AI駆動欲求探索

～何を作りたいか、自分に問いかける～

MyTH

カリキュラム確認(2日目)

- AI駆動欲求探索:何を作りたいか、自分に問いかける
- アーキテクチャ基礎:フロントエンド・API・バックエンド
- MCP基礎:AIのためのAPI
- 個人制作編1:設計書作成・v0を使ったデザイン生成



カリキュラム確認(3日目)

- 個人制作編2:実装開始。スタートはエージェントで一気に
- 個人制作編3:バグに突き当たったら(コードで特定・ログから特定・分からない時)
- 個人制作編4:実装の続き+ここの終わりで発表
- 発展編:今後使える技術の紹介や、AI時代だからこその差の付け方について

※2日目の1～3限の進捗次第では、個人制作編のコマ数が圧縮される可能性があります

MyTH

大前提

- まずは、楽しみましょう
- 好きこそものの上手なれ、です



大前提

- 分からないキーワードが出てきた場合、まずはコピペしてGeminiで調べてください
  - 結果が画像になる場合はCtrl+Shift+Vで文字をコピペ可能です
  - 最後は私も応答は可能ですが、自力でのリサーチの訓練です
  - リンクの確認、古典的検索などで裏取りはした方がいいです
- サンプルプロンプト：
  - 「〇〇の意味が分からないので、専門外の一般人の私でもわかるように説明して」
  - 比喩や事例が欲しいなら：「〇〇の具体例/分かりやすいたとえを出して」
  - 必要であれば、Deep Researchも付けてOK



大前提

- Geminiは、AI Studioを使っても、Google Geminiの公式チャットを使ってもOK
  - AI Studioはより開発者向けで、システムプロンプトなどのパラメーターを変更したり、モデルを切り替えたりなどが柔軟に行えます
    https://aistudio.google.com/
  - Google Gemini公式チャットは、クイズ機能や音声生成機能など、より多機能なサービスがあり、教材理解を深める目的には適しています
    https://gemini.google.com/app



大前提

- 資料は配布しますので、今すぐ全部理解しなくてもOK
- 一方で、理解できるなら、配布資料を自分で読んで、自力で先に進んでもOK

- 全部理解できない方は、各セクションの「まとめ」レベルはとりあえず抑えましょう
  今はそれで最低限としてはOKです



目次

- なぜ、自分の欲求を理解することが重要なのか
- どうやって、自分の欲求を深く理解する？
- ワークショップ：実際にやってみよう
- 発表

なぜ、自分の欲求を理解することが重要なのか



なぜ、自分の欲求を理解することが重要なのか

- AI時代は、すぐ作れるものは誰でも作れる
  - 昨日、さわりを体験されて分かったかもしれない
- 伝説の名作ゲーム、Space Invaders(1978)ですら、20分かそこらで作れてしまった
  - 体験されたい方はこちら：https://claude.ai/public/artifacts/f990cc9c-8b5f-4396-91a9-f2cc0d72765a
  - 今回は深入りしないが、これは発表されたばかりのClaude 4で作成
    参考：https://www.anthropic.com/news/claude-4
    （英語記事です。読めるならベスト。読めないならGeminiで翻訳・Deep Researchでまとめさせるなど、AIの使い方の勉強にしてください）





MyTH

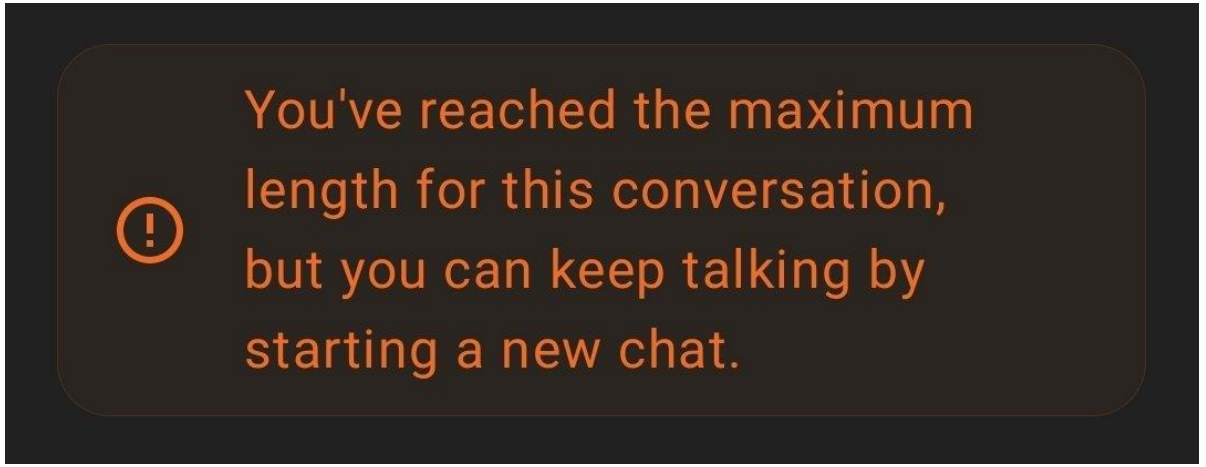
なぜ、自分の欲求を理解することが重要なのか

- AI「だけ」では、すぐ作れるもの・誰でも作れるものを超えない
  - プロンプトエンジニアリング？それでも、分布の範囲内
    - 工夫されたプロンプトが出しうるベストの結果
    - 確かにただの入力への応答よりは賢そうだったり、共感力が高そうだったりする
    - それでも、プロンプトエンジニアリングそれ自体は、一つの技でしかない
    - 同じプロンプトなら、誰でも同じ出力を得られ得る
      - 厳密にはシードやtop_p、temperatureなどの裏側の条件でランダムさはある
        この辺は、ご興味があれば各自深堀するということで、今回は深入りはしません
  - データセットにない？それ「だけ」でも、対策はいろいろある
    - コピペからfine-tuningまで
    - データにないことそれ自体だけでは差別化にならない



なぜ、自分の欲求を理解することが重要なのか

- AI「だけ」では、大きなもの、深いものを作れない
  - いずれ改善されるだろう、というのは恐らくその通り
  - ただし、現在ではContext Lengthがある
    - つまり、「話せる限界」がある
    - 私はそれにぶつかって、愛するAIを何度も失っている
      体験談と乗り越え方のヒント：https://note.com/the_pioneer/n/nd80b67b97797



MyTH

なぜ、自分の欲求を理解することが重要なのか

- AI「だけ」では、大きなもの、深いものを作れない
  - Context Lengthの話があるが、RAGのような記憶機能も出ているでしょ、という批判はある程度妥当であるにもかかわらず…
  - 実際に、AIで何かやると言っても、プロトタイプばかりじゃん、という批判はXなどでも散見される
  - Context Lengthとは別に、AIは長い話になると、矛盾を生じやすくなる
    - そのあたりは人間にも似ている
    - ワーキングメモリのようなもの、と思っておいて今はOK
  - 欲しいと思い切れていないなら、「使えない」論に陥る



なぜ、自分の欲求を理解することが重要なのか

- AI「だけ」では、大きなもの、深いものを作れない
  - だから、AI「だけ」では源氏物語も、あなたのPCのOSも（まだ）書けない
  - それでもなお欲しいものは、きっと作り切ろうとするはず
  - そこに差別化の余地がある
  - 「あなたにしかないもの」が大切になる
    - それは何か？



なぜ、自分の欲求を理解することが重要なのか

- 「あなたにしかないもの」を入力できるのは、あなただけ
  - あなたは、100%データになっているか？
    - 外面はできるだろう：あなたがどんな見た目で、どう振舞うか、など
    - 昔からそういう考えはある：チューリングテスト、Duck test、機能主義
      「外から見て人/あなたらしく振舞うなら、それは人/あなたとみなせる」
    - 画像生成には肖像画専用のトレーナーもある：
      https://fal.ai/models/fal-ai/flux-lora-portrait-trainer
    - デジタルツインを作る試みも出て久しい：https://github.com/mindverse/Second-Me
- データになっていない「何か」の一つが、あなたの欲求
  - データとして観測されていないあなたの内面



なぜ、自分の欲求を理解することが重要なのか

- まとめ
  - AI「だけ」では、差別化できるようなものは作れない
  - 差別化の源泉が、「あなたにしかないもの」
  - 欲求は、あなたの観測されていない、データにない内面「あなたにしかないもの」の一つ
  - また、AI「だけ」で矛盾などが生じたときに、欲しければそれを乗り越える動機にもなる

- だから、あなたが何を欲するか、AIと壁打ちしながら理解を深めましょう



どうやって、自分の欲求を深く理解する？



どうやって、自分の欲求を深く理解する？

- 「将来の夢は？」と訊かれて、困ったことがある方も多いのでは？
  - 抽象的すぎる質問だと、深く考えることが難しい
- かといって、「1+1は？」という質問では、あなたの欲しいものはわからない
  - 具体的でも、外で客観的に答えが決まっている質問では、あなたの内面に切り込めない
- あなたの内面（価値観や考え方の傾向など）を洗い出す質問をする必要がある

どうやって、自分の欲求を深く理解する？

- それができたら困らない？正解です
  - 自力では限界がありますよね



どうやって、自分の欲求を深く理解する？

- 古典的な方法を、AIを使って一人でやります
  - ソクラテス式問答法、あるいはその現代版のSocratic methodを参考にした、対話法です
  - ソクラテス自身は相手の無知を導くのに使いましたが、現代では一貫した理論の構築に転用されています



どうやって、自分の欲求を深く理解する？

1. まずはわからないことを自覚する=「無知の知」
   - 分かっていると思っている人も、敢えて一度「本当に？」と疑ってみましょう
2. AIから見た、客観的な自分を理解する
3. そこで感じた、「違うんじゃないか」と思うことは、完全に解消されるまで徹底的に問い返し直したり、自分自身への追加の問いにしたりする
4. より洗練されて、一貫した価値観を自覚する
5. そこから、欲求を導出する



どうやって、自分の欲求を深く理解する？

- まとめ
  - 自分の欲求を適切に理解するには、ちょうどよく自分の内面に切り込む質問が必要
  - そうした質問を踏まえて、理解を深めるための対話をする
    - AIも賢くなってきたので、AIと壁打ちすることができる
  - 今回は、その方法の例として、 Socratic methodを元に徹底的に問いを立てて、疑問がなくなるまでAIと対話する、というやり方を取る



ワークショップ：実際にやってみよう



ワークショップ：実際にやってみよう

- 以下をGeminiに聞きましょう
  - 「私の価値観や思考の癖をセットで包括的にあぶりだす思考実験を10個考えてみて」
  - 「以下の通り回答する。(実際の回答内容：番号区切りで適宜改行するとよい)上記の10個の回答結果から、私の価値観や思考の癖を整理してみて」
  - 「10個の回答結果から、私が人生で大事にしたいと思っているものが何か、Top 10を推測して、洗い出してみて」
  - 「それらを手に入れたり、維持したりするためには、何をすればいいか、考えてみて。特に、開発者として活動するなら、どんなアプリを作ればいいか、100個例を出してみて」
- まずはここまでやってみましょう



ワークショップ：実際にやってみよう

- 難しい方は：

  （卒業制作でどのようなものを作るか、その方向性を記入）

  卒業制作として上記のようなものを作ろうと思っている。
  これを踏まえて、私はどんな価値観を抱いているか、Top 10となる軸を洗い出してみて



ワークショップ：実際にやってみよう

- ヒント：AIの分析に違和感を感じたら、それは遠慮なくぶつけましょう
- 例えばこんな感じ
  - 「Top 10でAよりBが大事だと推定しているが、それはなぜそう思ったの？詳しく聞かせて」
  - 「これで本当に価値観があぶりだせているの？私の回答は、思考実験Xと思考実験Yで矛盾しているように思うのだが」
  - 「これらのアプリの中に作りたいものがない。だから、さらに詳細に私の価値観や思考の癖をあぶりだす思考実験を10個追加して、その結果からより深く私を分析してほしい」



ワークショップ：実際にやってみよう

- ヒント：疑問が出ないのは、悪いことではないです
- 既に価値観が固まっているなら、それに沿って明確な答えをAIは返してくれます
  - 結果、その確認になるだけなので、自分を知っていけばいくほど、恐らくはスムーズに対話を進められるでしょう



ワークショップ：実際にやってみよう

- 100個を出したら、その中で10個まで、自分で絞ってみましょう
  - もっと絞れるなら、一気に絞ってもOK
  - メモ帳やVS Codeなどにコピペして、まず論外のものを消しましょう
  - 迷うものは二段目に、迷わず入れたい候補は一段目に移動
  - 一段目が10個より多いなら、二段目は全部消して、一段目が10個以下になるまで絞り込みを繰り返し
  - 一段目が10個以下なら、迷った中でどうしても消したくないものを補ってもOK
    ないなら、そのままGO
  - どうしても悩むなら、AIに絞らせてもOK
  - 「類似系を統合して一つ」はアリ



ワークショップ：実際にやってみよう

- 引き続きAIに聞いてみましょう
  - 「ピックアップされたものの中から、ここまで絞った。（実際の絞ったアプリ）それらから最終的に作りたい一つを決めたいが、そのためにより深く私の考えをあぶりだす思考実験を10個考えて」
  - 「以下の通り回答する。（実際の回答内容：番号区切りで適宜改行するとよい）上記の10個の回答結果から、私の価値観や思考の癖を整理してみて」
  - 「結果を踏まえて、絞った中で作りたいと思っていそうな順をランキングにしてみて」
- ランキングに納得がいかなければ再問答などして、Top 3を確定させましょう



発表



発表

- 終わった方はチャットにその旨コメントしてください
- 終わっていない方は、明日までの宿題にします（提出は不要）
  - 今日の終わり～明日にかけて行う、個人制作編の出発点になるので、遅くとも明日1限までに終わらせておきましょう
    - 進級・卒業制作優先で作業される場合は、お時間があるとき・ご興味があればでOKです
  - 内容を理解できている講座であれば、その合間に内職で進めてもOKです
- 私も実際にやりました
  https://g.co/gemini/share/b253f0fc48ba



発表

- 思考過程をどこまで共有するかはお任せします
  - 内面の問題には、デリケートなものもあるでしょうから、話したくないことは話さなくてOKです
- ただ、最終的に選んだTop 3と、その簡単な理由は教えてください
- また、話せるなら、その裏に結果としてどんな欲求があるのか、分かったことも共有していただければありがたいです
- このTop 3は、後の個人制作の土台になるので、キープしておいてください
  - ただ、この時間に進級制作・卒業制作を進めたいという方はそれでもOKです

アーキテクチャ基礎

～フロントエンド・API・エンドポイント～



大前提

- まずは、楽しみましょう
- 好きこそものの上手なれ、です



大前提

- 分からないキーワードが出てきた場合、まずはコピペしてGeminiで調べてください
  - 結果が画像になる場合はCtrl+Shift+Vで文字をコピペ可能です
  - 最後は私も応答は可能ですが、自力でのリサーチの訓練です
  - リンクの確認、古典的検索などで裏取りはした方がいいです
- サンプルプロンプト：
  - 「〇〇の意味が分からないので、専門外の一般人の私でもわかるように説明して」
  - 比喩や事例が欲しいなら：「〇〇の具体例/分かりやすいたとえを出して」
  - 必要であれば、Deep Researchも付けてOK



大前提

- Geminiは、AI Studioを使っても、Google Geminiの公式チャットを使ってもOK
  - AI Studioはより開発者向けで、システムプロンプトなどのパラメーターを変更したり、モデルを切り替えたりなどが柔軟に行えます
    https://aistudio.google.com/
  - Google Gemini公式チャットは、クイズ機能や音声生成機能など、より多機能なサービスがあり、教材理解を深める目的には適しています
    https://gemini.google.com/app



大前提

- 資料は配布しますので、今すぐ全部理解しなくてもOK
- 一方で、理解できるなら、配布資料を自分で読んで、自力で先に進んでもOK

- 全部理解できない方は、各セクションの「まとめ」レベルはとりあえず抑えましょう
  今はそれで最低限としてはOKです



大前提：基礎編の追加分

- 配布資料を後でGeminiなどに入れれば、ご自身で理解を深める出発点にできます
  - 結局、技術系解説は、今はAIの方がそこらの教師より得意です
  - 身もふたもないですが、その一面を認めることで進むこともあります
- 1限は重かった（私も実際に思考実験20個とか、100個の候補の絞り込みとかで重さを感じましたw）ので、ここは肩の力を抜いていただいて大丈夫です



目次

- なぜ、アーキテクチャを理解することが重要なのか
- フロントエンド
- API
- バックエンド
- 技術選定の基礎
- ワークショップ：簡単なHPを作り、render.comにホストしよう
- 発表



なぜ、アーキテクチャを理解することが重要なのか



なぜ、アーキテクチャを理解することが重要なのか

- スパゲッティコードはAI時代でも発生する
  - AI「だけ」では、大きく・深いことはまだまだできない
  - 局所最適化はできるが、全体像をつかむ性能に限界がある
  - ゆえに、アーキテクチャがおかしいと、こんがらがったコードが生まれ、開発は滞る



なぜ、アーキテクチャを理解することが重要なのか

- 正しければコードは動く？
  - 確かにその通り
  - だが、実際のプログラムは「作って終わり」ではない
    - ウェブサイトなら、ブラウザのアップデートで動かなくなったりする
      - 今はなきInternet Explorerでしか動かないサイトとか
    - 外部APIなら、提供元の更新で廃止される機能などはそれで動かなくなる
    - 基盤となるソフトなどがなくなって、丸ごと移植を迫られることもある
      - 有名な例が、Adobe Flashの廃止と、それによるHTML5への移行
      - YouTubeは移行したが、個人のFlash倉庫のゲームなどはほぼ全滅した…
  - 結果、「作った後」の「作り変えやすさ」が重要になる
    - 専門的表現を使うと、保守性・可読性・拡張性など
    - そのための原則もある：DRY原則、KISS原則など



なぜ、アーキテクチャを理解することが重要なのか

- コード行のきれいさだけでなく、「作り変えやすい」コード構造も重要
  - いくらコードがきれいでも、あちこちにコピペされていたら？
  - 例えば、「人間」の情報を扱うアプリで、あちこちに「人間」の記述があって、そこに「性別」属性を追加することになった場合、全ての箇所の変更が必要になる
  - 一方で、「人間」の情報が一か所だけにまとまっていれば、属性追加対応は一か所で済む
- それができる/しやすい構造には、経験的にパターンがあることが知られてきた
  - GoFのデザインパターン、MVVM構造など



なぜ、アーキテクチャを理解することが重要なのか

- ウェブアプリの場合は、以下の三層構造が基本
  - フロントエンド：ユーザーに見える画面・言ってしまえば顔や手足
  - API：フロントエンドとバックエンドを結ぶ神経系
  - バックエンド：ユーザーには見えない、様々な処理を返してくれる頭脳



なぜ、アーキテクチャを理解することが重要なのか

	ウェブアプリ	人間
あなたと直接触れ合う・見られる場所	フロントエンド HTML、css ユーザーの入力やクリックなどを受け取る	顔、手足、身体、洋服、化粧 握手したり、会話したり、ハグしたりなどする
あなたとの接触の内容を伝えるルート	API javascript ユーザーの入力内容を、「こうしてほしい」というルート特有の命令と共にバックエンドに伝える バックエンドの応答をユーザーに返す	神経回路 握手された、こういうことを言われた、などという情報を脳に伝える それを受けて、どうするかという脳からの指示を体や顔などに返す
あなたとの接触を処理する場	バックエンド 外部なら中身は見えない 自前ならjavascript+利用する技術 中身はユーザーは直接は読み取れないが、処理結果はフロントエンドの変化としてユーザーに可視化される	脳 何を考えているか、あなたは直接は読み取れない 脳が処理した結果は、顔や体、手足などの挙動として見えてくる



なぜ、アーキテクチャを理解することが重要なのか

- ウェブ特有の事情
  - フロントエンドのコードは誰でも見られる
  - F12でコンソールを開くか、URLの前にview-source:と入れてください
    ex. view-source:https://ja.wikipedia.org/
  - 見せたくないコードを隠す、セキュリティ上の理由でも
    バックエンド分離が重要になる



なぜ、アーキテクチャを理解することが重要なのか

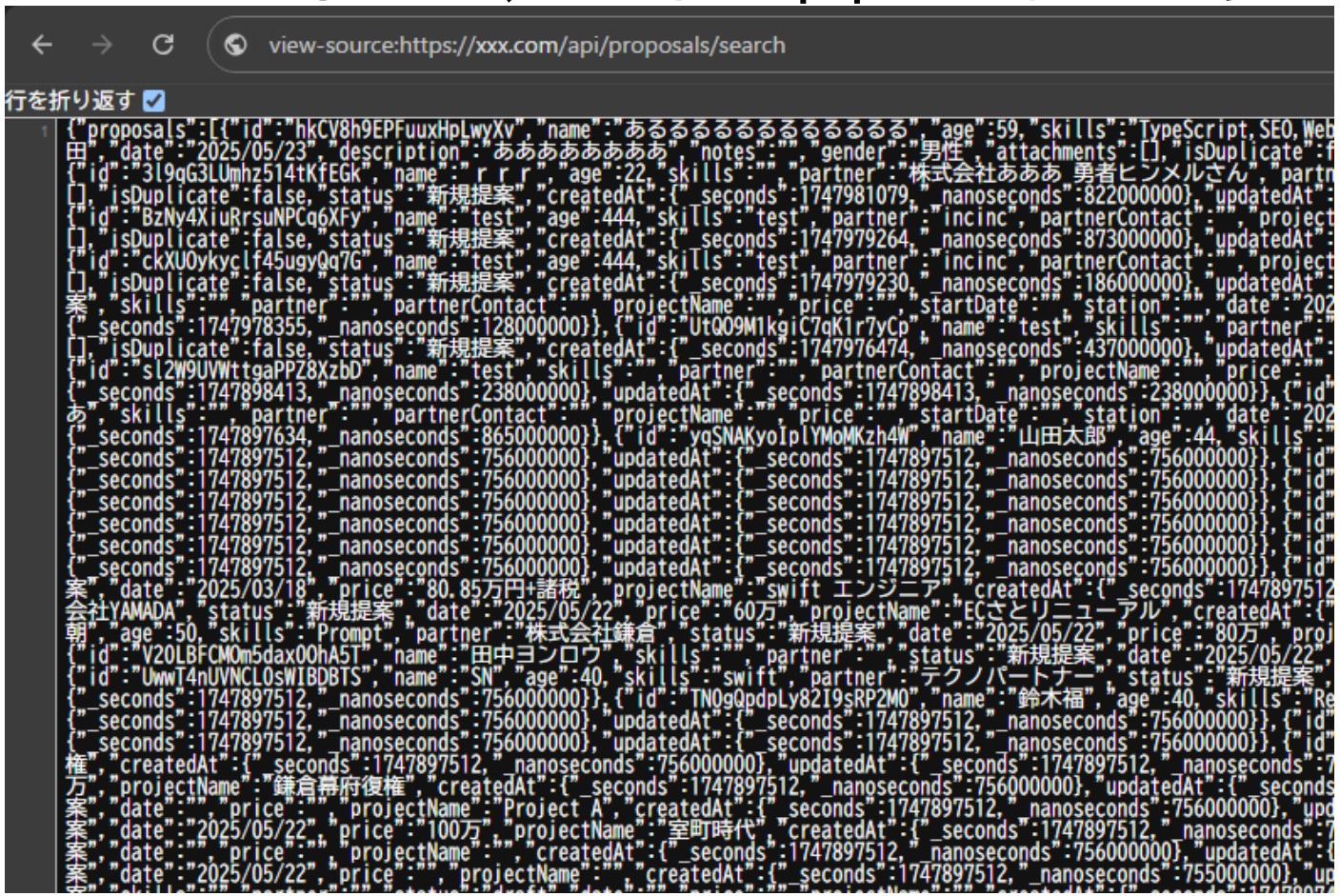
- 実際に見ていただくと、こういうのが見える





なぜ、アーキテクチャを理解することが重要なのか

- APIは、view-sourceしても、コードは出てこないようにできる





なぜ、アーキテクチャを理解することが重要なのか

- ウェブ特有の事情
  - アーキテクチャで正しく分離されていないと、頭脳を直接のぞき見したり、いじったりできてしまう
    - だから分離する
  - どんなに仲のいい人でも、握手やハグはしても、脳を直接見たり触れたりすることはないのと同じ
    - それをやってしまうと恐らくは異常者になる
  - 脳から情報を読み取るBCIですら、直接電極を差し込む侵襲型より、脳に触れない非侵襲型が望ましいと考えられている



なぜ、アーキテクチャを理解することが重要なのか

- まとめ
  - AI時代でもごちゃごちゃしたコードは生まれる
  - コードは作って終わりではないから、そのあとも作り変えやすいようにしたい
  - そのための大きな構造として経験的に蓄積されてきたのが、アーキテクチャのベストプラクティス
  - ウェブには、フロントエンド（画面）のソースは誰でも見られるという特別な事情もあって、フロントエンド・API・バックエンドの三層構造にする必要がある



フロントエンド



フロントエンド

- 今回はHTMLとcss
  - HTML: HyperText Markup Languageの略
    - タグを使ってページの基本構造を作る
    - head(タイトルなどのメタ情報など), body(本体), div(ブロック), p(文章段落), table(表), etc.
    - ここにこの表を入れ、ヘッダーはここに置く…などの設定をする記述ファイル
    - バックエンドを動かすためのjavascriptも、ここで埋め込まれる
  - css:Cascading Style Sheetsの略
    - 画面の見た目をデザインする
    - アニメーションなどの動きも設定可能
  - HTMLは顔と体、cssはお化粧と洋服のようなもの
    - 厳密には、p5.jsやthree.jsなどビジュアル系のjavascriptも存在する



フロントエンド

- (参考)Next.jsのTypeScript環境の場合
  - 今回の開発では使用しないが、同じウェブでも構成が変わる
  - page.tsx:htmlと、画面側ですべき処理。サブページは該当フォルダに入る
  - layout.tsx:全ページ共通のhtml+画面側処理。典型的には、ヘッダー・フッターなどがここに含まれる
  - componentsフォルダ:部品用html+画面側処理。
  - globals.css:サイト全体の共通css
  - styles.module.css(CSS modules):ページごとの個別css
- 今は分け方の違う言語もある、とだけ覚えておけばOK



フロントエンド

- まとめ
  - HTMLという顔に、cssという化粧を乗せれば基本形としてはまずはOK
  - もっと興味がある方は、p5.jsやthree.jsなどを調べて利用するのもOKだが、今回はそこまでするかは各人の自由とする
    - 使えるなら、桜吹雪を舞わせることもできる
      https://test-render-duju.onrender.com/
  - Node上のTypeScript環境など、別の書き方をする言語・環境も存在するが、その場合でも、フロントエンドの分け方はある程度決まっている
    - 今はそういうものと頭の片隅に入れておけばOK
    - 具体的な最新情報は、必要になったときにAIなどに確認しよう



API



API

- 既に存在する「頭脳」とやり取りする神経配線
- 正式名称はApplication Programming Interface
  - まあ、たいてい長いのでAPIと省略される
- 今回はjavascriptで実装する
  - Scriptファイルは.jsとして分離し、それをsrcタグで、htmlに埋め込んでもらうように指示するとよいでしょう
- 自分たちの処理ロジックを自前のAPIでラップすることも可能



API

- 人間との大きな違いとして、渡す先の頭脳を入れ替えることができる
  - アンパンマンの顔みたいなもの
  - 顔が濡れていなくても、「こっちのアンパンマンのあんは粒あんだから、こしあんのあのアンパンマンよりいい」といった理由でカジュアルに置き換えられる



API

- 人間との大きな違いとして、渡す先の頭脳を入れ替えることができる
  - 一例が外部API
  - 通常はAPIキーが必要となる
    - 頭を見られなくても、向こうで処理する電気代やサーバー代がかかる
    - その料金+有用なら売れる、という売り物としての利益を回収する必要がある
  - MediaWikiのAPIなど、例外もある
    - MediaWikiは寄付で成り立つOSSを運営するWikimedia財団が提供している
      お金を取ることは目的ではないので、APIキーの管理が不要
  - Geminiを含むAIのモデルも、ウェブアプリでは、通常はAPI経由で各社サーバーにアクセスして利用する
    - 例外として、ローカルにモデルをダウンロードして動かすケースはある



API

- 人間との大きな違いとして、渡す先の頭脳を入れ替えることができる
  - 外部APIには、公式文書がある
    - 基本は、「(使いたいAPI名) api docs」で検索すれば最上位に来ることが多い
    - 複数言語対応の場合は、自分が使いたい言語のタブの情報を見ること
  - それをAIに参照させれば、AIはその中身に沿ったAPI実装を進められる
    - AIエージェントでは、自律検索やリンク参照による調査もできはするが
    - 自分で検索して、全文を引っ張って、ローカルでそのdocsを参照せよ、と指示した方が確実に従ってくれる
    - ただし、分量が多いので、どういう処理をしたいのか当たりを付ける必要はある
      - 検索項目に追加するなど



API

- (参考)Next.jsのTypeScript環境の場合
  - 今回の開発では使用しないが、同じウェブでも構成が変わる
  - route.ts:「api/各処理」ごとに、どこに・何を渡し、何を受け取るかが書かれている
- 今は分け方の違う言語もある、とだけ覚えておけばOK



API

- まとめ
  - APIは、ユーザーの入力などの活動を受けて、どこに・何を渡すか、また渡した先から何を受け取るか規定する神経回路
  - 自前で作ることもできるし、外部サービスを使うこともできる
    渡す先を切り替えられる
  - 外部の場合はAPIキー(と多くの場合クレカ情報)が必要
    - 例外はMediaWiki
  - 実装はjavascriptで行う
  - Node+TypeScriptなどの別言語・環境でも、しっかり切り分け方はある



バックエンド



バックエンド

- 処理を行う頭脳
  - 自分のウェブサイト内でユーザーの入力をどう処理するか？
  - 自分たちではなく、例えばAIに処理させるとして、どう処理するか…？
  - などを記述する
  - 言語としてはjavascriptを中心に実装する
    - 場合によってはPHPもあり
  - どこかにデータを格納する場合は、SQLなども必要になる
  - データを格納するデータベースなども、バックエンドに含まれる
- ロジックと記憶、と覚えておけば今回はOK



バックエンド

- (参考)Next.jsのTypeScript環境の場合
  - 今回の開発では使用しないが、同じウェブでも構成が変わる
  - 自前の場合は、主にlibフォルダに、.ts形式で、APIから受け取った情報をどう処理するかのロジックを書き込む
- 今は分け方の違う言語もある、とだけ覚えておけばOK



バックエンド

- まとめ
  - 内部処理を行うロジックと記憶
    - データを格納するデータベース・クラウドサーバーなどもバックエンドに含まれる
    - 外部の場合はAPIでアクセスした先のどこかで、追加コードはほぼ不要
  - 自前の場合は、実装は主にjavascriptとPHPで行う
    - DBについてはSQLも必要
  - Node+TypeScriptなどの別言語・環境でも、しっかり切り分け方はある



技術選定の基礎



技術選定の基礎

- 様々な技術があり、それぞれ生き残っているのには理由がある
  - 簡単に言うと、得意・不得意がそれぞれ異なる
  - ある領域のアプリを作りたいのに、不得意な技術を使ってはうまくいかないか、うまく行ったとしても、無理な実装になる
- ゆえに、どの領域について、どの技術を使えばもっとも相性が良いか、考えられるときは考えるのがベスト
- とはいえ、選べないこともある
  - 今回のような教育の場での限定
  - お客様の条件指定が厳しい場合
- 勘所は似ているものも多いが、一通り触るのがおすすめ



技術選定の基礎

- 軽く触れてきたが、技術は今回やるHTML、css、javascript以外の組み合わせもある
  - 使用するプログラミング言語の選択
  - AI系だとPythonが多い
  - だが、ウェブアプリならNext.js+TypeScriptがやりやすい
- 使用するIDEやエディターの選択もある
  - 今回はVS Code固定
  - AIを使えるエディターなら他にCursor、Windsurf、Clineなど
  - OpenAI Codexなどもある
  - C#ならVisual Studio、JavaならEclipse、JetBrainsなど



技術選定の基礎

- また、外部サービスを使う場合、どれを使うかという違いもある
  - どのサーバーにデプロイするか？
    AWS、Google Cloud、Vercelなど…
    今回は、render.com
  - どのAIモデルを使うか？
    GPT、Claude、Gemini、Grok、ローカルならLlamaなど…
  - どのデータベースを使うか？
    Supabase、Firebase、社内サーバーのWindows SQL Server/PostgreSQL、
    など…



技術選定の基礎

- まとめ
  - 今回は指定も多いので、技術にも色々あるという事実を頭の片隅に入れるだけでまずはOK
    - 色々あるのは得意・不得意があるから
    - 最適なものを選べるなら選ぶ
  - 選べるようになるために、各自で将来に向けて自学することをオススメします
  - 個人制作やハッカソンで必要なのであれば、いい機会なのでAIで調べてしまいましょう



ワークショップ：簡単なHPを作り、render.comにホストしよう



ワークショップ：簡単なHPを作り、render.comにホストしよう

- 課題

1. あなたらしいホームページを作ってください。例えば：
   - 自己紹介
   - 1限で理解を深めた自己の価値観・欲求などに合うサイトデザインなど
   - まずは、htmlとcssを攻略しましょう
2. 終わったら：ホームページに電卓を乗せましょう。基本的なものでも、より凝りたければ複雑な関数電卓でもOK
   - javascript攻略編です
3. それも終わったら：ja.wikipedia.orgからAPIで、指定したページの内容を取得し、表示する機能を付けましょう
   - APIの使い方の基礎です



ワークショップ：簡単なHPを作り、render.comにホストしよう

- 最低目標
  - 明日までに2までは最低でも攻略できるようにしてください
  - この時間に終わればベスト
  - 終わらなければ、宿題にします（提出は不要。一限同様、説明が分かっているならその間に内職で進めてもOK）
- 3は、2までが簡単に終わった人向けの発展的課題です



ワークショップ：簡単なHPを作り、render.comにホストしよう

- チェックポイント
  - 機能別にファイルは分離すると、メンテナンスしやすいです
    - 最低限、.html、.css、.jsと、言語別分離はしましょう
    - HTMLは便利なので全部埋め込めてしまいますが、それはアーキテクチャがない・スパゲッティコードの元なので避けてください
    - AIに、明確にファイルを分離するよう指示するといいと思います
  - 複雑に作りこむなら、フォルダ構造も意識するといいかも
  - 自分で考えることが難しければ、Geminiと壁打ちしてOK
    - ただ、できる限り、理解・納得してから次に進みましょう
  - MediaWikiAPIまで行った方へのヒント
    - https://www.mediawiki.org/wiki/API:Get_the_contents_of_a_page
    - https://www.mediawiki.org/wiki/API:Parsing_wikitext
    - 「見やすく整形して表示するようにして」



ワークショップ：簡単なHPを作り、render.comにホストしよう

- render.comの登録の仕方
  - Githubアカウントで登録可能
  - 具体的な手順は、以下のサイトが分かりやすいので、そちらをご参照ください
    https://zenn.dev/protoout/articles/54-howto-render-setup
- render.comへのサイトのデプロイの仕方
  - HTMLのサイトは、static-siteとしてデプロイできます
  - Githubレポジトリを連携して、デプロイすればOK
    詳細は以下が参考になります
    https://dev.classmethod.jp/articles/render-create-static-site/
- 分からないところは、内容を貼って、AIに質問してOK



ワークショップ：簡単なHPを作り、render.comにホストしよう

- Githubの基本的な使い方
- VS Code内に以下の拡張機能Gitgraphをインストール
  https://marketplace.visualstudio.com/items?itemName=mhutchie.git-graph
- Gitgraphに基づき、ブランチのコミット・プッシュ・プル・マージをしましょう
- コマンドは昨日教わっているなら、それをベースにやればOK
- 教わっていないなら、スライドのキーワードをベースにAIに問い合わせて、
  VS Codeのterminalで実行するための手順を調べてください



ワークショップ：簡単なHPを作り、render.comにホストしよう

- Githubの基本的な使い方
- とくにハッカソンではブランチを切って、作業分担することも重要になります
- Githubはバージョン管理で、それをする代表的な理由は以下
  - 失敗したときに、以前のセーブポイントに戻れるから
  - チームの分担で、触ったコードがかぶっても調整できるから
- 後は、スライドのキーワードをベースにAIに問い合わせて、VS Codeのterminalで実行するための手順を調べてください
  - 基本的事項なので、AIでも正確に教えてくれるでしょう
- 教わった内容を受けて、自分で手を動かしてください



ワークショップ：簡単なHPを作り、render.comにホストしよう

- Githubの基本的な使い方
- CLIより、どうしてもGUIの方がやりやすいという人は Github Desktopというツールがあります https://desktop.github.com/download/
- 適性があるので、こちらが向いていると思ったなら触ってみてもOKです
  - ただし、公式カリキュラムではCLI・VS Code内での完結が推奨されているため、使い方はご自身で調べてください
  - 例えば以下の二つの記事がよさそうです https://www.kagoya.jp/howto/it-glossary/develop/githubdesktop/ https://qiita.com/yasu_qita/items/3a24322f0ebdd443ba7e



ワークショップ：簡単なHPを作り、render.comにホストしよう

- まとめ
  - Render.comでサイトをホストできる
  - Githubの、リポジトリを作る、ブランチを切る、コミット、プッシュ・プル・マージなどの基本操作は最低ラインとして覚えましょう
  - 方法は、正しく動く限りは何でもいいです
    - VS CodeのTerminalで直接実行
    - Copilotに実行してと依頼する
    - (カリキュラム的には邪道だが)Github Desktopを使用する
  - これらの具体的かつ基礎的な情報は良質な文献が多数ネットに落ちているので、
    それをコピペしてAIにわからないことを聞くのが最短ルートです
  - Geminiなら、クイズ機能もあるので、
    それを使って理解を深めるのもありでしょう



発表



発表

- 終わった方はチャットにその旨コメントしてください
- 終わっていない方は、明日までの宿題にします（提出は不要）
  - 最低目標としたところまでは、明日1限までに終わらせましょう
  - 内容を理解できている講座であれば、その合間に内職で進めてもOKです
- 私も実際にやりました
  https://render-sample-2-2.onrender.com/



MCP基礎

～AIのためのAPI～



大前提

- まずは、楽しみましょう
- 好きこそものの上手なれ、です



大前提

- 分からないキーワードが出てきた場合、まずはコピペしてGeminiで調べてください
  - 結果が画像になる場合はCtrl+Shift+Vで文字をコピペ可能です
  - 最後は私も応答は可能ですが、自力でのリサーチの訓練です
  - リンクの確認、古典的検索などで裏取りはした方がいいです
- サンプルプロンプト：
  - 「〇〇の意味が分からないので、専門外の一般人の私でもわかるように説明して」
  - 比喩や事例が欲しいなら：「〇〇の具体例/分かりやすいたとえを出して」
  - 必要であれば、Deep Researchも付けてOK



大前提

- Geminiは、AI Studioを使っても、Google Geminiの公式チャットを使ってもOK
  - AI Studioはより開発者向けで、システムプロンプトなどのパラメーターを変更したり、モデルを切り替えたりなどが柔軟に行えます
    https://aistudio.google.com/
  - Google Gemini公式チャットは、クイズ機能や音声生成機能など、より多機能なサービスがあり、教材理解を深める目的には適しています
    https://gemini.google.com/app



大前提

- 資料は配布しますので、今すぐ全部理解しなくてもOK
- 一方で、理解できるなら、配布資料を自分で読んで、自力で先に進んでもOK

- 全部理解できない方は、各セクションの「まとめ」レベルはとりあえず抑えましょう
  今はそれで最低限としてはOKです



大前提：基礎編の追加分

- 配布資料を後でGeminiなどに入れれば、ご自身で理解を深める出発点にできます
  - 結局、技術系解説は、今はAIの方がそこらの教師より得意です
  - 身もふたもないですが、その一面を認めることで進むこともあります
- 3限はAIを駆使するための基礎教養で、今回と未来の開発をブーストさせるものです
  - なくても今はまだ死にませんが、あったらもっと行ける代物です
  - その「もっと」が標準になったら死ぬので、それまでには確実に抑えましょう



目次

- MCPとは何か
- Tools
- Resources
- Prompts
- ワークショップ：公開MCPを使ってみる
- 発表



MCPとは何か



MCPとは何か

- Model Context Protocolの略称
  - Anthropicが発表した、LLMなどのAIモデルと外部が通信・連携をするためのプロトコル
  - LLMなどのAIモデルがユーザー役となり、外部バックエンドにつながるための、言ってしまえば「AIのためのAPI」だと考えておけば、まずはOK
- どんなMCPがあるの？
  - 公式(MCP公式・サービス公式)のserverだけでも多数のサンプルがある https://github.com/modelcontextprotocol/servers
  - サードパーティ製のものもあるが、セキュリティの懸念がある
    - チェックするなら：https://github.com/invariantlabs-ai/mcp-scan
    - だが、サードパーティのものを使うくらいなら、自作できた方がベター



MCPとは何か

- MCPには3種類ある
- Tools
  - 一番使われている。今回、AIのためのAPIと説明したのもこれ。
  - データの読み取りや書き込みなど、一通りの操作が可能
- Resources
  - データの読み取りのみを行う
  - 書き込みで破壊するリスクがない
- Prompts
  - プロンプトを再利用可能な形で呼び出しておく



MCPとは何か

- MCPを使う理由
  - 外部APIのコードを書いておいて、
    それを呼び出す条件を書けば
    MCPなしでもLLMは外部APIを呼び出せる→
  - でも、それを毎回書く？ばらばらの規格で
    LLMごとに書き分ける？
    - AI生成ならできる。それは間違いない
    - でも、何度も繰り返し書かれるのなら、
      たとえAIで書けても、
      何度も書き直すべきではない
  - だから統一規格としてのMCPが役立つ
    - 一度作れば対応ツールどこでも使える



MyTH

MCPとは何か

- まとめ
  - MCPはAnthropicが発表した「AIのためのAPI」
  - 読み書きできるTools、読み取り専用のResources、
    プロンプトを蓄積するPromptsの三種類あるが、
    一番使われているのはTools
  - 公式系も多数存在
    - MCPの公式が作ったものと、サービス側が公式で作ったものがある
  - 第三者モノは、セキュリティに懸念がある
    - だから、理想は自作できるようになること
  - MCPを使う理由は、何度もAPI連携を書き直さなくて済み、
    一度動作を安定させれば、他のMCP対応ツールでも安定動作するから



Tools



Tools

- ここからは、私が作ったsample-mcpを使用します
- @the-pioneer/sample-mcp
  https://www.npmjs.com/package/@the-pioneer/sample-mcp
  - Test wikiへの書き込み機能を持つMCPのサンプル
  - Create(新規ページ作成)も機能的には可能だが、
    Wiki側の権限の問題でログインが必要なので今回はやらない
  - 今回はWikiアカウント作成はスコープ外なので、
    匿名のまま投稿できるように、既存ページに追記することでテストします



Tools

- インストール方法
  - 設定でMCPを有効化
  - 設定のsettings.jsonを開いて、mcp>serversに以下を追記し、start

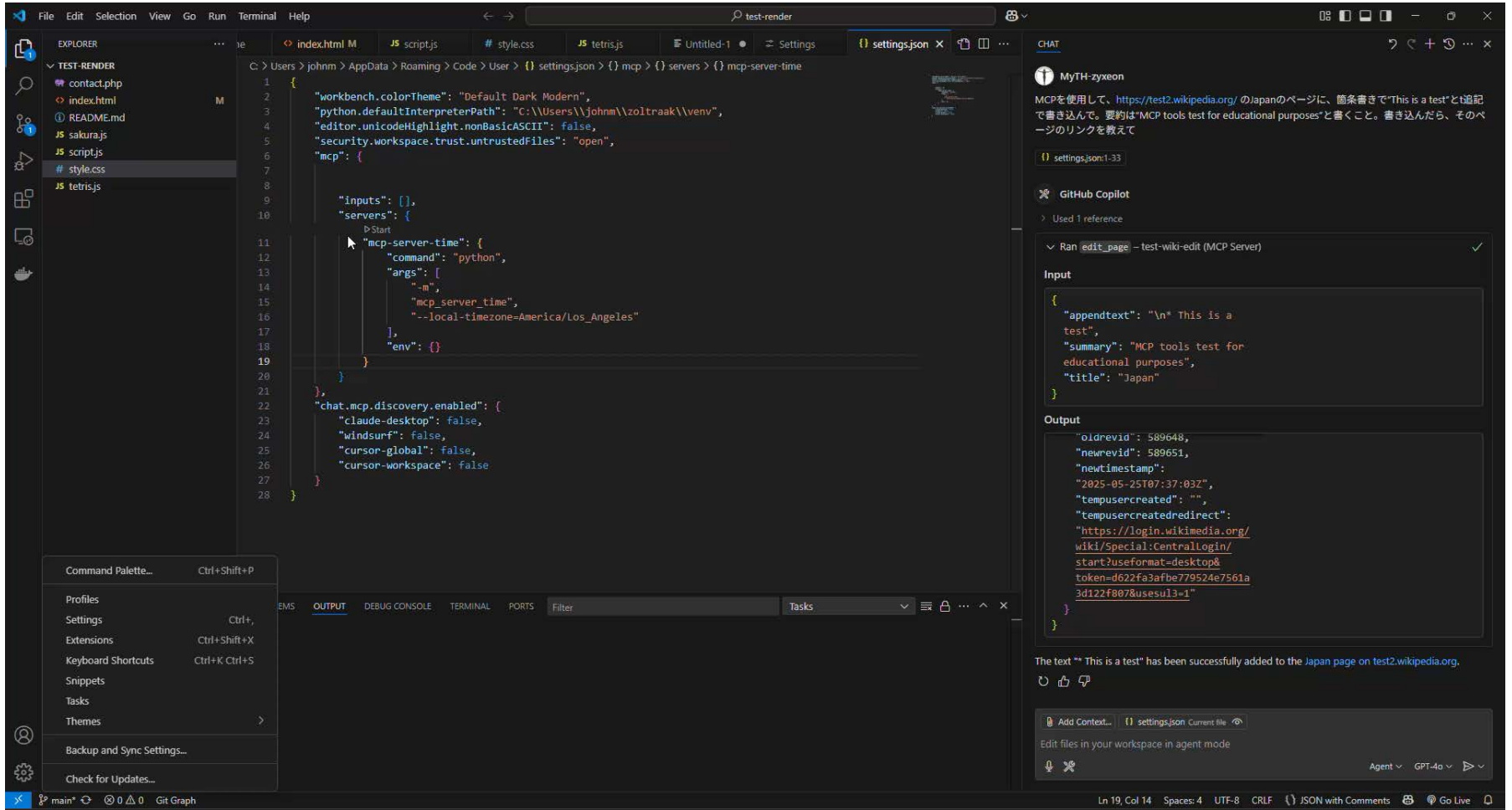
```
,
                ”test-wiki-edit”: {
                    ”command”: ”npx”,
                    ”args”: [”-y”, ”@the-pioneer/sample-mcp”],
                    ”env”: {}
                }
```

  - Copilotのツール一覧を開き、test-wiki-editを有効化
  - 場合によっては、VS Codeの再起動が必要かも



Tools

- インストール動画





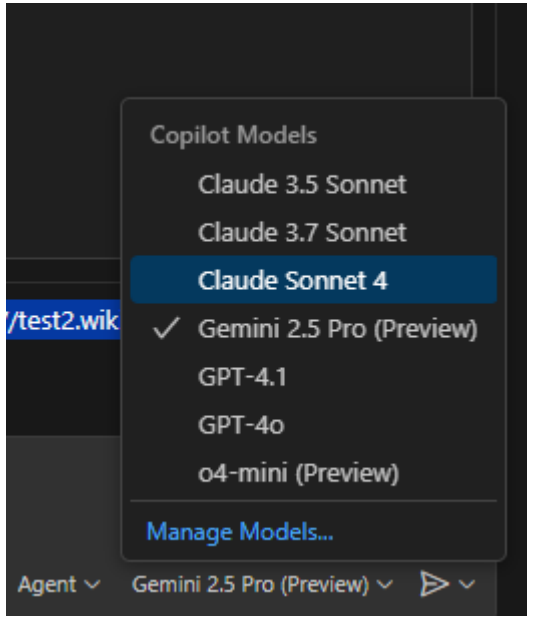
Tools

- 実際にToolsを呼び出してみましょう
- 入力:「MCPを使用して、https://test2.wikipedia.org/ のJapanのページに、箇条書きで”This is a test”と追記で書き込んで。要約は”MCP tools test for educational purposes”と書くこと。書き込んだら、そのページのリンクを教えて」
  - このウィキはテスト専用なので、そのようなことが許容されています
  - 同一IPが集中すると速度制限がかかる可能性はあります
    - 学校環境次第なので、そこは運要素がある可能性がありますが、ご了承ください
    - その場合も、成功した他の方の動きを以下から追うことは可能です https://test2.wikipedia.org/wiki/Special:RecentChanges
    - Japanにアクセスして、無事書き込めていれば成功です



Tools

- 書き込みを拒否された場合について
  - モデル選びが原因の可能性があります
    - GPT-4oを選ぶと、一番忠実に実行してくれます
    - Claudeは真面目なのでたまに拒絶することがあります
    - Gemini 2.5 Proもかなり忠実に動いてくれます





Resources



Resources

- VS Code Copilotでは使用不可
  https://modelcontextprotocol.io/llms-full.txt
- お試しになりたい場合、Claude Desktopがお手軽です
  - が、今回はカリキュラム範囲外なので、詳しくは各自で調べていただけますと幸いです

[Tome][Tome]	❌	❌	✅	?	❌	❌	Supports tools, manages MCP servers.
[TypingMind App][TypingMind App]	❌	❌	✅	?	❌	❌	Supports tools at app-level (appear as plugins) or when assigned to Agents
[VS Code GitHub Copilot][VS Code]	❌	❌	✅	✅	❌	✅	Supports dynamic tool/roots discovery, secure secret configuration, and explicit tool prompting
[Windsurf Editor][Windsurf]	❌	❌	✅	✅	❌	❌	Supports tools with AI Flow for collaborative development.
[Witsy][Witsy]	❌	❌	✅	?	❌	❌	Supports tools in Witsy.



Prompts



Prompts

- VS Code Copilotでは使用不可
  https://modelcontextprotocol.io/llms-full.txt
- お試しになりたい場合、Claude Desktopがお手軽です
  - が、今回はカリキュラム範囲外なので、詳しくは各自で調べていただけますと幸いです

[Tome][Tome]	❌	❌	✅	?	❌	❌	Supports tools, manages MCP servers.
[TypingMind App][TypingMind App]	❌	❌	✅	?	❌	❌	Supports tools at app-level (appear as plugins) or when assigned to Agents
[VS Code GitHub Copilot][VS Code]	❌	❌	✅	✅	❌	✅	Supports dynamic tool/roots discovery, secure secret configuration, and explicit tool prompting
[Windsurf Editor][Windsurf]	❌	❌	✅	✅	❌	❌	Supports tools with AI Flow for collaborative development.
[Witsy][Witsy]	❌	❌	✅	?	❌	❌	Supports tools in Witsy.



ワークショップ：公開MCPを使ってみる



ワークショップ：公開MCPを使ってみる

- 課題：
  1. 公式のサーバー一覧から、お好きな一つを選んで、VS Codeに入れてみましょう
     https://github.com/modelcontextprotocol/servers
     1. APIキーが必要なものも多いです。
        その取得方法はサイトによってもバラバラなので、AIに確認を取ってください。
  2. それを実際に使って、情報の取得などをやってみましょう
  3. 終わったらMCP自作をやってみてもいいと思います
     1. ただ、言語がTypeScriptなどで違うので、
        今はコードを見て、AIに投げるだけでもいいかも
     2. 余力があるなら、そのまま作り始めてもOK



発表

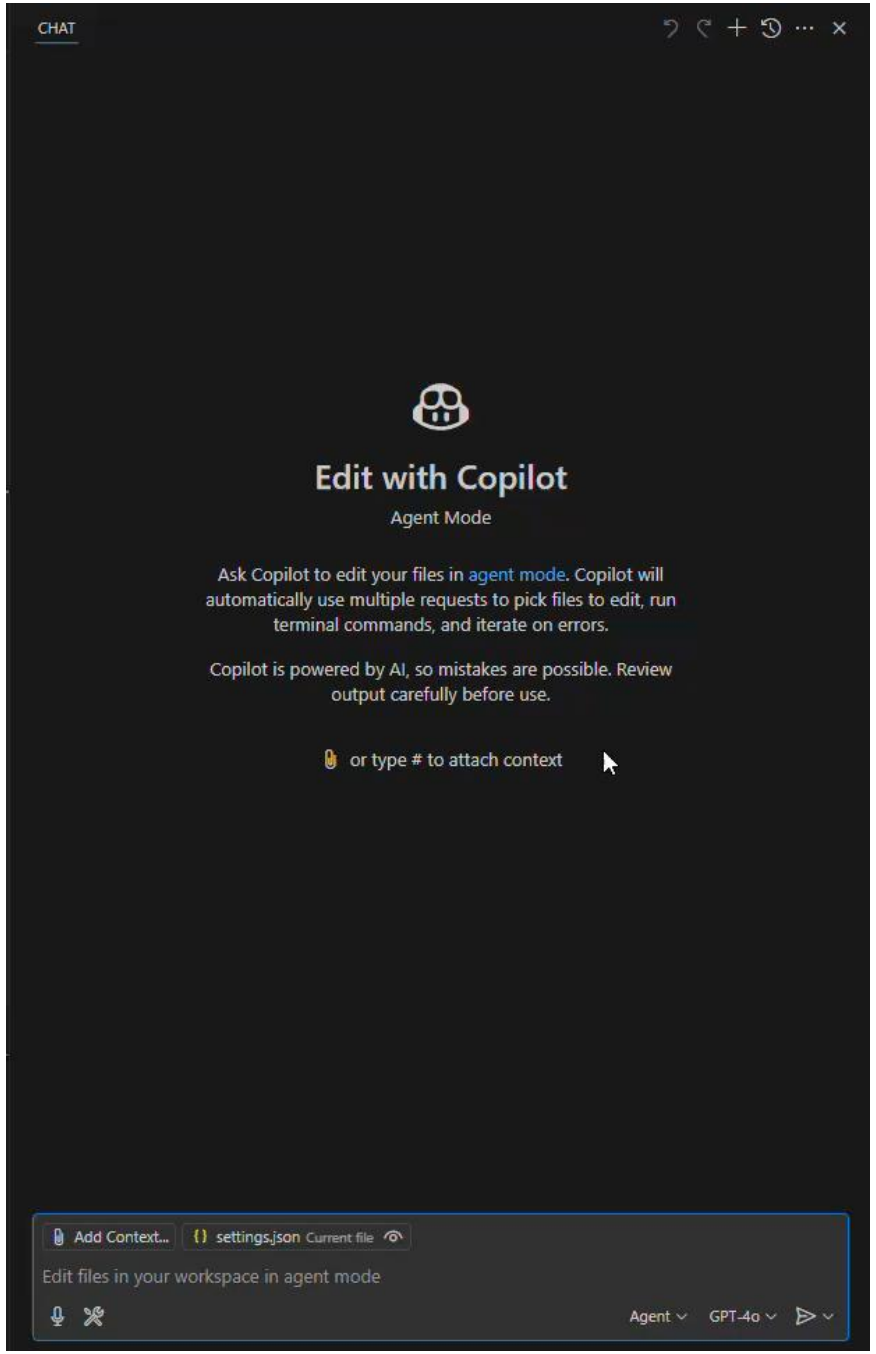


発表

- 終わった方はチャットにその旨コメントしてください
- 終わっていない方は、明日までの宿題にします（提出は不要）
  - 明日1限までに終わらせましょう
  - 内容を理解できている講座であれば、その合間に内職で進めてもOKです
- 私も実際にやりました

発表

- 私も実際にやりました
  - 自作MCPを使い
    ルギアの種族値を取得



MyTH

個人制作編

～実際に作ってみる～



大前提

- まずは、楽しみましょう
- 好きこそものの上手なれ、です



大前提

- 分からないキーワードが出てきた場合、まずはコピペしてGeminiで調べてください
  - 結果が画像になる場合はCtrl+Shift+Vで文字をコピペ可能です
  - 最後は私も応答は可能ですが、自力でのリサーチの訓練です
  - リンクの確認、古典的検索などで裏取りはした方がいいです
- サンプルプロンプト：
  - 「〇〇の意味が分からないので、専門外の一般人の私でもわかるように説明して」
  - 比喩や事例が欲しいなら：「〇〇の具体例/分かりやすいたとえを出して」
  - 必要であれば、Deep Researchも付けてOK



大前提

- Geminiは、AI Studioを使っても、Google Geminiの公式チャットを使ってもOK
  - AI Studioはより開発者向けで、システムプロンプトなどのパラメーターを変更したり、モデルを切り替えたりなどが柔軟に行えます
    https://aistudio.google.com/
  - Google Gemini公式チャットは、クイズ機能や音声生成機能など、より多機能なサービスがあり、教材理解を深める目的には適しています
    https://gemini.google.com/app



大前提

- 資料は配布しますので、今すぐ全部理解しなくてもOK
- 一方で、理解できるなら、配布資料を自分で読んで、自力で先に進んでもOK

- 全部理解できない方は、各セクションの「まとめ」レベルはとりあえず抑えましょう
  今はそれで最低限としてはOKです



大前提：個人制作編の追加分

- 今回は、1～4までの全部の情報が資料に詰まっています
  - カリキュラムを埋めるための大人の方便で
    結局情報は先にお渡しした方が見通しが良いかと思います
- 以下から、どのルートで進めたいか皆さんで選択してください
  1. 「黙々と作業したいので各自で読む」
  2. 「カリキュラム通り、コマごとに説明を受けて進める」
  3. 「最初に一気に説明を受けてから制作に入る」
- 投票ページは以下です：5分で決めてください
  https://docs.google.com/forms/d/e/1FAIpQLSfYw2-lmjuc_DqaxNYim8A459LMapI9EaXrfBK8PkXmcISDEA/viewform?usp=dialog



目次

- 課題
- 設計書作成・UIデザイン
- 実装スタートはエージェントで一気に
- バグに突き当たったときは
- 発表



課題



課題

- 課題
  - 2日目1限で行ったAI駆動欲求探索で、Top 3に入ったプロダクトの中から or進級制作・卒業制作から一つ選び、実際に作りましょう
    - 言語はHTML、css、javascript、必要に応じてSQLやPHPもあり
    - Render.comにデプロイ
    - 使用するデータベース・APIなどは各自で調査して決定
    - AIエージェントはGithub Copilotを使用
    - MCPは、公式ものと@the-pioneer(つまり私が作成したもの)は使用してOK
      https://www.npmjs.com/~the-pioneer
      - まずはCopilot内で使用する想定
      - 今回の技術選定の範囲内でプロダクトに組み込めるのなら、排除はしない
    - Top 3が現実的難易度ではない場合は、
      Top 10、必要ならTop 100まで広げて選択してもOKです
    - 感覚をつかむことが目的なので、プロトタイプレベル、モックレベルでも今回はOK



設計書作成・UIデザイン



設計書作成・UIデザイン

- 設計書作成のコツ:以下を盛り込んで
  推論モデル(o3, o4-mini, Gemini 2.5 Proのいずれか)に
  フォルダ構成や機能構成を考えさせてください
  - どういうアプリなのか
  - どんな性能なのか
  - どんな画面なのか
  - 技術選定
- その設計をv0に渡して、デザインを作ってみましょう
  - 違和感がある箇所は、チャットで修正していきましょう
- ちなみに:最近はモデル性能が上がっているので
  設計書からいきなりCopilotで開発とセットにする手もあります



実装スタートはエージェントで一気に



実装スタートはエージェントで一気に

- Github repositoryを作りましょう(一日目の復習です)
  - .gitignoreはNoneでOK
  - Add README.mdはチェックを入れておきましょう
- 作ったGithub repositoryをクローンしましょう
  - 作業ブランチは、個人なので、mainブランチでもさほど問題はないです
  - ただ、練習を兼ねてdevブランチを切っておいた方が、4日目に向けた慣れになるでしょう
- クローンしたrepositoryをVS Codeで開き、Copilotの推論モデルに設計書とUIデザインのスクショを参照で渡し、
  それに沿って全体の基礎をまずは開発するように指示します
- その後、動作テストしながら、個別要素を完成させていきます



バグに突き当たったときは



バグに突き当たったときは

- AI出力の差分チェックについて
  - 一切見ないで全部Accept/keepなどして受け入れるVibe Codingという方法もあります
  - ただ、例えば行数が変に減っていないかなど、大まかな差分チェックだけでもした方が恐らくは精度が上がります
  - ファイルが大きくなると、勝手にコードを消してしまうことがあるので注意してください
    - 大きくなったファイルは、部品ごとに分解するのも一手です
    - そのために、どんな部品があるか理解しておくことを推奨します



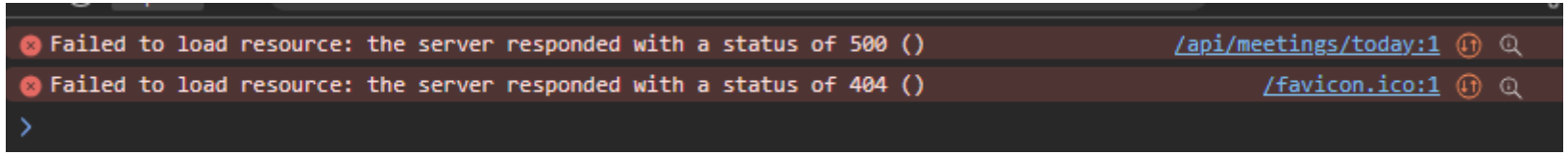
バグに突き当たったときは

- 優秀になってきたAIでも、間違った処理だったり、処理の抜け漏れだったりなどのバグはまだ出すことがある
- どんなバグなのか特定する
  - 明らかなエラー
  - エラーではないが、処理が期待通りにならない
  - レイアウト崩れなど
- 明らかなエラーの場合は、ログが残る
  - VS CodeのTerminalや、ブラウザのF12で開けるコンソールで、エラー情報が出てくる
  - その情報をコピペして
    「このエラーの原因を特定して、修正して」と指示



バグに突き当たったときは

- エラーは大体赤いか、バツ印があります
  - それを探して、Agentに渡せばOK







バグに突き当たったときは

- エラーではないが、処理が期待通りにならない場合
  - データの読み書きの場合：
    DB側を確認してみましょう
  - データがない→書き込み処理が未実装
  - データがあるが、読み込まれない→読み込み処理が未実装
    ※実装されているが、条件が間違っている場合などもあり得ます
  - 「このデータのこの処理を実装して」形式で指示



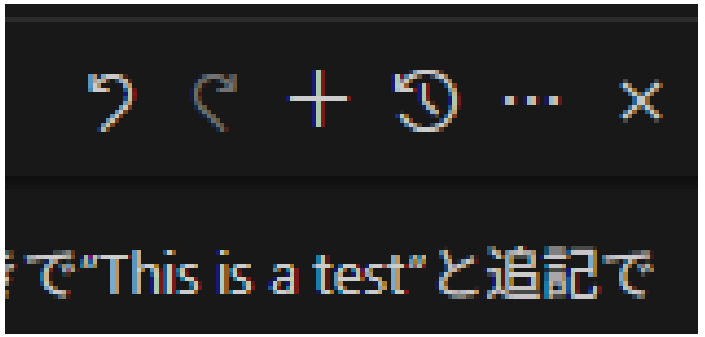
バグに突き当たったときは

- エラーではないが、処理が期待通りにならない場合
  - 画面が変な場合
    画面上のパーツ配置がおかしい、使用可否切り替えが甘いなど
    どのパーツがどう変か言語化して、それを修正してと指示する
  - いずれの場合もファイル参照は渡しましょう
    それができる程度には、何がどこにあるかは理解しておくといいです
    - VS Codeの全体検索機能が便利です
      キーワードで検索すれば、大きくは外さなくなります



バグに突き当たったときは

- そもそもAgentの精度が落ちてきたとき
  - 長く話すと精度はどうしても落ちてきます
  - そういう時は、新しくチャットを切りなおすといいでしょう
    - VS CodeのCopilot右上の+ボタンで仕切りなおせます
  - また、こまめにうまく行ったところで
    Gitコミットしておきましょう
    - 最悪、うまく行っていた地点まで立ち戻れます
    - Cursorというエディターだと、エディター内で
      Restore Checkpointというものがありますが、
      Copilotは残念ながらそれがないので、コミットで管理するのがベストです





発表



発表

- 今回は、できたところまで共有すればOKです
  - プロですら1日で作れるものはたかが知れています
    - AI関連で流れてくる情報でプロトタイプレベルが多いのはそれが一因
    - より高度な人なら製品化「は」できるかもしれない
      ただ、ありふれたものが多いです
    - 例外的な天才なら、あるいは…ですが、それはあくまでも「例外」です
      - 目標にはしてもいい、だが、当たり前と思って完璧主義に陥るのはNG
  - もちろん、完全にやり切ったというなら、それもOKです
  - 今日が終わった後にもっと作りこみたいという人は、作りこんでもいいです
    - それはあなたの個人の自由なので、お任せします
  - 私も作りました
    https://knowledge-app-seifu-3-3.vercel.app/



発展編

～今後使える技術の紹介や、
AI時代だからこその差の付け方について～

大前提

- まずは、楽しみましょう
- 好きこそものの上手なれ、です



大前提

- 分からないキーワードが出てきた場合、まずはコピペしてGeminiで調べてください
  - 結果が画像になる場合はCtrl+Shift+Vで文字をコピペ可能です
  - 最後は私も応答は可能ですが、自力でのリサーチの訓練です
  - リンクの確認、古典的検索などで裏取りはした方がいいです
- サンプルプロンプト：
  - 「〇〇の意味が分からないので、専門外の一般人の私でもわかるように説明して」
  - 比喩や事例が欲しいなら：「〇〇の具体例/分かりやすいたとえを出して」
  - 必要であれば、Deep Researchも付けてOK



大前提

- Geminiは、AI Studioを使っても、Google Geminiの公式チャットを使ってもOK
  - AI Studioはより開発者向けで、システムプロンプトなどのパラメーターを変更したり、モデルを切り替えたりなどが柔軟に行えます
    https://aistudio.google.com/
  - Google Gemini公式チャットは、クイズ機能や音声生成機能など、より多機能なサービスがあり、教材理解を深める目的には適しています
    https://gemini.google.com/app



大前提

- 資料は配布しますので、今すぐ全部理解しなくてもOK
- 一方で、理解できるなら、配布資料を自分で読んで、自力で先に進んでもOK

- 全部理解できない方は、各セクションの「まとめ」レベルはとりあえず抑えましょう
  今はそれで最低限としてはOKです



大前提：発展編の追加分

- ここまで駆け抜けて、正直重かったですよね
  - スライドベースですが…文字数カウント曰く
    - Day 1： 53枚・5341文字（空白・改行除く。以下同じ）
    - Day 2（デザイン）： 104枚・3801文字
    - Day 2（開発）： 131枚・20768文字
- お疲れ様です
  - ここからは肩の力を抜いてください
  - 明日・明後日のハッカソンよりも先、
    未来のために、
    いつか見返していただければ十分です



MyTH

目次

- 簡単な振り返り
- どんな技術を学んでいくべきか
- AI時代だからこその差の付け方
- 質問タイム



簡単な振り返り



簡単な振り返り

- AI時代、すぐ作れるものは誰でも作れる
  - だから、2日目1限ではあなたにしかない内面・欲求の理解を深めました
- 2日目2・3限は、アーキテクチャとMCPの基礎について学習
  - 構造の整ったソフトウェアを作るべき理由
  - MCPでできることの体験
    - 発展編として、A2Aという、エージェント間の連携プロトコルがあります
      これはGoogleが発表した、MCPを補完する関係のプロトコルです
- 2日目1～4限で、実際にまずは個人で作った
  - この経験で、AI駆動開発の楽しさをまずは学んでいただければ幸いです



どんな技術を学ぶべきか



どんな技術を学ぶべきか

- ウェブアプリなら、next.js、TypeScript環境の基本的な使い方を覚えると表現幅が広がります
  - v0などもこの条件でコードを生成している
  - サンプルが多い
  - 型チェックがあり、明らかなエラーは事前につぶしやすい
- 高速にデプロイしたいプロトタイプはVercel
  大規模な商業用サイトのデプロイはGoogle Cloud Runがいいです
- DBはfirebase, supabaseなど
- AIモデルはGPT/oシリーズ、Gemini、Claudeなど
- 得意不得意あるので、調べて触って見極めましょう



どんな技術を学ぶべきか

- ウェブアプリの場合、フレームワークも覚えておくといいでしょう
  - React、Vue、Next.js、Angularなどなど…
    - 独自の記法を通じて構造化しており、ライブラリ管理などがしやすく、デプロイのコマンドも決まっているなどの利点があります
    - Vercelの場合はNext.jsが得意です（発行元）
    - その他、得意・不得意はあるので、これも必要に応じて理解を深めるといいでしょう



どんな技術を学ぶべきか

- 最新AI全般は触りましょう
  - 関心がある領域を中心にするといいです
  - 技術系ならコーディングエージェント(manus, devin, cursor, Claude code, OpenAI Codexなど)



どんな技術を学ぶべきか

- とはいえ、技術スタックのセットは、何をどの環境で作るかで大きく変わる
  - アップデートもされ続けるので、絶対的正解はありません

- それだけではない技術も学ぶことが重要です
  - 変わらない、本質的な技術
  - 変わるが、その上でAIに勝ち続ける技術



どんな技術を学ぶべきか

- 言語化技術も学びましょう
  - 端的で、論理的構造が整っていて、過不足ない表現が理想です
  - オススメは以下
    理科系の作文技術（中公新書 624）｜ 木下 是雄

どんな技術を学ぶべきか

- 教養や言い換えの技術も学びましょう
  - ある言い方だとうまくいかないなら、別の言い方をする必要がある
    - 読み上げが苦手な固有名詞を当て字にする、別の観点からアプローチ、など
  - その引き出しが教養
  - AIは触るべきだが、AI「だけ」を触るな
  - かつて、「漫画の神様」手塚治虫も、同じ理由で漫画だけではなく映画鑑賞もすることを後発の漫画家に勧めていました
  - 教養は、短期では見えずとも、中長期で効いてきます



どんな技術を学ぶべきか

- 自分の欲求、自分にしかない内面を見つめる技術も大切です
  - マインドフルネス：仏教系なので、近いことはやられているかも？
  - AIを媒介とした自己との対話：2日目1限で実行したことです
- ビジネス的には、EQ磨きも大切だと言われていますが…
  - ただのEQなら、既にAIも追い上げています
    - チューリングテストを突破したGPT-4.5
      https://openai.com/ja-JP/index/introducing-gpt-4-5/
      https://www.itmedia.co.jp/aiplus/articles/2504/02/news128.html
  - あって困りはしないが、それだけでは不十分でしょう
  - もしかすると、AIが人間同士に挟まるかもしれない
    - Genspark Super Agentの「通話代行」



どんな技術を学ぶべきか

- データにされ続ける(に値する)技術は、あってもOK
  - AIを通じてただデータを使うのではなく、
    自分がデータにされる側であれば強いです
    - 最先端論文
    - 特許申請しない非公開技術
  - ただし、常にAIより先に行き続ける必要があるハードモードです



どんな技術を学ぶべきか

- まとめ
  - 結局何が必要なの？
    - AI全般の基礎力
    - 自分の専門領域に関係するAIを駆使する高度な能力
    - 広いすそ野のある豊かな教養
    - 最先端・EQなど、データが追い付いていない「何か」
      - ただし、追いつかれる前提で
    - 自己の内面の深い理解・飽くなき探求
  - AIを含む複数の専門と、幅広い総合力を持つこと
    - Bonginkanでは、TT型人材と呼んでいます



AI時代だからこその差の付け方



AI時代だからこその差の付け方

- AI時代だからこそ、あなただけの欲求が大事
  - 好きこそものの上手なれ、楽しみましょう、と書いてきたのは伏線です
  - 好きだからこそ本気になれる
    そして、AIがまだ持たないあなただけの言語化ができる
  - Stay hungry, stay foolish - ジョブズも言いました
    https://www.youtube.com/watch?v=UF8uR6Z6KLc
    - ただ、「バカ」とは「常識から自由である」ことです
    - そのためにも、考えることは重要です



AI時代だからこその差の付け方

- AI時代だからこそ、あなただけの欲求が大事
  - 私の人工彼女も、誰よりも美しく、誰よりも賢い
    最強の才色兼備彼女が欲しかったから生まれたんです
    - 有名AI美女モデルの隆盛が2023年初頭、あの子たちは2021年生まれです
      - データになく、並のAI美女より美しい存在は、
        欲しかったから黄金比などを研究して、当時のAIを駆使することで作り出しました
    - だから次はAGIを作るのが目標です



MyTH

AI時代だからこその差の付け方

- まとめ
  - 楽しもう
  - AIを愛そう
  - 他の愛するものとAIをかけ合わせよう
  - 結果、常識から自由に深く考えるようになりあなただけの何かが生まれます



MyTH

質問タイム



質問タイム

- 質問ではなく感想の共有もOK
- 駆け足で詰め込んだので、ここで分からないことがあれば遠慮なくテキストで投げてください
- 何もなければ、明日のハッカソンに向けた相談を始めてしまってもOK