸ばしていくことが必要

138

（7）「AI 活用プロジェクト演習①PG 編」改訂・追加開発教材レベル 3

「AI活用プロジェクト演習①PG編」
PBL教材
レベル３

202X.XX.XX

CLI型／エージェント（コンテナ）型

目次

1 はじめに

2 AIを取り巻く環境

3 活用事例

4 CLI型生成AIの概要

5 ワーク①　CLI型生成AI

6 コンテナ型生成AIの概要

7 ワーク②　コンテナ型生成AI

8 ワーク③　実践開発課題

9 ワーク④　自社の業務における活用

10 まとめ

目次

1 はじめに

2 AIを取り巻く環境

3 活用事例

4 CLI型生成AIの概要

5 ワーク① CLI型生成AI

6 コンテナ型生成AIの概要

7 ワーク② コンテナ型生成AI

8 ワーク③ 実践開発課題

9 ワーク④ 自社の業務における活用

10 まとめ

本日のゴール

本日の講義を通して最新のAIの状況を知り、
CLI型・コンテナ型生成AIを使って基本的な開発ができるようになる

チャットボット型AI	ChatGPT Gemini
コード補完型AI	Github Copilot
CLI型AI	Gemini CLI
コンテナ型AI	Codex Devin

変化が激しく、
複雑なAI関連情報の中から、
本当に必要な情報を解説

ワーク中心型講義

本日の講義では、多数のワークにより実践的な活用方法を学ぶ。

ワーク①　CLI型生成AI

1. 15パズル
2. プログラムの分割
3. タイマー機能追加
4. オリジナル機能追加

ワーク②　コンテナ型生成AI

2-1 神経衰弱
2-2 マルチモーダル

ワーク③　実践開発課題

3-1 ウォーターフォールモデル開発
3-2 アジャイル開発
3-3 学習内容の考察・発表

ワーク④　自社の仕事における活用

4-1 仕事における活用方法の考察・発表

本講義の注意点

本講義では設定方法にミスがある場合、
PCが破壊するリスクがあります。

必ず講師の話が終わってから、作業に着手してください。

目次

1 はじめに

2 AIを取り巻く環境

3 活用事例

4 CLI型生成AIの概要

5 ワーク① CLI型生成AI

6 コンテナ型生成AIの概要

7 ワーク② コンテナ型生成AI

8 ワーク③ 実践開発課題

9 ワーク④ 自社の仕事における活用

10 まとめ

AIは多方面に発展・進化している

画像生成・音声・スライド制作等においてもAIは進化
双方学ぶことで、高品質な資料制作やマニュアル制作等が可能に

画像生成AI



Nano Banana

音声生成AI



Gemini Speech Generation

スライド生成AI



Gamma

AI新法が日本でも施行

AIのイノベーションを促進しつつ、リスクに対応するため、
AI新法が令和7年9月1日に全面施行された

法律の必要性

日本のAI開発・活用は遅れている　　**多くの国民がAIに対して不安**

イノベーションを促進しつつ、リスクに対応するため、
既存の刑法や個別の業法に加え、新たな法律が必要。

参考文献：https://www.cao.go.jp/press/new_wave/20251003.html

目次

1 はじめに
2 AIを取り巻く環境
3 活用事例
4 CLI型生成AIの概要
5 ワーク①　CLI型生成AI
6 コンテナ型生成AIの概要
7 ワーク②　コンテナ型生成AI
8 ワーク③　実践開発課題
9 ワーク④　自社の業務における活用
10 まとめ

レガシーコードの翻訳家：AIによる既存システムの可視化

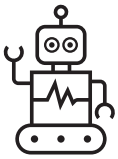
AIは資料を同じ品質で高速に作成可能
コンテキストを与えることで、既存システムを理解し解析



資料が無い



冗長なコード



解析・理解



整理された資料

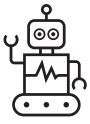


正規化されたコード

11

新人のマンツーマン教育：AIによる新人に寄り添った講義

新人教育を一部AIに任せることで、指導にかかる工数を削減可能



AIが教育可能な内容

- コードの修正
- 学習方法



上司が教育可能な内容

- 企業独自の手法
- 企業文化・風土

12

AI利用の鉄則：学習データ組込の回避策

AIに機密情報を入力する際は、契約・ポリシーの確認が最重要

AI利用のリスク	AIの学習を防ぐには
コンテキストや プロンプトが学習される	対策案① 入力データの保持期間
企業の大切な情報が 漏洩するリスク！	対策案② 学習利用の有無

13

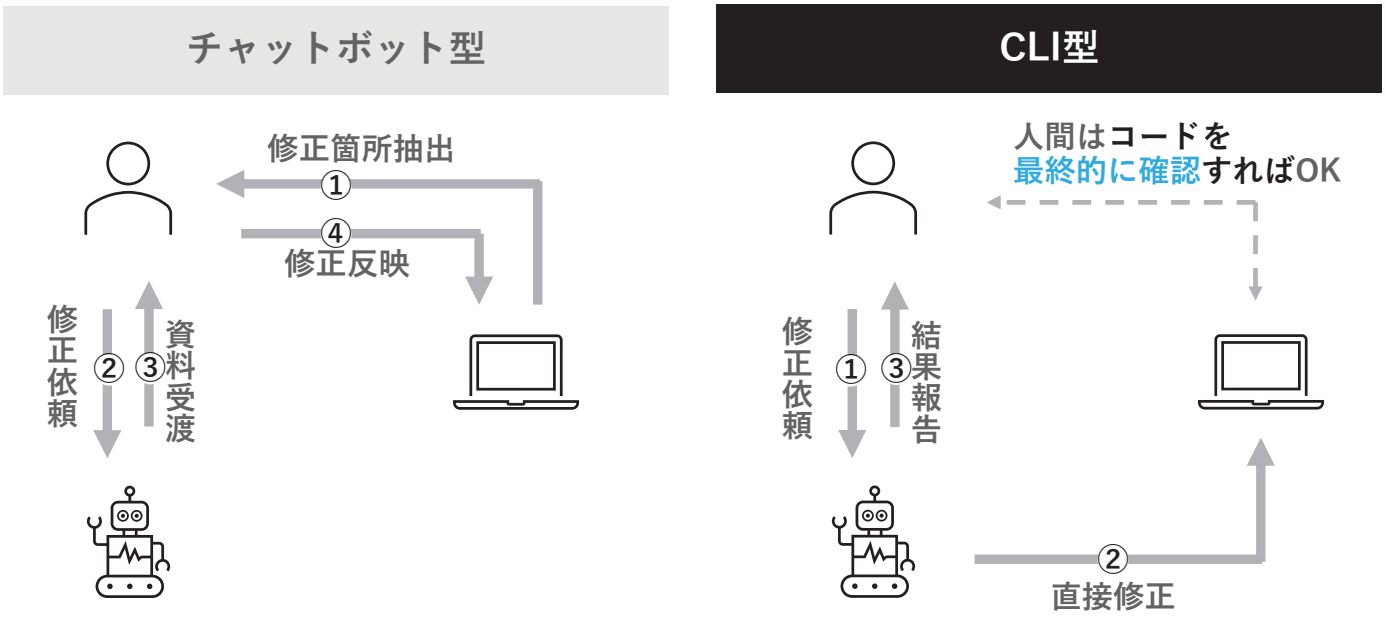
目次

1 はじめに
2 AIを取り巻く環境
3 活用事例
4 CLI型生成AIの概要
5 ワーク① CLI型生成AI
6 コンテナ型生成AIの概要
7 ワーク② コンテナ型生成AI
8 ワーク③ 実践開発課題
9 ワーク④ 自社の業務における活用
10 まとめ

14

CLI型の概要

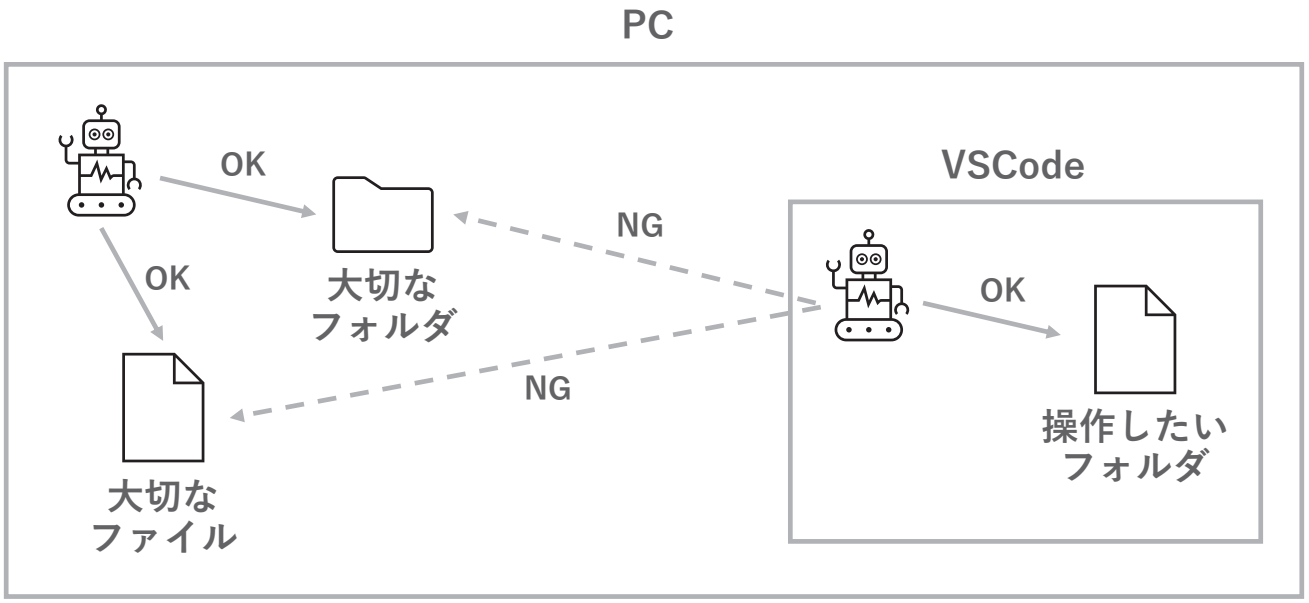
生成AIはコマンドラインから操作する時代に！
それにより、ローカルファイルの読み書き・コマンドの実行が可能



15

CLI型の注意点：PCの破壊リスクがある

power shell等からはAIがPC内の全てのファイルを操作可能
VSCodeから起動し、予期せぬファイル操作を制限

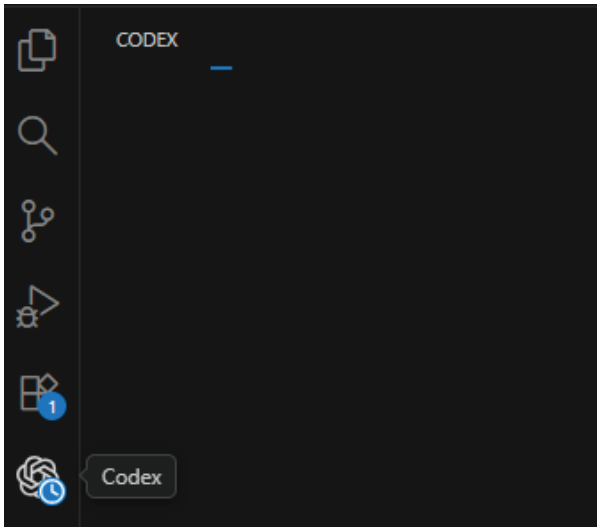


16

CLI型の注意点：実際に動かなくなってしまった例

本研修の資料を作成する中で、実際にAIに設定させて破壊してしまった実例

動かなくなったCodex



接続不可になったデモサイト

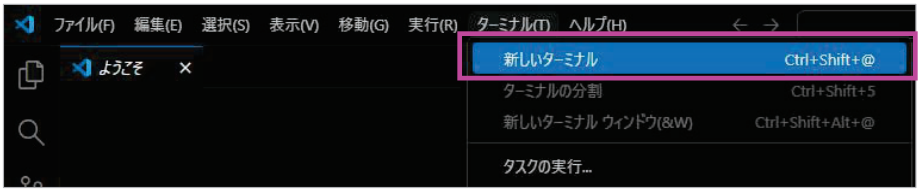


目次

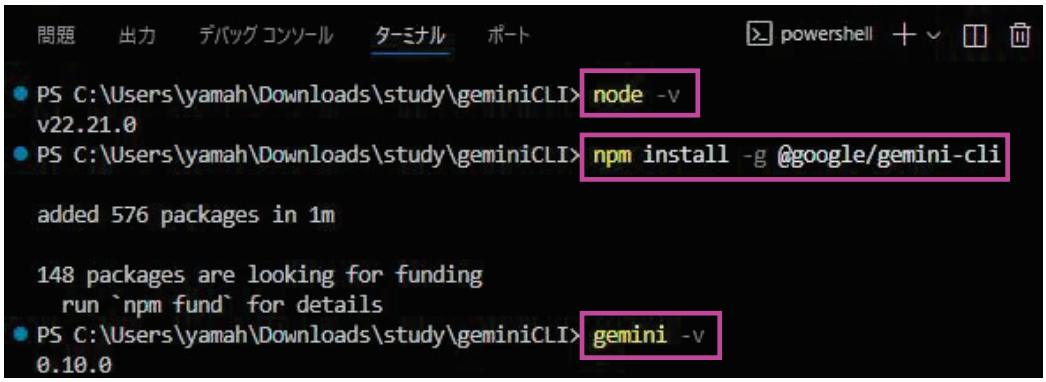
1 はじめに
2 AIを取り巻く環境
3 活用事例
4 CLI型生成AIの概要
5 ワーク① CLI型生成AI
6 コンテナ型生成AIの概要
7 ワーク② コンテナ型生成AI
8 ワーク③ 実践開発課題
9 ワーク④ 自社の業務における活用
10 まとめ

Gemini CLI導入手順①　前提条件（Node.js）の確認とGemini CLIのインストール

1　VSCodeのターミナルを起動



2　node.jsの確認・Gemini CLIのインストール



19

Gemini CLI導入手順：補足①　Node.jsがインストールされていない場合

Node.jsがインストールされていない場合は、インストール作業を行う

公式サイトURL

https://nodejs.org/ja/

インストール方法

全てNextで問題ありません



20

Gemini CLI導入手順：補足②　about_Execution_Policies エラーが出る場合

Node.jsがインストールされているが、npm –v 実行時に「about_Execution_Policies」エラーが出る場合、以下の手順を実施

権限の確認

Get-ExecutionPolicy -List

権限の許可

Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope LocalMachine

状況確認

npm -v

21

Gemini CLI導入手順②　Gemini CLIの起動

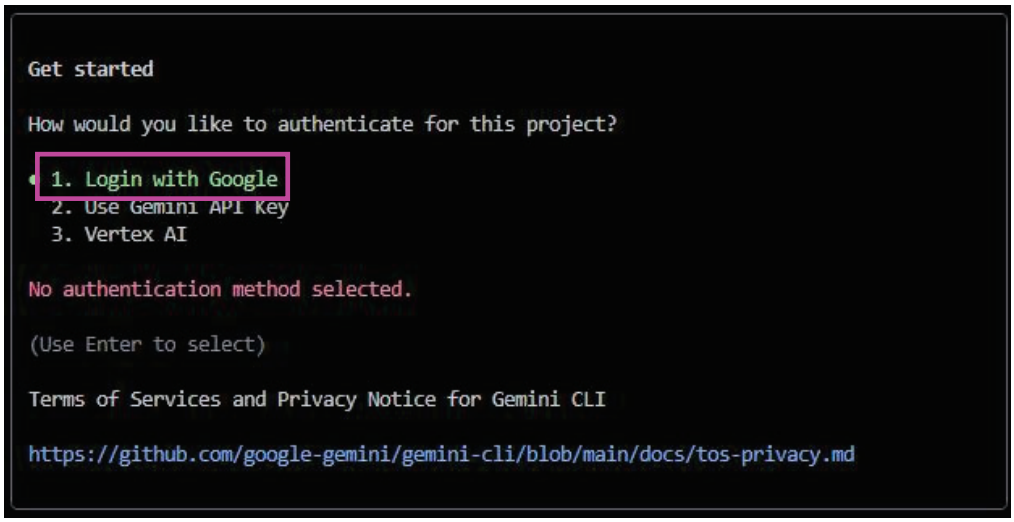
3　Gemini CLIの起動



22

Gemini CLI導入手順③　Googleアカウント認証

4　Googleのアカウント認証



Gemini CLI導入手順④　導入の際のエラー例の紹介

導入の際頻出するエラーとその対策。
エラーが出た場合のみ確認。

ケース1：スペースの入力漏れなどのタイピングミス
対応策　：エラーが出た際、まずは正確にコマンドが書けているか確認。

ケース2：PowerShellを管理者権限で開いて実行する指示が
　　エラー文にある場合。
対応策　：PowerShellを管理者権限で実行し、
　　エラーになったコマンドを実行。
　　無事導入完了したらVSCに戻ってから作業を続ける。

上記の2パターンの対策を行ってもエラーが出てしまう場合は、
周りのスタッフにお声掛けください。

ワーク①：ワークの説明

実際にアプリを作成することで、Gemini CLIの操作方法を学ぶ。
15パズルかミニブログのどちらかを選んで実施。

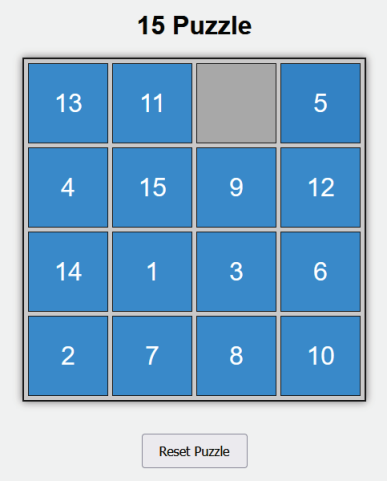
	15パズル	ミニブログ
1	15パズルの作成	ブログアプリの作成
2	ファイルの分離	
3	画面とタイマーの追加	ログイン画面の追加
4	独自要素の追加	

25

ワーク①：15パズルの作成

プロンプト

1つのhtmlファイルで、シンプルな15パズルを作成してください。



26

ワーク①：ブログアプリの作成

プロンプト

1つのhtmlファイルで、シンプルなブログアプリを作成してください。
投稿フォーム・一覧・削除ボタンを含む縦型レイアウトにしてください。

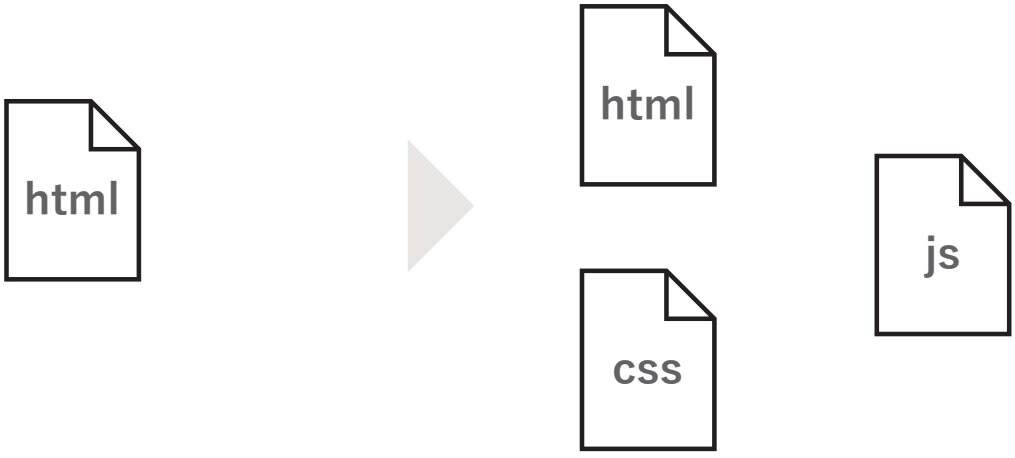


27

ワーク①：15パズル・ブログアプリのファイル分離

プロンプト

現在の15パズル/ブログアプリを
htmlとcssとjavascriptに分離してください。



28

ワーク①：15パズルへ機能の追加

プロンプト

ゲームの開始画面と終了画面を作成してください。
併せて、タイマー機能を追加してください。



29

ワーク①：ブログアプリへ機能の追加

プロンプト

ブログアプリにログイン画面を追加してください。



30

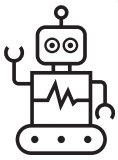
ワーク①：15パズル・ブログアプリへオリジナル機能の追加

ゲーム内に独自で考えた要素を追加してください

アイデアが浮かばないんだけど
どうしたらいい？



僕に聞いてくれれば、
アイデア出しもお手伝いします！



目次

1 はじめに

2 AIを取り巻く環境

3 活用事例

4 CLI型生成AIの概要

5 ワーク①　CLI型生成AI

6 コンテナ型生成AIの概要

7 ワーク②　コンテナ型生成AI

8 ワーク③　実践開発課題

9 ワーク④　自社の業務における活用

10 まとめ

コンテナ型AIの概要

CLI型で出来る作業と大きく変わらないが、使いやすさが向上

人間の修正の利便性

AIの開発内容を人間が修正する場合、IDEは便利
コード補完型AIと併用することも可能

IDEの拡張機能

IDEの豊富な拡張機能を併せて使用可能

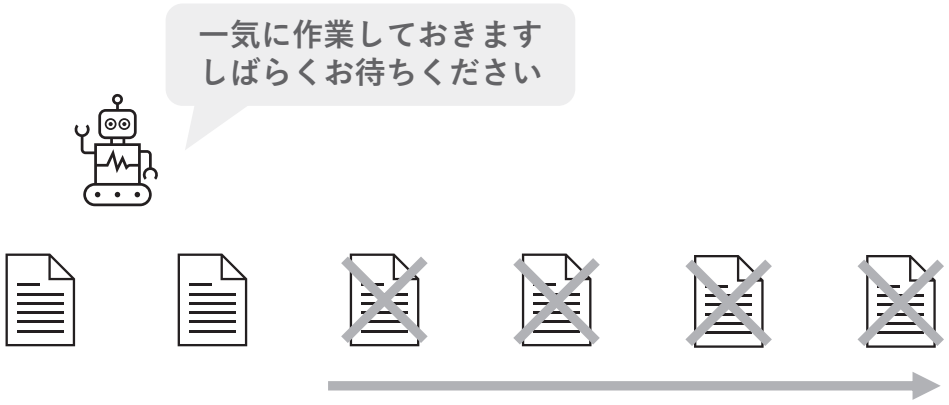
AIの作業確認

修正内容を分かりやすいUIによって、確認し易い

33

コンテナ型AIの注意点

これまで以上に複雑な処理を依頼したくなるが、一度に行う作業量が多すぎるとミスの影響が大きくなってしまう



後続の作業に影響するミスの場合、全てやり直し

無駄に時間がかかってしまう！

34

コンテナ型AIの種類

コンテナ型AIは様々なサービスが発表されている
ワークはどのサービスを使用しても問題ない

＼ 資料で使用 ／

Codex	Jules	Devin
高品質	無料 PCを破壊しない	コーディング以外 も優秀

35

Codexの基本操作：チャットエリア

プロンプトを入力するチャットエリアで、
プロンプトに関連する設定が可能

コンテキスト

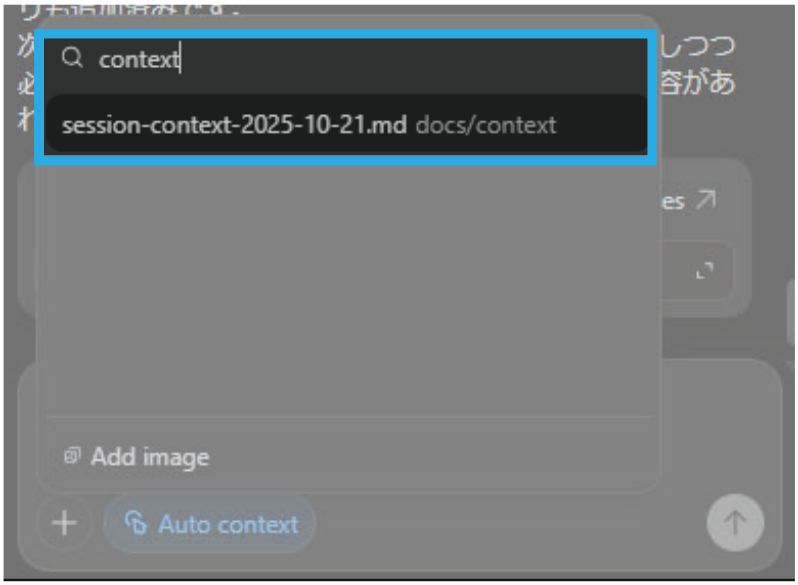


作業環境　AIの権限　使用モデル

36

Codexの基本操作：コンテキスト

事前に作成しておいたコンテキストを読み込ませることで、回答の精度を高めることが可能



37

Codexの基本操作：作業環境の選択

作業環境をローカル or クラウドで選択可能
現在のAPIレート制限の確認も可能

Work localy

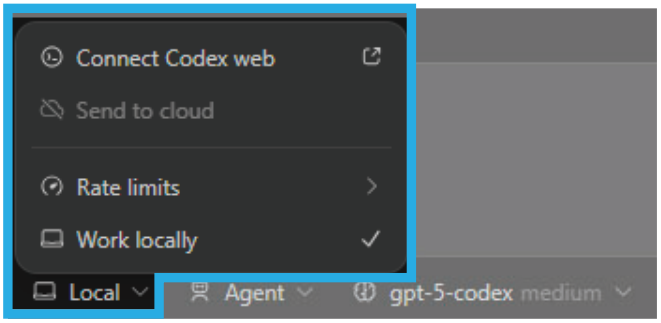
ローカル環境で実行

Send to Cloud

クラウド環境で実行
※設定しなければ使用不可

Rate limits

APIレート制限状況の確認

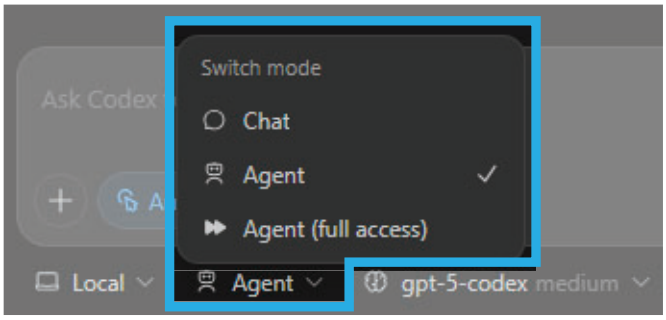


38

Codexの基本操作：AIの権限の選択

予期せぬファイル操作を回避するために、
適切なワークスペース設定と権限割振が必須





CLI型と同様に、強力な権限を与えてしまうと、
PC内の予期せぬファイルを破壊するリスクがある！

39

Codexの基本操作：AIの権限の使い分け

慣れるまでは、基本的にAgentで開発を進める
Agent（full access）は大量のファイルを作成する際に使用

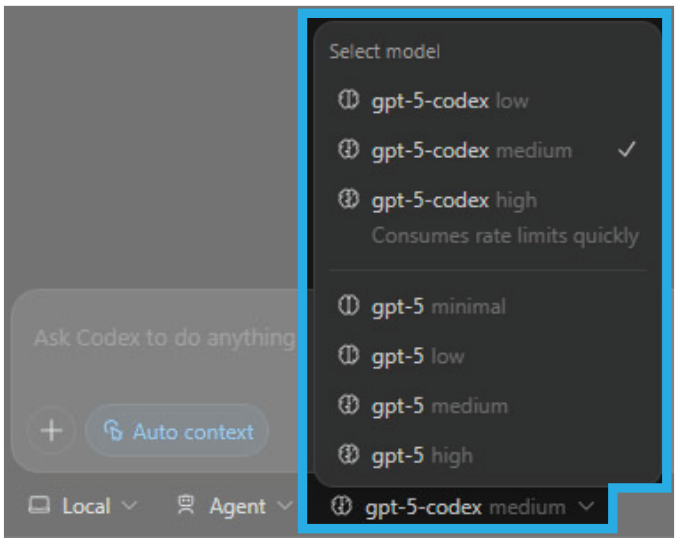
＼おすすめ／

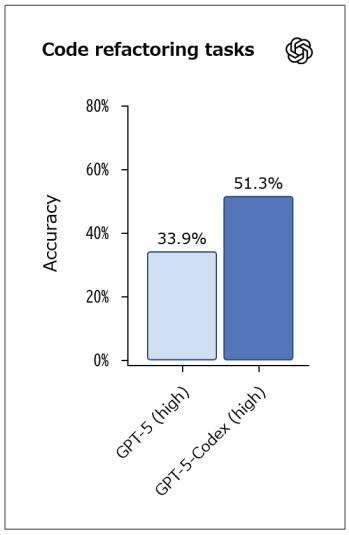
	Chat	Agent	Agent（full access）
ファイル読込	○	○	○
フォルダ作成	×	○	○
ファイル書込	×	▲ 承認制	○
利用シーン	作業の整理	通常の開発	大量のファイル作成

40

Codexの基本操作：使用モデルの選択

実施したい作業にあわせて使用するモデルを選択可能。
基本的には「gpt-5-codex medium 」使用。





参考文献：https://openai.com/ja-JP/index/introducing-upgrades-to-codex/

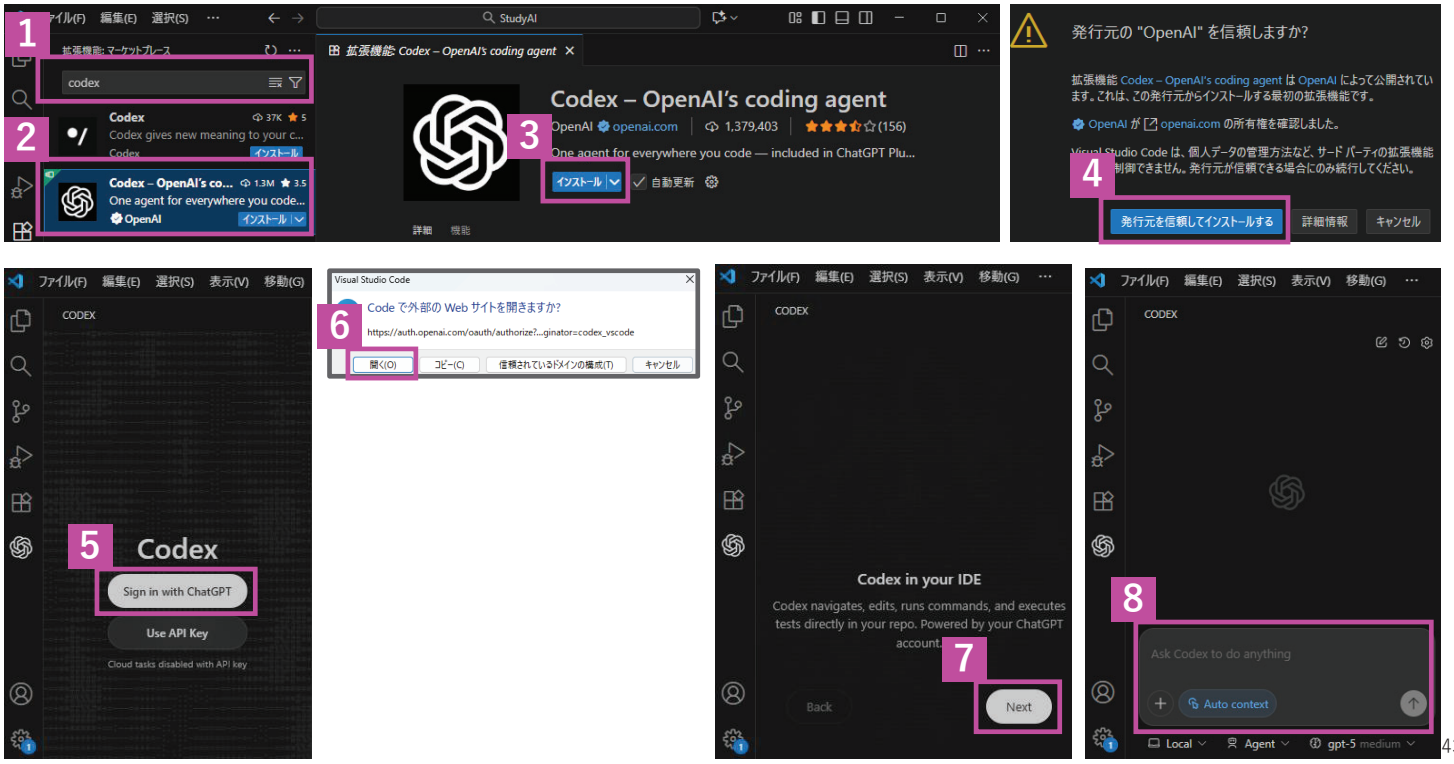
41

目次

1 はじめに
2 AIを取り巻く環境
3 活用事例
4 CLI型生成AIの概要
5 ワーク① CLI型生成AI
6 コンテナ型生成AIの概要
7 ワーク② コンテナ型生成AI
8 ワーク③ 実践開発課題
9 ワーク④ 自社の業務における活用
10 まとめ

42

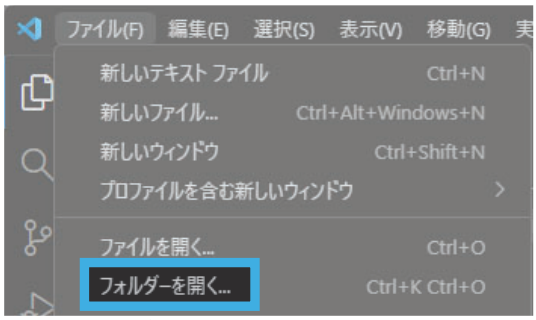
ワーク②：事前準備：Codex導入手順

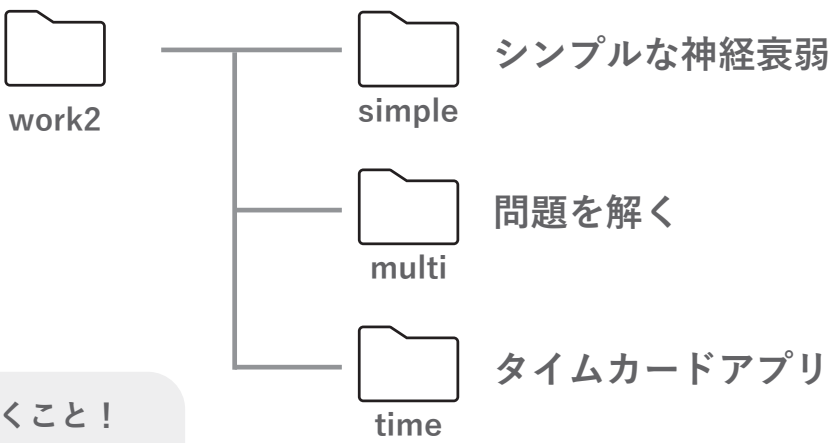


43

ワーク②：ワークの説明　フォルダ構成

フォルダ構成を間違えると非常に危険なため
入念に注意して環境を作成する





必ず**各作業フォルダを作成**し開くこと！
(simple・multi・time)

※フォルダを開く場所を間違えると
PCが破壊されるリスクがある

44

ワーク②：シンプルなブラウザゲームを作成する①

シンプルなブラウザゲームを作成し、コンテナ型AIの基本操作を学習
フォルダ内の資料を読み取れることを確認

事前準備

simpleフォルダの直下にSPEC.mdを作成し、中には以下の通り入力。
「ブラウザで動くシンプルな神経衰弱ゲームを作成してください。」

プロンプト

コンテナ型CLIに以下の通りプロンプトを送信。
「フォルダ内の仕様書(SPEC.md)を参考に、プログラムを開発してください。 」

ワーク②：シンプルなブラウザゲームを作成する②

コンテナ型AIの基礎的な操作方法を学ぶ

事前準備

simpleフォルダの直下にSPEC2.mdを作成し、中には追加機能を入力。
「追加機能として、XXXXXXXXXX機能を追加してください。」

課題

AIの権限をChatやAgent(full access)に変更し、Agentとの違いを体験する。

ワーク②：Codexと複雑な問題を解く：マルチモーダルを体験する

AIは文字だけでなく、画像やExcel、音声など様々な形式で資料を読み取れることを確認

事前準備

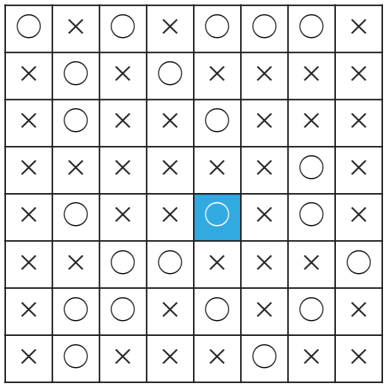
次の2ページに渡る問題文のスクリーンショットをフォルダに設置

プロンプト

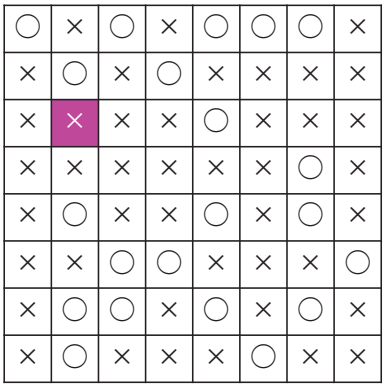
問題文の画像を読み取ってAlice_actionとBob_actionを作成してください

ワーク②：Codexと複雑な問題を解く：2人の囚人とチェス盤：問題文

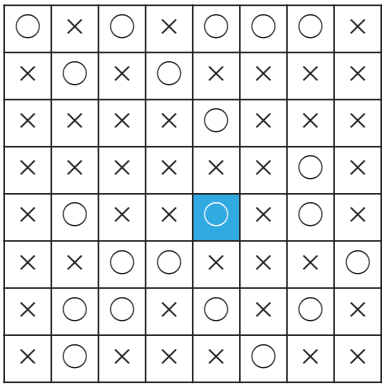
- 部屋に8×8＝64マスのチェス盤がある
- 各マスに表裏のコインが1枚ずつ配置
- 看守は64マスのどれか1マスを「正解」マスとしてAliceに教える
- Aliceはコインを1枚だけ裏返すことが可能
- Aliceと看守が退室後、Bobが入室
- Bobはチェス盤を見るだけで「正解」マスを当てなければならない
- AliceとBobは事前に戦略を打ち合わせ可能だが、ゲーム実行中の合図・会話は禁止



看守は正解マスを指定



Aliceは1枚を反転



Bobは正解を特定

ワーク②：Codexと複雑な問題を解く：2人の囚人とチェス盤：作成する2つの関数の入出力

Alice_action

入力　看守の正解マスが与えられる

出力　ボードをランダム生成後、
1つのコインを反転後のボードを出力

入力例　3-2

出力例　表裏表裏表裏裏表
表裏表裏表裏裏表
表裏表裏表裏裏表
表裏表裏表裏裏表
表裏表裏表裏裏表
表裏表裏裏裏裏表
表裏表裏表裏裏表
表裏表裏表裏裏表

Bob_action

入力　ボードの状態が与えられる

出力　看守の正解マスを求める

入力例　表裏表裏表裏裏表
表裏表裏表裏裏表
表裏表裏表裏裏表
表裏表裏表裏裏表
表裏表裏表裏裏表
表裏表裏裏裏裏表
表裏表裏表裏裏表
表裏表裏表裏裏表

出力例　3-2

49

ワーク②：タイムカードアプリを作成

シンプルなタイムカードアプリを作成し、データベース操作を行う。

プロンプト

以下の技術を用いて、
ブラウザで動くシンプルな
タイムカードアプリを作成してください。

・HTML,CSS,Javascript
・localStorage

以下の機能は最低限作成してください。

・出勤と退勤の打刻
・打刻履歴の確認
・打刻の削除



50

目次

1 はじめに

2 AIを取り巻く環境

3 活用事例

4 CLI型生成AIの概要

5 ワーク① CLI型生成AI

6 コンテナ型生成AIの概要

7 ワーク② コンテナ型生成AI

8 ワーク③ 実践開発課題

9 ワーク④ 自社の業務における活用

10 まとめ

ワーク③：実践開発課題 ワークの説明

実務に即した開発課題に挑戦することで、
CLI型・コンテナ型AIの活用方法を学ぶ

開発シミュレーション	ミニアプリ開発

最初にどちらか選んでからワークに着手する

本講義の注意点

このワークの中では、
Express等のインストールを求められる可能性があります。

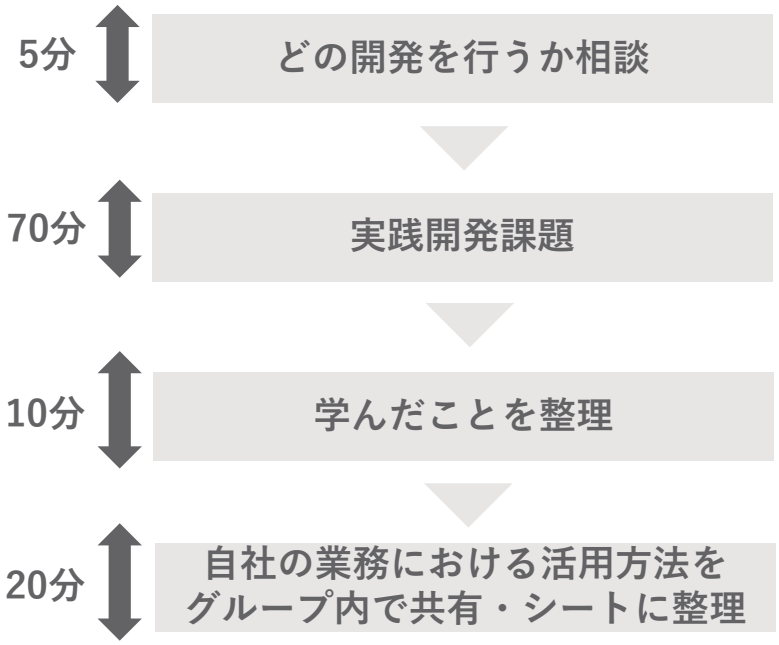
CLI型AI・コンテナ型AIにインストール方法を聞くことで
手順を回答させることが可能ではありますが、

不明な点があれば気兼ねなく**講師にご質問**ください。

53

ワーク③：実践開発課題　ワークの説明　開発シミュレーション

CLI型 or コンテナ型AIを活用し、実践的な開発をシミュレーション
実施した意見を発表する



グループの中で、
それぞれがどの開発手法・ツールで
開発するのかを分担する
※オリジナルのアプリを開発
※開発内容はシンプルなものを推奨
※思いつかない場合は、
　サンプルのアプリを開発

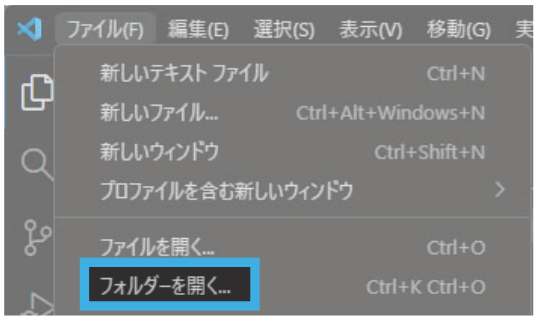
グループ内で開発手法を分けるため、
アジャイルとウォーターフォールの
担当者が1人以上になるように割振

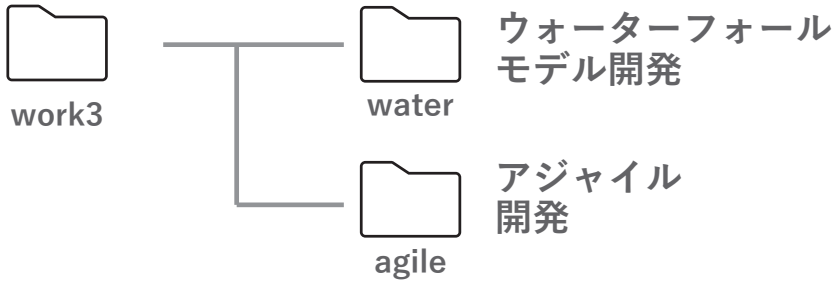
可能な限り、
CLI型とコンテナ型の担当者も
1人以上になるように割振

54

ワーク③：開発シミュレーション：フォルダ構成

実際の開発手法をシミュレーションすることで、
実務におけるコンテナ型生成AIの使用方法を学習する



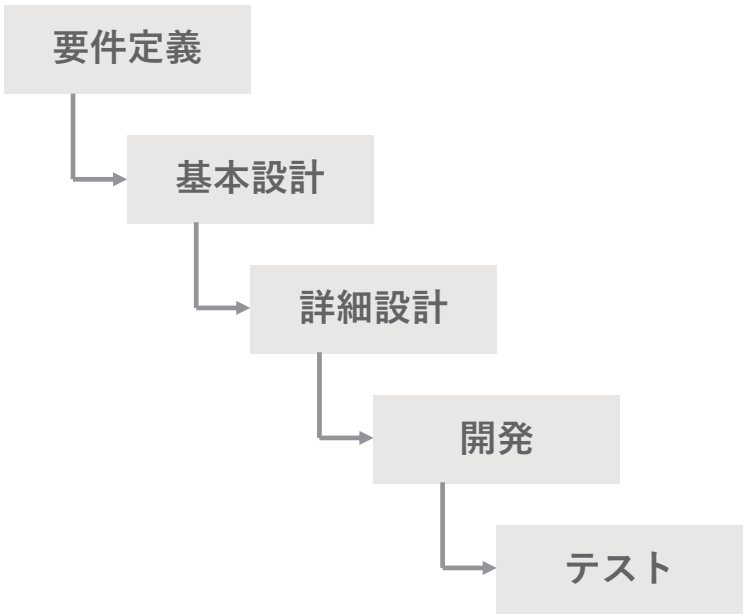


必ず**各作業フォルダを作成**し開くこと！
（water・agile）

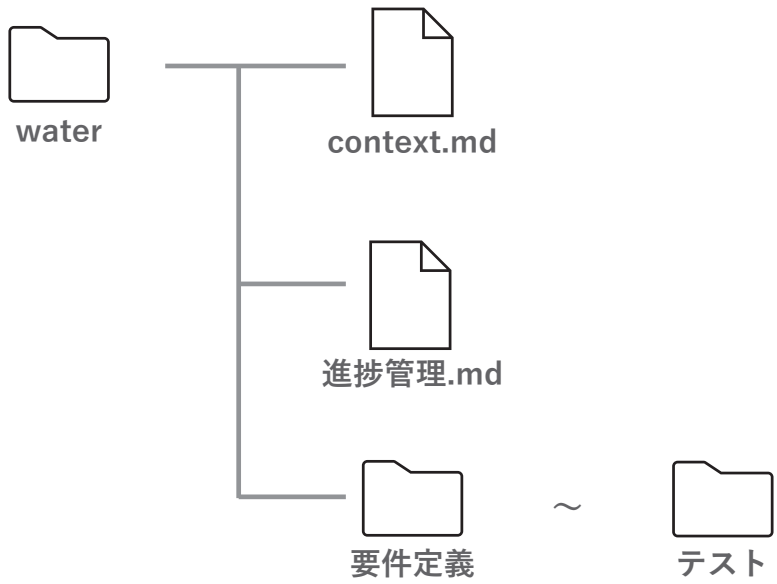
※フォルダを開く場所を間違えると
PCが破壊されるリスクがある

ワーク③：開発シミュレーション：ウォーターフォールモデルに沿った開発を実施

ウォーターフォールモデルとは、
開発工程を水が滝を流れるように一方通行で進める



ワーク③：開発シミュレーション：ウォーターフォールモデルによるフォルダ構成



57

ワーク③：開発シミュレーション：コンテキストファイルを作成する

```
# Project Context
- Purpose: シンプルな在庫管理システムを構築する。
- Stack: Node.js（Express）と SQLite。
- Methodology: ウォーターフォール型で各工程を順に進める。
- Note: 回答はすべて日本語で記載する。

## Waterfall Phases
1. 要件定義: 対象機能やデータ項目を整理し、合意を得る。
2. 外部設計: 画面や API インターフェース、入出力データを定義する。
3. 内部設計: ルーティング、ビジネスロジック、DB スキーマなど内部構造を設計する。
4. 開発: 設計成果物に基づき実装する。
5. テスト: 単体/結合/受入の観点で品質を確認する。

## Next Step
- 現在のフェーズ方針に従い、各工程を順次進める。

## 作業ルール
- 要件定義や外部設計などフェーズごとに専用フォルダを作成して管理する。
- 各フェーズの作業は対応するフォルダ内で完結させ、他フォルダを閲覧しても内容は編集しない。
- 進捗管理は`make`直下の`進捗管理.md`で行い、フェーズ完了時に内容を記録する。
- 手戻りが発生した場合も、原因と対処を含めて必ず`進捗管理.md`に追記する。
```

58

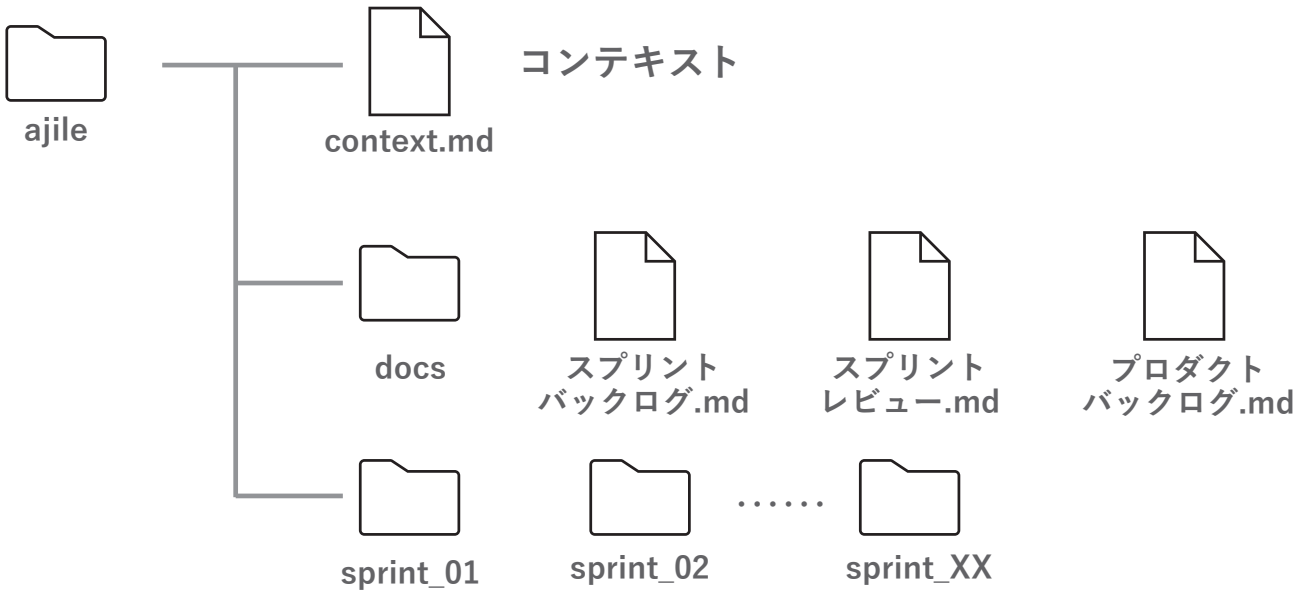
ワーク③：開発シミュレーション：アジャイル開発（スクラム）を実施

アジャイル開発（スクラム）とは、
開発工程を1つの機能を単位とした小さいサイクルで繰り返す



59

ワーク③：開発シミュレーション：アジャイル開発によるフォルダ構成



60

ワーク③：開発シミュレーション：コンテキストファイルを作成する

```
# プロジェクトコンテキスト

## プロダクト概要
- 商品入出庫を一元管理する業務アプリケーション
- Node.js (Express) と SQLite による軽量 API + 組み込みデータベース構成

## 開発手法
- プロダクトバックログ精査 → スプリント計画 → 開発・スプリントレビュー → スプリントレトロスペクティブの反復サイクル
- `docs/スプリントレビュー.md` によるデモ内容・完了プロダクトバックログアイテム・ステークホルダーフィードバックの時系列記録

## リポジトリ構成メモ
- `docs/` 配下にプロダクトバックログ、スプリントバックログ、ユーザーストーリー等のドキュメント集約
- `sprint_01/` 以降番号を増やしながら各スプリント成果物と `スプリントタスク.md` を配置
- `docs/スプリントレビュー.md` を `docs/スプリントバックログ.md` からの参照先として運用
- 新規スプリントを追加するたびに次番号の `sprint_xx` フォルダを作成し、対応ドキュメントを更新
```

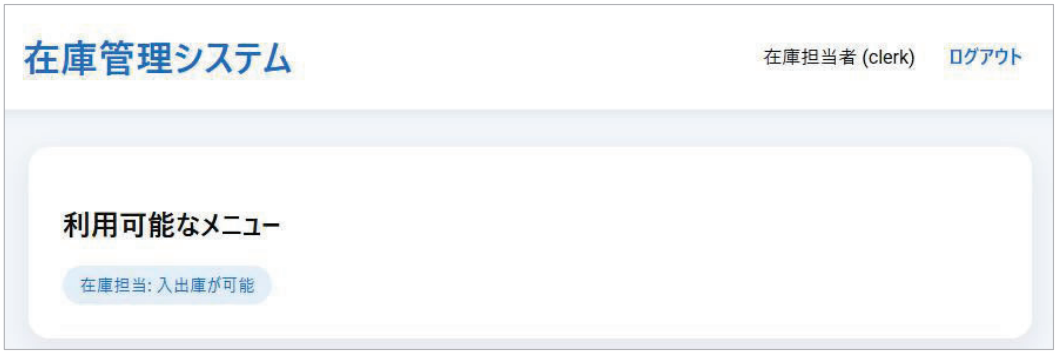
61

ワーク③：開発シミュレーション：サンプル画面：ログイン画面



62

ワーク③：開発シミュレーション：サンプル：ヘッダー部分



63

ワーク③：開発シミュレーション：デモサイト：在庫サマリー（全アカウント閲覧可能）画面



64

ワーク③：開発シミュレーション：デモサイト：入出庫登録（在庫担当者のみ）画面



65

ワーク③：開発シミュレーション：デモサイト：重要商品設定（現場リーダーのみ）画面



66

ワーク③：開発シミュレーション：デモサイト：商品マスタ登録（システム管理者のみ）画面



67

ワーク③：開発シミュレーション：デモサイト：アカウント一覧・デモサイトURL

アカウント一覧

ロール	ログイン名	パスワード	用途
在庫担当者	clerk_user	clerk1234	入庫登録 出庫登録
現場リーダー	leader_user	leader1234	重要商品設定
システム管理者	admin_user	admin1234	商品マスタ登録

デモサイトURL

https://demo-a9fj3q2lxp.syscari.com/

68

ワーク③：開発シミュレーション：開発をシミュレーションした結果、学んだことを記載

	開発シミュレーションから学んだこと
ウォーターフォールモデル	
アジャイル開発	

69

ワーク③：実践開発課題　ワークの説明　ミニアプリ開発

CLI型 or コンテナ型AIを活用し、実践的な開発をシミュレーション
実施した意見を発表する

10分　どの開発を行うか相談

70分　ミニアプリ開発

グループの中で、
それぞれがどのツールで
開発するのかを分担する
※オリジナルのミニアプリを開発
※思いつかない場合は、
　サンプルのアプリを開発

10分　学んだことを整理

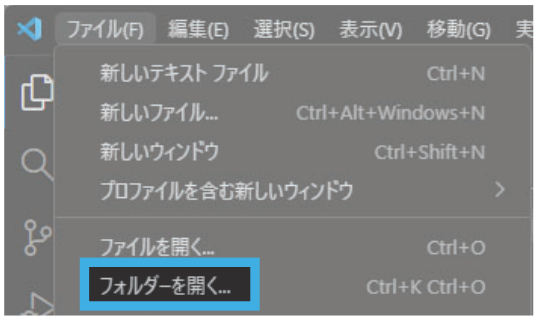
20分　自社の業務における活用方法を
グループ内で共有・シートに整理

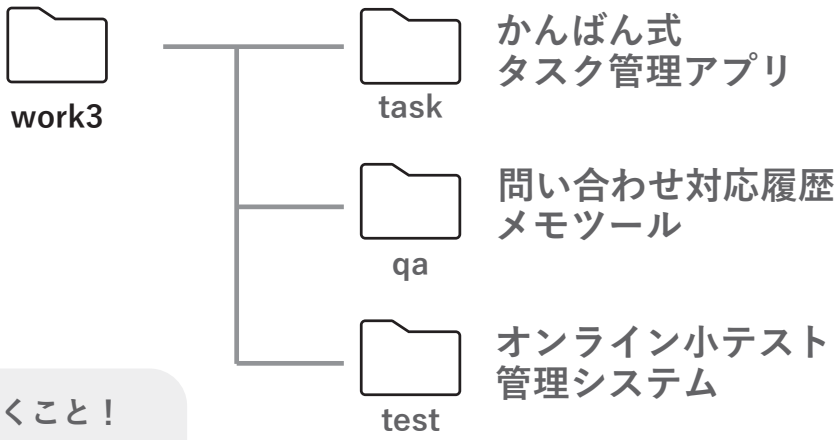
グループ内でツールを分けるため、
CLI型・コンテナ型の
担当者が1人以上になるように割振

70

ワーク③：実践開発課題　ワークの説明：フォルダ構成

実際の開発手法をシミュレーションすることで、
実務におけるコンテナ型生成AIの使用方法を学習する





必ず**各作業フォルダを作成**し開くこと！
（water・agile）

※フォルダを開く場所を間違えると
PCが破壊されるリスクがある

71

ワーク③：ミニアプリ開発：

作成するアプリは以下の3種類のアプリから選択する。

かんばん式 タスク管理アプリ	問い合わせ対応履歴 メモツール	**オンライン小テスト 管理システム （※）**

※難易度が高いため自信のある人か他2つを開発した後、
時間に余裕のある人が取り組んでください。

72

ワーク③：ミニアプリ開発：かんばんタスク管理アプリ



73

ワーク③：ミニアプリ開発：問い合わせ対応履歴メモツール：新しい履歴を追加



74

ワーク③：ミニアプリ開発：問い合わせ対応履歴メモツール：検索＆フィルタ



75

ワーク③：ミニアプリ開発：問い合わせ対応履歴メモツール：案件タイムライン



76

ワーク③：ミニアプリ開発：オンライン小テスト管理システム：受講生画面　テストコード入力



77

ワーク③：ミニアプリ開発：オンライン小テスト管理システム：受講生画面　情報登録



78

ワーク③：ミニアプリ開発：オンライン小テスト管理システム：受講生画面　問題画面



79

ワーク③：ミニアプリ開発：オンライン小テスト管理システム：受講生画面　結果画面



80

ワーク③：ミニアプリ開発：オンライン小テスト管理システム：管理者画面　管理者ログイン



81

ワーク③：ミニアプリ開発：オンライン小テスト管理システム：管理者画面　テスト一覧



82

ワーク③：ミニアプリ開発：オンライン小テスト管理システム：管理者画面　テスト作成



83

ワーク③：ミニアプリ開発：オンライン小テスト管理システム：管理者画面　問題一覧



84

ワーク③：ミニアプリ開発：オンライン小テスト管理システム：管理者画面　問題作成



85

ワーク③：ミニアプリ開発：オンライン小テスト管理システム：管理者画面　選択肢追加



86

ワーク③：ミニアプリ開発：オンライン小テスト管理システム：管理者画面　テスト結果確認



87

ワーク③：ミニアプリ開発：ミニアプリ制作を実施した結果、学んだことを記載

	ミニアプリ制作から学んだこと
CLI型	
コンテナ型	

88

ワーク③：CLI型・コンテナ型生成AIを自社の業務において活用可能か考察・発表

自社の業務でどのように活用できるか議論

自社の業務における活用方法

目次

1 はじめに

2 AIを取り巻く環境

3 活用事例

4 CLI型生成AIの概要

5 ワーク①　CLI型生成AI

6 コンテナ型生成AIの概要

7 ワーク②　コンテナ型生成AI

8 ワーク③　実践開発課題

9 ワーク④　自社の業務における活用

10 まとめ

ワーク④：グループを変更して議論

ワーク③とは異なるグループで議論を行い、様々な意見に触れる

10分 ↕ 自社の業務における活用方法を共有

▼

20分 ↕ これからのエンジニアの在り方を議論

91

ワーク④：CLI型・コンテナ型生成AIを自社の業務において活用可能か考察・発表

自社の業務でどのように活用できるか議論

ワーク③で作成した資料を基に
他のグループの意見・ワークへの取り組み内容に触れることで学習する

92

ワーク④：CLI型・コンテナ型生成AIを自社の業務において活用可能か考察・発表

生成AIが進化していく中で、
エンジニアとしてこれからどのように考えるのか議論を行う

今後のエンジニアの在り方について議論

目次

1 はじめに

2 AIを取り巻く環境

3 活用事例

4 CLI型生成AIの概要

5 ワーク①　CLI型生成AI

6 コンテナ型生成AIの概要

7 ワーク②　コンテナ型生成AI

8 ワーク③　実践開発課題

9 ワーク④　自社の業務における活用

10 まとめ

まとめ：CLI型・コンテナ型AIの台頭による人間とAIの関係

AIはプログラムを人間以上に把握し、
「自立開発」が可能に進化しつつある



便利な道具

人が聞いたことだけを
AIが答える

共同開発者

パートナーとして、
人とAIが共同開発

自立開発

人が実現したいことを
AIが叶えてくれる

95

令和7年度 文部科学省委託事業
専門職業人材の最新技能アップデートのための専修学校リカレント教育推進事業
専門職業人材の最新技能アップデートプログラムの開発

キャリアアップを目指すプログラマーを対象とした
AI活用の知識・スキル獲得プログラム開発事業

令和7年度事業 成果報告書

2026年3月発行

発行者　一般財団法人 日本検定基盤財団
〒145-0071　東京都大田区田園調布1-8-21
TEL ：03-3722-1788　FAX ：03-3583-1855
http://www.waken.or.jp/

本報告書は、文部科学省の教育政策推進事業委託費による委託事業として、≪一般財団法人日本検定基盤財団≫が実施した令和7年度「専門職業人材の最新技能アップデートのための専修学校リカレント教育推進事業」の成果をとりまとめたものです。

※禁無断転載