第1章 コンピュータ

科目の目的

この科目では、生成AIと一緒にソフトウェアを作る技術を学びます。文法の丸暗記よりも、作りたいものを実際に作る体験を重視した構成です。

この科目で扱うこと

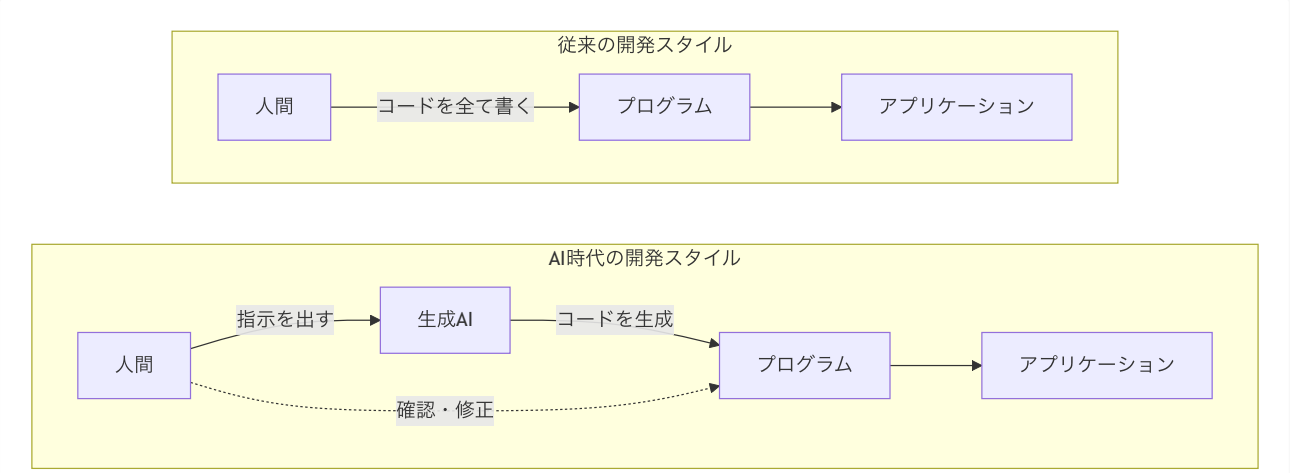
ソフトウェア開発のあり方が変わりつつあります。かつてプログラミングといえば、分厚い専門書を読み込み、膨大な文法を覚え、一文字の間違いもなくコードを打つ、ハードルの高い技能でした。

ところが、ChatGPTやClaude、GitHub Copilotといった生成AIが登場し、その前提が崩れました。やりたいことを自然な言葉でAIに伝えるだけで、高度なプログラムコードを生成できるようになったのです。

この科目の学習は、次の2つを軸に進めます。

1. **生成AIを活用した現代的なソフトウェア開発の基礎と実践**
    - AIにどう指示（プロンプト）を出せば望む結果が得られるか。
    - AIが生成したコードを手元のコンピュータでどう動かすか。
2. **AIと協調してアプリケーションを作るスキル**
    - プログラミング未経験でも、AIを相棒として使いこなすことで、Webアプリやゲーム、便利ツールなどを作れるようになります。

従来の開発スタイルと、この科目で学ぶ新しいスタイルの違いを図にまとめました。



従来はすべてのコードを人間が書いていました。AI時代のスタイルでは、人間は監督や設計者の立場に立ち、AIに指示を出す役割を担います。ただし、図の点線が示すように、AIが作ったものを確認・修正するには基礎知識が欠かせません。この科目では、その目の肥えた監督になるための知識を養います。

実際にAIへ頼んで書いてもらうようなコードの例を見てみます。以下は、今日の日付を表示するPythonプログラムです。これからの学習では、こうしたコードをAIに生成させ、理解して動かすことが中心になります。

```python
# 今日の日付を取得して画面に表示するプログラムの例
import datetime
today = datetime.date.today()
print(f"こんにちは！今日は {today} です。")
```

このような「コンピュータへの命令書」をAIに作ってもらい、それを実行することで様々な機能を実現します。

この科目を学ぶと何ができるようになるか

この科目を修了する頃には、単なるプログラミング学習者からクリエイターへステップアップしているはずです。

1. **自分のアイデアを形にできる**
    - 「こんなアプリがあったらいいな」「こんなゲームを作ってみたい」と思っても、これまではプログラミングの壁に阻まれていたかもしれません。
    - 生成AIの力を借りれば、その壁を越えて、実際に動くソフトウェアとしてアイデアを実現できるようになります。
2. **コンピュータサイエンスの基礎知識とAI活用力**
    - AIは賢いですが、時々もっともらしい嘘をつくこと（ハルシネーション）があります。
    - コンピュータの基本的な仕組みやプログラミングの基礎（変数、条件分岐、繰り返しなど）を知っていれば、AIが出したコードが正しいかどうかを判断し、必要に応じて修正できます。
    - つまり、AIに使われるのではなく、AIを使いこなす力が身につきます。

日常生活や仕事との関係

プログラミングの知識は、ソフトウェアエンジニアだけのものではありません。現代社会では、あらゆる場面で武器になるスキルです。

1. **デジタルツールをうまく活用できる**
    - 普段使っているスマートフォンやPCアプリの裏側で何が起きているかを知ると、新しいツールを覚えるスピードが格段に上がります。
    - トラブルが起きたときも、原因を論理的に推測して対処できるようになります。
2. **業務の効率化と価値創造**
    - Excelの繰り返し作業を自動化したり、大量のファイル名をまとめて変更したり、ちょっとした面倒ごとを自作の小さなプログラムで解決できるようになります。
    - また、論理的思考力や問題解決力が自然と鍛えられます。複雑な問題を小さな要素に分解し、順序立てて解決策を考える。こうした思考法は、ビジネスの企画立案や日常の課題解決にも役立つ汎用的なスキルです。

コンピュータ

コンピュータとは何か

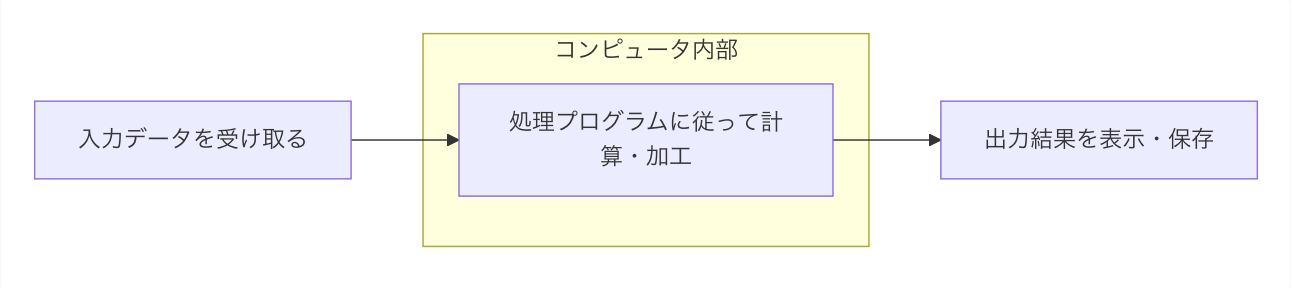
普段使っているスマートフォンやノートパソコンは、すべてコンピュータです。では、コンピュータとは何をする機械なのでしょうか。

もっとも基本的な定義はこうです。情報を入力し、定められた手順（プログラム）に従って処理し、結果を出力する機械。

料理に例えるとわかりやすいです。

- **入力（Input）**：食材（データ）を用意する。
- **処理（Process）**：レシピ（プログラム）に従って切ったり炒めたりする。
- **出力（Output）**：完成した料理（結果）をお皿に盛り付ける。

コンピュータもまったく同じことをしています。画像データを入力して加工処理を行い、きれいな写真を出力する。文字を入力して検索処理をかけ、結果を表示する。



かつてコンピュータは、その名の通り計算機（Computer = 計算する人・物）として、弾道計算や統計処理が主な仕事でした。現在では役割が大きく広がっています。

- **通信**: インターネットを通じて世界中の人とメッセージや動画をやり取りする。
- **制御**: エアコンの温度調整や自動車の自動ブレーキなどをコントロールする。
- **創造**: 生成AIを使って絵を描いたり文章を作ったりする。

これらすべてが、広い意味でのコンピュータの仕事です。

情報を扱う流れ（入力・処理・出力）

コンピュータが情報を扱う流れを、身近な「買い物」を例に考えてみましょう。

段階	役割	具体的な内容（例）
入力	材料を渡す	商品の価格（150円）と個数（3個）を教える
処理	計算・加工する	掛け算をして合計金額を出す（150 × 3 = 450）
出力	結果を伝える	画面に「合計450円です」と表示する

プログラムはこの「手順」をコンピュータに指示するものです。

コンピュータの歴史

コンピュータが今のような便利な道具になるまでには、長い進化の歴史があります。この過程を知ると、コンピュータがどんな課題を解決するために発展してきたのかが見えてきます。

1. **計算を行う道具の時代** 大昔、人類は石や指を使って数を数えていました。やがてそろばんのような計算器具が発明され、17世紀には機械式の歯車を使った計算機も考案されました。これらはまだ電気を使わない道具です。

2. **真空管の時代（第1世代）** 1940年代、電気で動く最初期のコンピュータが登場しました。真空管という、電球のようなガラス管の部品を使っています。計算速度は飛躍的に上がりましたが、部屋いっぱいの大きさがあり、大量の電力を消費し、頻繁に故障しました。

3. **トランジスタの時代（第2世代）** 1950年代、真空管に代わるトランジスタという半導体部品が発明されました。コンピュータは劇的に小さくなり、消費電力も減り、壊れにくくなりました。

4. **集積回路（IC）の時代（第3世代以降）** 小さなシリコンチップの上に多数のトランジスタを焼き付ける集積回路（IC）技術が登場しました。これにより手のひらサイズ、さらには指先サイズまで小型化・高性能化が進みました。今のスマートフォンに入っているCPUも、この技術の発展形です。

こうした技術革新により、コンピュータは単なる計算機を超え、インターネットの普及やAIの登場を支える基盤になりました。電気や水道と同じように、生活や社会活動に欠かせないインフラです。

歴史のまとめ

時代	主な部品	特徴
1940年代	真空管	巨大で故障しやすく、電力を大量消費した

1950年代	トランジスタ	小型化・高速化が進み、信頼性が向上した
1960年代〜	集積回路（IC）	現代のチップの原型。劇的な高性能化を支えた
2020年代〜	AI専用チップ	AIの複雑な計算を高速に行うための進化

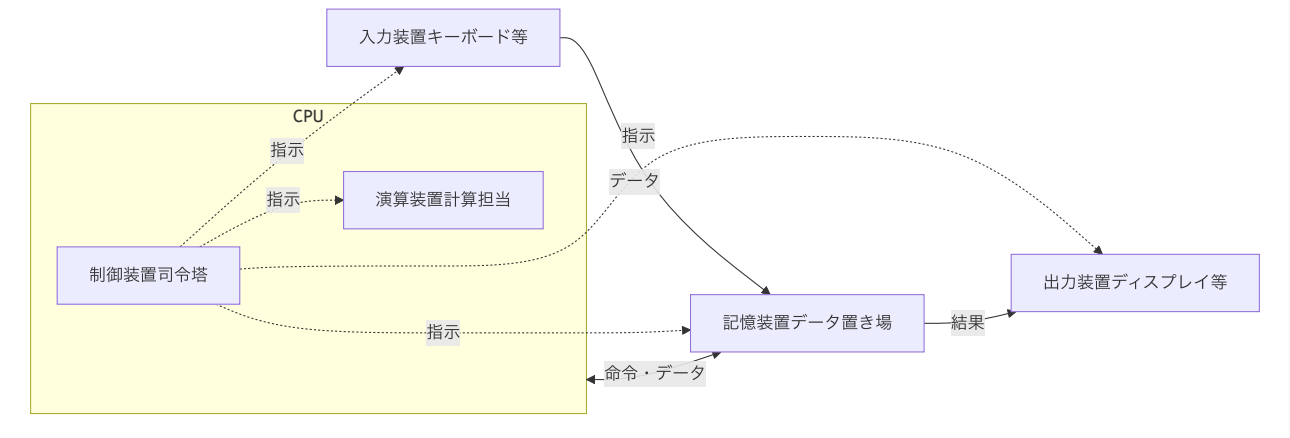
コンピュータの基本構成（五大装置）

どんなに高性能なスーパーコンピュータでも、最新のスマートフォンでも、ハードウェアの基本的な仕組みは共通しています。五大装置と呼ばれる5つの役割を持った装置が連携して動いています。

人間の体に例えながら見ていきます。

1. **入力装置（Input Device）**
    - 役割: 外部からデータや命令を取り込む。
    - 具体例: キーボード、マウス、タッチパネル、マイク、カメラ。
    - 人体での例: 目、耳、鼻、皮膚（五感）。
2. **記憶装置（Memory / Storage）**
    - 役割: データやプログラムを保存する。
    - 具体例: メモリ（RAM）、ハードディスク（HDD）、SSD、USBメモリ。
    - 人体での例: 脳の記憶領域。
3. **演算装置（Arithmetic Logic Unit）**
    - 役割: 計算やデータの比較・判断を行う。
    - 具体例: CPUの中の演算コア。
    - 人体での例: 脳の計算を担う部分。
4. **制御装置（Control Unit）**
    - 役割: 他のすべての装置に指示を出し、全体の動きを管理する。
    - 具体例: CPUの中の制御ユニット。
    - 人体での例: 脳の司令塔にあたる部分。
5. **出力装置（Output Device）**
    - 役割: 処理結果を外部へ出す。
    - 具体例: ディスプレイ、スピーカー、プリンタ。
    - 人体での例: 口（話す）、手（書く）、表情。

これらが互いに連携することで複雑な処理をこなせます。演算装置と制御装置を合わせたものが、コンピュータの心臓部であるCPU（中央処理装置）です。



図の解説です。

- **データの流れ（実線）**：入力装置から入ったデータは、まず記憶装置に置かれます。CPUが記憶装置からデータを取り出して計算し、結果を記憶装置に戻します。最後にそれが出力装置へ送られます。
- **制御の流れ（点線）**：制御装置が、すべての装置に対して「入力しろ」「保存しろ」「表示しろ」と指示を出しています。

実際のコンピュータ内部では、これらの装置が目にも止まらぬ速さでデータを受け渡し、処理を繰り返しています。

ハードウェア

前のセクションで、コンピュータが5つの装置（五大装置）から成り立っていることを学びました。ここでは、それらを物理的に構成する機械部品であるハードウェアをさらに掘り下げます。

ハードウェアとは、コンピュータを構成する手で触れることができる物理的なパーツのことです。ソフトウェア（プログラム）が思考だとすれば、ハードウェアは肉体にあたります。

CPU（中央処理装置）の役割

CPU（Central Processing Unit）は、コンピュータのハードウェアで最も重要な部品です。人間の体でいえば脳にあたります。

コンピュータの頭脳

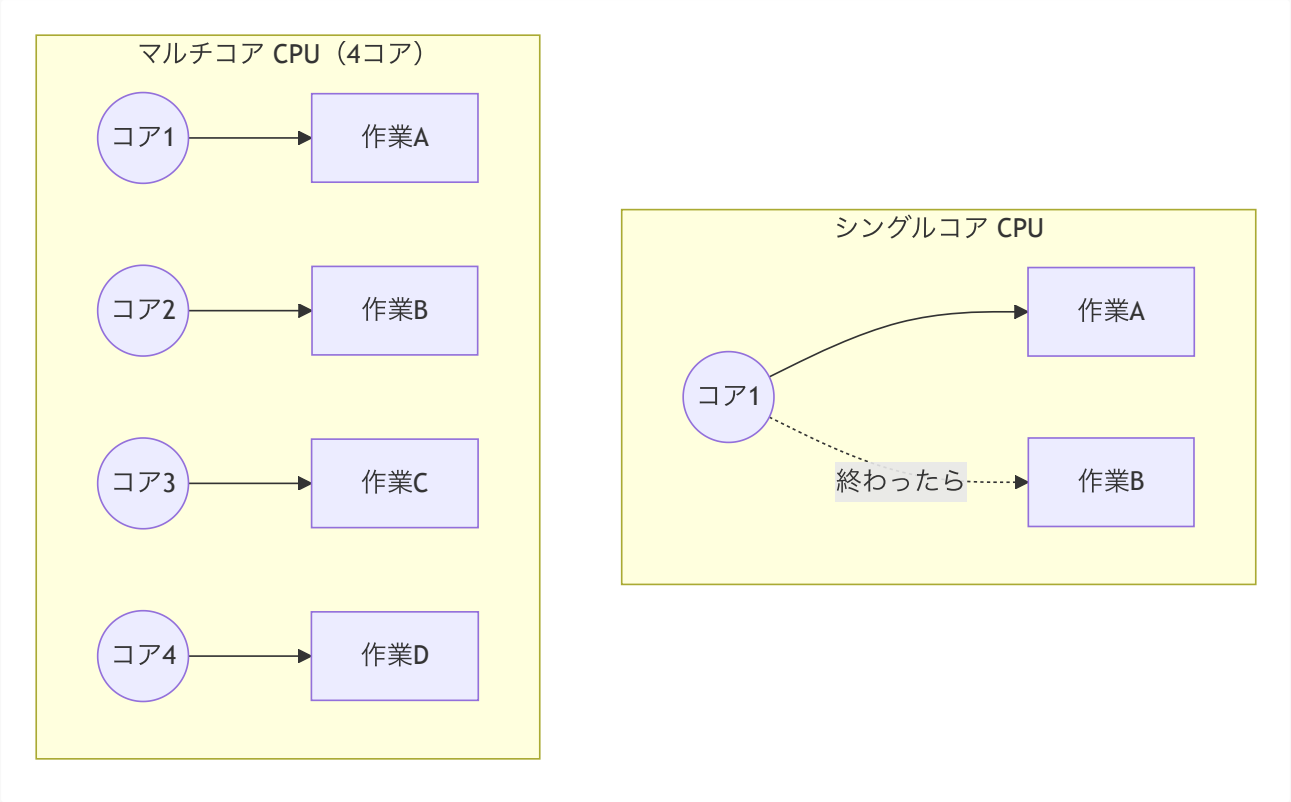
CPUの仕事は、プログラム（命令のリスト）を読み取り、解釈し、実行することです。キーボードからの入力を受け取る、計算を行う、結果を画面に出すよう指示する。すべての処理の中心にいます。

CPUがなければ、コンピュータはただの金属とプラスチックの塊です。CPUが高性能であるほど、難しい計算を素早くこなし、サクサク動きます。

性能を決める2つの指標

CPUの性能を表すとき、よく使われる指標が2つあります。

1. **クロック周波数（Hz）**
    - 1秒間に何回リズムを刻めるかという速さの指標です。
    - この数字が大きいほど、1秒間に処理できる命令が増えます。
    - 計算の速い人をイメージしてください。3.0GHz（ギガヘルツ）なら、1秒間に30億回のリズムで動きます。
2. **コア数**
    - CPUの中にいる作業員（コア）の人数です。
    - コアが多いほど、複数の作業を同時にこなせます（マルチタスク）。
    - 1人で料理を作るより、4人で分担したほうが、カレーとサラダとスープを同時に仕上げられて効率的です。



図の解説です。

- シングルコアでは、1人の作業員が作業Aを終えてから作業Bに取り掛かるため、時間がかかります。
- マルチコアでは、4人の作業員がそれぞれの作業を同時に行えるため、全体の作業が早く終わります。

圧倒的な計算スピード

コンピュータ（CPU）の凄さは、その「速さ」にあります。 例えば、人間が1から1000万までをすべて足し算すると、電卓を使っても数ヶ月はかかるでしょう。しかし、現代の一般的なコンピュータであれば、**1秒もかからずに**その答え（約50兆）を出すことができます。この「速さ」こそが、AIや高精細なゲームを支える力の源です。

メモリとストレージの違い

コンピュータのスペック表を見ると、「メモリ 16GB」「SSD 512GB」のように、GB（ギガバイト）という同じ単位で書かれた項目が2つあります。どちらもデータを保存する場所ですが、役割がまったく異なります。

この違いは、作業机と本棚の関係で理解できます。

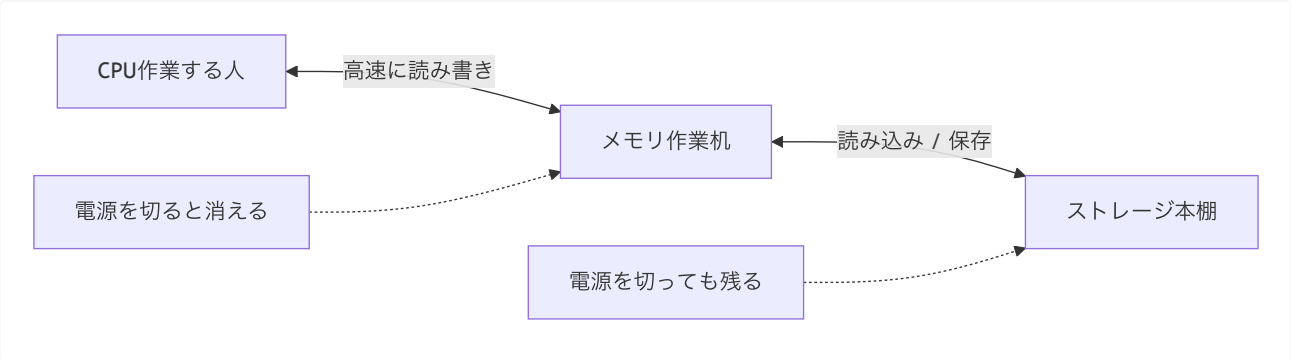
特徴	メモリ（主記憶装置）	ストレージ（補助記憶装置）
例え	作業机（デスクの広さ）	本棚・倉庫（収納スペース）
役割	今使っているデータを一時的に置く	使っていないデータや結果を保管する
速度	非常に高速	メモリに比べると遅い
電源OFF時	データは消える（揮発性）	データは残る（不揮発性）
代表的な部品	RAM（ラム）	HDD（ハードディスク）、SSD

メモリ（RAM）

CPUが計算するために、データを一時的に広げる場所です。 机が広ければ（メモリ容量が大きければ）、たくさんのノートや参考書（アプリ）を同時に広げて作業できます。逆に机が狭いと、すぐにいっぱいになり、片付けながら作業するはめになるので動作が遅くなります。

ストレージ（HDD/SSD）

作成した書類やインストールしたアプリをしまっておく場所です。 本棚が大きければ（ストレージ容量が大きければ）、写真や動画、ゲームをたくさん保存しておけます。



図の解説です。

- CPUは作業机（メモリ）にあるデータしか直接扱えません。
- 本棚（ストレージ）にある本を読みたいときは、一度作業机に広げてから読みます。これが「ロード」や「読み込み」と呼ばれる処理です。
- 作業が終わったら本棚に戻さないと、電源を切ったときに机の上は片付けられて（消えて）しまいます。だから「保存」が大切なのです。

入力装置・出力装置の例

コンピュータが人間とやり取りするための手足や感覚器官にあたるのが、入力装置と出力装置です。

入力装置（Input Devices）

人間側の情報をコンピュータが理解できるデジタルデータに変換して送る装置です。

- **キーボード**: 文字情報を入力します。
- **マウスやタッチパッド**: 画面上の位置（座標）やクリック動作を入力します。
- **マイク**: 空気の振動（音）を電気信号に変えて入力します。
- **カメラやスキャナ**: 光の情報を画像データとして入力します。
- **センサー**: 温度、加速度、GPSなどの物理情報を入力します。

出力装置（Output Devices）

コンピュータ内部の処理結果を、人間が理解できる形（光、音、紙など）に変換して出す装置です。

- **ディスプレイ（モニタ）**: データを光の点（ピクセル）の集まりとして表示します。
- **スピーカーやイヤホン**: デジタル信号を空気の振動（音）に変えて出します。
- **プリンタ**: データを紙にインクで定着させます。
- **バイブレーション機能**: データを物理的な振動で伝えます（スマホなどで利用）。

ソフトウェア

ここまでの内容でコンピュータの体であるハードウェアを学びました。ここからは、コンピュータの思考にあたるソフトウェアに入ります。

ハードウェア（CPUやメモリなど）は高性能ですが、それだけでは何もできません。何をどう処理するかという命令の手順書があって初めて、役に立つ道具になります。この手順書の集まりがソフトウェアです。

ソフトウェアは大きく分けて、OS（基本ソフトウェア）とアプリケーションソフト（応用ソフトウェア）の2種類があります。

OS（オペレーティングシステム）とは何か

OS（Operating System）は基本ソフトウェアとも呼ばれ、コンピュータが動作するための土台となるソフトウェアです。

コンピュータの現場監督

OSの役割をひと言でいえば、ハードウェアの管理とアプリケーションへの場所貸しです。

もしOSがなかったら、プログラマーは画面に文字を一つ表示するだけでも、画面のどのピクセルを光らせるか、電気信号をどう送るかといったハードウェアの詳細な制御を一から書かなくてはなりません。これは膨大な作業です。

OSはそうした面倒なハードウェア制御を一手に引き受け、アプリケーション（ブラウザやゲームなど）が簡単に動ける環境を提供しています。

OSの主な役割は次の4つです。

1. **ハードウェアの管理**: CPUやメモリ、ストレージといった物理的な資源を、どのアプリにどれくらい使わせるか調整します。
2. **ファイル管理**: ストレージ上のデータをファイルやフォルダという人間にわかりやすい形で整理・保存します。
3. **ユーザーインターフェース（UI）の提供**: ウィンドウやメニュー、マウス操作など、人間がコンピュータを操作するための基本的な画面を用意します。
4. **API（機能）の提供**: 音を出す、ファイルを保存するといった共通機能を、アプリケーションが使いやすい形で公開します。

代表的なOSの種類

現在、世界中で使われている代表的なOSには以下のものがあります。

- **PC向け**
  - **Windows**: Microsoft社製。世界で最も普及しているOSです。
  - **macOS**: Apple社製。Macコンピュータに搭載されています。
  - **Linux**: オープンソース（無料）のOS。サーバーや開発者向けによく使われます。
- **スマートフォン・タブレット向け**
  - **iOS / iPadOS**: Apple社製。iPhoneやiPadに搭載されています。
  - **Android**: Google社主導で開発。多くのメーカーのスマートフォンに搭載されています。

OSの情報を取得するコード

私たちが書くプログラム（Pythonなど）もOSの上で動いています。以下のコードは、Pythonで自分が今動いているOSの情報を取得する例です。

```
import platform
import os

# OSの名前を取得します（例: Windows, Darwin(=macOS), Linux）
os_name = platform.system()

# OSの詳細なバージョン情報を取得します
os_version = platform.release()

print(f"現在動作しているOS: {os_name}")
print(f"OSのバージョン: {os_version}")

# OSの機能を使って、現在の「作業ディレクトリ（今いるフォルダ）」を教えてもらいます
current_directory = os.getcwd()
print(f"現在の作業場所: {current_directory}")
```

コードの解説です。

- `import platform` と `import os` は、OSの機能をPythonから利用するための窓口となるライブラリです。
- `platform.system()` は、PythonがOSに対して名前を問い合わせている処理です。

- プログラマーがWindowsでもMacでも同じコード（`print`など）を書けるのは、OSが裏側で機種ごとの違いを吸収してくれているからです。

アプリケーションソフトの例

アプリケーションソフトは、OSの上で動作し、ユーザーが特定の目的（文章を書く、Webを見る、絵を描くなど）を達成するために作られたソフトウェアです。単にアプリやソフトとも呼ばれます。

OSが家（土台）だとすれば、アプリケーションはそこで使われる家具や家電にあたります。

目的ごとの専門家

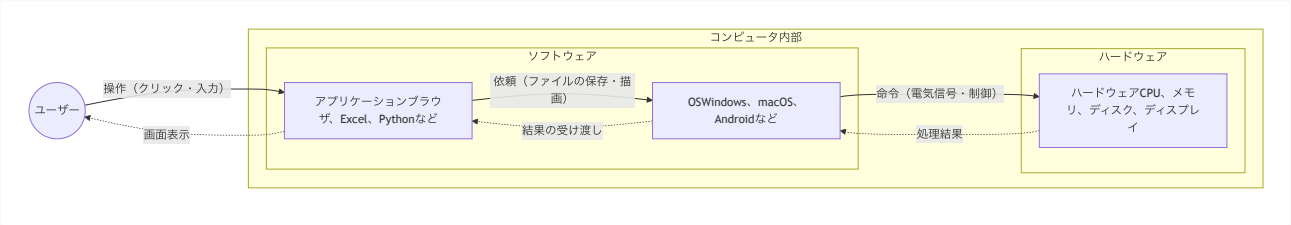
アプリケーションは、それぞれ得意な仕事を持っています。

- **Webブラウザ**（Chrome, Safari, Edge）：インターネット上のWebページを表示します。
- **オフィスソフト**（Word, Excel, PowerPoint）：文書作成、表計算、プレゼンテーションを扱います。
- **ゲームソフト**: エンターテインメントを提供します。
- **開発ツール**（VS Code, Python本体）：プログラムを書いたり実行したりします。

私たちはOSそのものを操作するというよりも、OSの上で動くこれらのアプリケーションを通じてコンピュータの恩恵を受けています。

アプリケーションの階層構造

コンピュータの中では、ユーザー、アプリケーション、OS、ハードウェアが以下の階層構造で連携しています。



図の解説です。

- **User → App**: ユーザーはアプリケーションを操作します。保存ボタンを押す、などです。
- **App → OS**: アプリは直接ハードディスクを操作しません。「このデータを保存して」とOSに頼みます。これをシステムコールと呼びます。
- **OS → Hardware**: 依頼を受けたOSが、実際にハードウェアを制御してデータを書き込みます。

ハードウェアとソフトウェアの関係

ハードウェアとソフトウェアは、**「肉体と精神」**や**「ピアノと楽譜」**のような関係です。

- ハードウェア（肉体）だけでは、動くための意思（手順）がありません。
- ソフトウェア（精神）だけでは、実行するための体（機械）がありません。

両方が揃って初めて、私たちはスマートフォンで動画を見たり、AIと会話したりすることができます。

ビットによる情報の表現

普段、コンピュータの画面で文章を読んだり写真を見たり音楽を聴いたりしていますが、ハードウェアの中ではこれらの情報がすべて数値として扱われています。しかも、私たちが使う0から9の数字（10進数）ではなく、0と1だけを使った世界です。

このセクションでは、コンピュータがどうやって0と1だけで複雑な情報を表現しているか、その仕組みを学びます。

ビット（bit）

情報の最小単位

ビット（bit）は、binary digit（2進数の数字）の略で、コンピュータが扱う情報の最小単位です。

ビットが持てる状態は2通りだけです。

- **0** か **1**
- **OFF** か **ON**
- **False（偽）** か **True（真）**

電気のスイッチを思い浮かべてください。部屋の明かりはついているかいないかのどちらかです。このスイッチ1つ分の情報量が1ビットです。

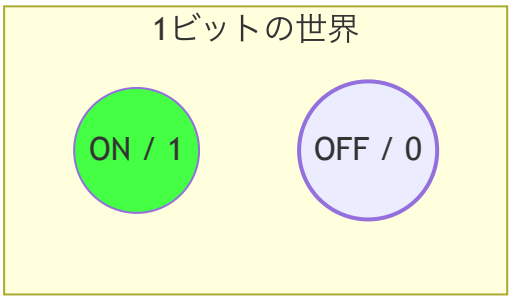
物理的な仕組みとの対応

コンピュータが0と1しか使わないのは、機械にとってそのほうが扱いやすく、間違いが少ないからです。

コンピュータ内部の電子回路では、次のような物理的状態でビットを表現しています。

- **電気信号**: 電圧が高い（5Vなど）なら1、低い（0V近く）なら0。
- **磁気ディスク**: 磁石の向きがN極なら1、S極なら0。
- **光ディスク**: 表面に凹凸があれば1、なければ0。

  0.5や3のような中途半端な状態を作ろうとすると、少しのノイズ（電圧の乱れなど）で値が変わってしまう危険がありますが、あるかないかの2択なら、多少のノイズがあっても正確に情報を伝えられます。



バイト（byte）

1ビットだけでは、はいかいいえの2通りしか表せません。文字や画像を表現するには足りなさすぎます。 そこで、ビットを束にして扱います。

8ビット ＝ 1バイト

バイト（byte）は、8個のビットをまとめた単位です。現在のコンピュータ処理の基本単位になっています。

- 1ビットで2通り（0, 1）
- 2ビットで4通り（00, 01, 10, 11）
- 3ビットで8通り（000, 001, ... 111）
- ...
- **8ビット（1バイト）で256通り**（2^8通り）

1バイトあれば0から255までの数値を表せます。アルファベットの大文字・小文字や数字、記号を十分区別して番号を振れる数です。

データの大きさの単位

データ量が増えるとバイトだけでは桁が大きくなりすぎるため、より大きな単位を使います。

- **KB（キロバイト）**: 約1,000バイト（正確には1024バイト = 2^{10}バイト）
- **MB（メガバイト）**: 約100万バイト（1024 KB）
- **GB（ギガバイト）**: 約10億バイト（1024 MB）

- **TB（テラバイト）**: 約1兆バイト（1024 GB）

※ 日常生活（10進数）では「キロ＝1000」ですが、コンピュータ（2進数）の世界では1024（2^{10}）が区切りになるのが一般的です。

数値の表現（2進数）

人間は「0〜9」の10種類の数字を使いますが、コンピュータ内部では「0と1」の2種類だけを使う**2進数**が使われています。

10進数（人間）	2進数（コンピュータ）
0	0
1	1
2	10
3	11
4	100

どんなに大きな数でも、0と1の組み合わせだけで表現できます。

文字の表現（文字コード）

文字も数値（0と1の組み合わせ）に置き換えられます。この「どの文字をどの番号にするか」という対応表を**文字コード**と呼びます。

- **UTF-8**: 世界中の文字（日本語、英語、絵文字など）を扱える世界標準のルールです。

Webサイトやメールが「文字化け」するのは、この対応表（ルール）を送り手と受け手で間違えてしまったときに起こります。

メディアの表現

文字だけでなく、画像や音もすべて数値として扱われます。

- **画像**: 小さな色の点（ピクセル）の集まりです。それぞれの点に「赤の強さ」「緑の強さ」「青の強さ」を数値で割り振っています。
- **音**: 空気の震え（波）の高さを、非常に短い間隔（1秒間に何万回など）で細かく測定して数値化しています。

このセクションのまとめ

コンピュータは魔法の箱ではなく、あらゆる情報を0と1の膨大な組み合わせに置き換えて処理する超高速な計算機です。

- ビットは世界最小の情報単位（0か1）。
- バイトは8ビットまとめた単位で、文字などを表す基本サイズ。
- 文字も画像も音も、すべてルールに従って数値（ビット列）に変換されている。

この裏側の仕組みを知っていると、プログラミングでデータを扱う際のエラー（文字化けや型の不一致など）の意味がよく理解できるようになります。

問題1 10進数から2進数への変換

問題文

正の整数Nが与えられます。 Nを2進数に変換し、その2進数表現を出力してください。

入力データの特徴

- $1 \leq N \leq 1000$

- 入力は整数である。

入力

入力は以下の形式で標準入力から与えられる。

```
N
```

出力

Nの2進数表現を出力せよ（先頭に `0b` を付けないこと）。

入力例1

```
5
```

出力例1

```
101
```

5を2進数に変換すると `101` になります。

入力例2

```
10
```

出力例2

```
1010
```

10を2進数に変換すると `1010` になります。

入力例3

```
255
```

出力例3

```
11111111
```

255は8ビットで表現できる最大値であり、2進数では `11111111` となります。

問題2 データサイズの計算

問題文

データサイズN（バイト単位）が与えられます。 このデータサイズを、適切な単位（B, KB, MB, GB）で表示してください。

単位の変換規則は以下の通りです：

- 1 KB = 1024 B
- 1 MB = 1024 KB
- 1 GB = 1024 MB

入力データの特徴

- $1 \leq N \leq 10^{12}$
- 入力は整数である。

入力

入力は以下の形式で標準入力から与えられる。

```
N
```

出力

以下の規則に従って出力せよ：

- 1024未満の場合は `X B` （例: `512 B` ）
- 1024以上1048576未満の場合は `X KB` （例: `256 KB` ）
- 1048576以上1073741824未満の場合は `X MB` （例: `100 MB` ）
- 1073741824以上の場合は `X GB` （例: `5 GB` ）

ただし、Xは整数部分のみを出力し、小数点以下は切り捨てる。

入力例1

```
512
```

出力例1

```
512 B
```

512バイトは1KB未満なので、そのままバイト表記します。

入力例2

```
1048576
```

出力例2

```
1 MB
```

1048576バイト = 1024 KB = 1 MB です。

入力例3

```
5368709120
```

出力例3

```
5 GB
```

5368709120バイト = 5 GB です。

問題3 文字コードの変換

問題文

1文字の英字（大文字または小文字）が与えられます。 その文字のASCIIコード（10進数）を出力してください。

入力データの特徴

- 入力は半角英字1文字である（A-Z または a-z）。

入力

入力は以下の形式で標準入力から与えられる。

```
C
```

出力

文字CのASCIIコード（10進数）を出力せよ。

入力例1

```
A
```

出力例1

```
65
```

大文字「A」のASCIIコードは65です。

入力例2

```
a
```

出力例2

```
97
```

小文字「a」のASCIIコードは97です。

入力例3

```
Z
```

出力例3

```
90
```

大文字「Z」のASCIIコードは90です。

第2章 プログラム

プログラム

前の章では、コンピュータというハードウェア（体）について学びました。この章では、そのコンピュータを動かすための指示書であるプログラムを詳しく見ていきます。

コンピュータは自ら考えて動くわけではありません。人間が「何を、どの順番でやりなさい」と事細かに指示を与えて初めて、その能力を発揮します。この指示の束がプログラムです。

プログラムとは何か

プログラムとは、コンピュータに対する命令のリストです。

やってほしいことを、コンピュータが理解できる言葉（プログラミング言語）で順序立てて記述したものです。日常にあるものに例えると、こんなイメージです。

- **料理のレシピ**: 「玉ねぎを刻む」「肉を炒める」「水を加えて煮込む」という手順に従えば、誰でもカレーが作れます。プログラムも手順通りに処理が進みます。
- **音楽の楽譜**: 「ドの音を出す」「1拍休む」という記号に従って演奏すれば音楽になります。プログラムに従ってコンピュータは画面を表示したり音を出したりします。

命令を順序立てて書く

プログラムの最も基本的な性質は、書かれた順番通りに実行されることです。 以下のPythonコードは、カップラーメンを作る手順をプログラムに見立てたものです。

```
import time

print("--- カップラーメン作成プログラム開始 ---")

# 手順1: お湯を沸かす
print("手順1: お湯を沸かしています...")
# time.sleep(1) は、1秒待つという命令です（お湯が沸く時間を表現）
time.sleep(1)
print("  -> お湯が沸きました！")

# 手順2: フタを開ける
print("手順2: カップラーメンのフタを半分まで開けます。")

# 手順3: お湯を注ぐ
print("手順3: 線までお湯を注ぎます。")

# 手順4: 待つ
print("手順4: 3分間待ちます（ここでは3秒短縮します）。")
for i in range(1, 4):
    print(f"  {i}秒経過...")
    time.sleep(1)

# 手順5: 完成
print("手順5: 完成！いただきます。")

print("--- プログラム終了 ---")
```

コードの解説です。

- このプログラムを実行すると、上から下へ順番に文字が表示されます。
- `time.sleep(1)` のような命令も、書かれた場所に到達して初めて実行されます。
- もし「お湯を注ぐ」と「フタを開ける」の順番を間違えて書くと、現実と同じように、プログラムも正しく動きません（フタの上からお湯をかけることになります）。

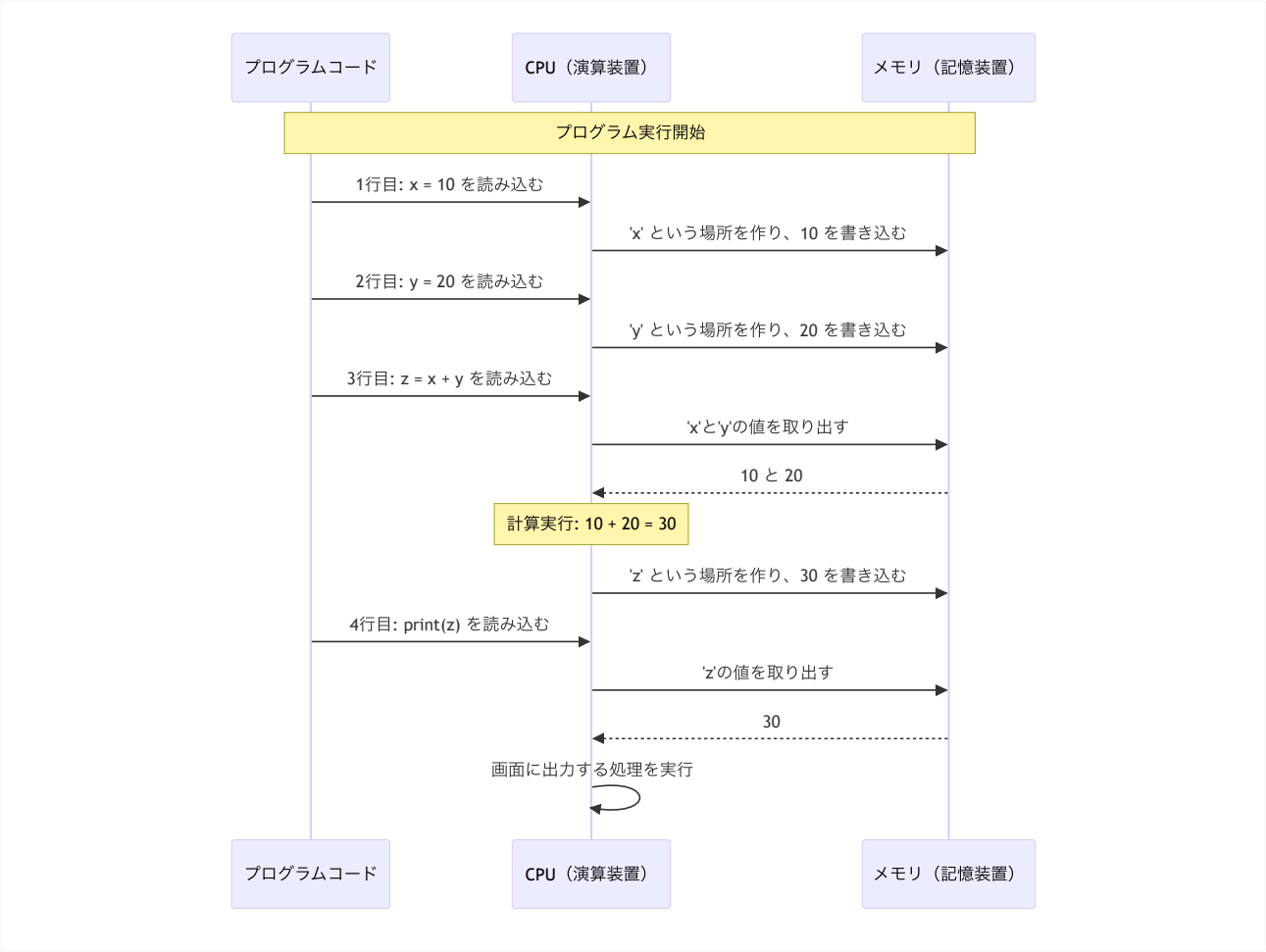
プログラムはどのように実行されるか

書いたプログラム（ソースコード）は、コンピュータ内部でどのように処理されているのでしょうか。CPUとメモリの働きを中心に見ていきます。

実行の流れ

1. **読み込み（ロード）**：ストレージに保存されているプログラムファイルが、メモリ（作業机）上に読み込まれます。
2. **解釈と実行**: CPU（頭脳）がメモリ上の命令を1行ずつ読み取り、実行します。
3. **変数の確保**: 計算結果を保存したいという命令が来ると、メモリ上に場所を確保し、そこに値を書き込みます。

この流れを図で整理します。



このように、コンピュータは「命令」を一つずつ読み込み、メモリ（データ置き場）の値を書き換えながら、一歩ずつ処理を進めていきます。

プログラムの翻訳（コンパイルとインタプリタ）

PythonやJavaScriptで書くプログラムは、人間にとって読みやすい言葉（高水準言語）で書かれています。しかし、CPUは0と1でできた機械語（マシン語）しか理解できません。

そのため、プログラムを実行するには人間語から機械語へ翻訳する必要があります。翻訳方式は大きく2種類です。

1. コンパイラ方式（一括翻訳）

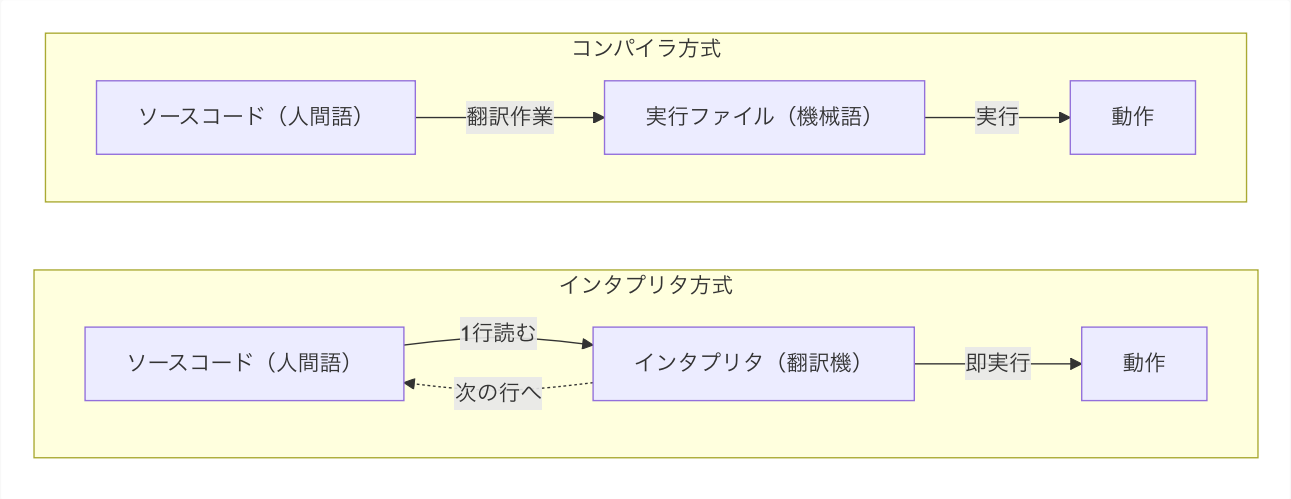
プログラム全体を一度にすべて機械語に翻訳し、翻訳済みのファイル（実行ファイル）を作る方式です。

- 海外の小説を翻訳家が時間をかけて日本語版の本に仕上げるイメージです。
- 実行時には翻訳作業が不要なので動作が高速です。
- 書き直すたびに翻訳し直す必要があり、少し手間がかかります。
- 代表的な言語はC言語、C++、Java（Javaは中間方式ですがコンパイルが必要です）。

2. インタプリタ方式（逐次翻訳）

プログラムを1行ずつ読み込みながら、その場で機械語に翻訳して実行する方式です。

- 外国人のスピーチを通訳者がリアルタイムで日本語に訳す同時通訳のイメージです。
- 書き換えたらすぐ実行できるため、修正や確認が手軽です。
- 実行中に翻訳を行う分、コンパイラ方式より動作が少し遅くなります。
- 代表的な言語はPython、JavaScript、Rubyです。



私たちが学ぶPythonは？

この講座で主に使うPythonはインタプリタ方式の言語です。コードを書いてすぐに実行結果を確認できるため、学習や試行錯誤に向いています。

プログラムとソフトウェアの違い

「プログラム」と「ソフトウェア」は日常会話ではほぼ同じ意味で使われますが、技術的な文脈では少しニュアンスが異なります。

プログラム ＝ コードそのもの

プログラムとは、これまで見てきた「コンピュータへの命令が書かれたテキスト（ソースコード）」や、それが翻訳された実体を指します。機能を実現するための部品や記述に焦点を当てた言葉です。

ソフトウェア ＝ 製品としての全体

ソフトウェア（ソフト）は、プログラムに加え、それを利用するために必要なデータや文書を含んだパッケージ全体を指します。

- 構成要素の例
  - プログラム本体（メインの命令書）
  - 設定ファイル（動作を調整するデータ）
  - 画像・音声データ（アイコンやBGMなどの素材）
  - ドキュメント（マニュアルやヘルプファイル）

ゲームソフトで例えると

- **プログラム**: キャラクターをコントローラーで動かしたり、ダメージ計算をしたりするロジック部分。

- **ソフトウェア**: それに加え、キャラクターの絵、BGM、ステージのデータ、パッケージや説明書まで含めた商品全体。

違いを構造で見る

以下のディレクトリ構造を見ると、ソフトウェアがプログラム以外のものも含んでいることがわかります。

```
# これ全体が「お絵かきソフトウェア」です

MyPaintApp/                 <-- ソフトウェアのフォルダ
 ├── bin/
 │    └── main.exe            <-- これが「プログラム」（実行ファイル）
 ├── config/
 │    └── settings.json       <-- 設定データ
 ├── assets/
 │    ├── icon.png            <-- 画像データ
 │    └── sound.wav           <-- 音声データ
 └── README.txt              <-- 説明書（ドキュメント）
```

プログラムはソフトウェアの一部です。 これから行うプログラミングは、ソフトウェアの中核となるプログラムを作る作業ですが、最終的に人に使ってもらうにはデータや説明書も整えてソフトウェアとして仕上げる視点も必要になります。

出力

プログラムにおける出力について学びます。 前の節で、プログラムは命令のリストであり、CPUがそれを処理することを見てきました。しかし、どれだけ素晴らしい計算をしても、結果を知る手段がなければ意味がありません。

コンピュータが処理した結果を、人間や他のシステムに伝える行為。それが出力（Output）です。

コンソール画面

プログラミングを学び始めて最初に触れる、最も基本的な出力先がコンソール画面です。

コンソールとは何か

コンソール（Console）は、コンピュータと文字だけで対話するためのウィンドウです。ターミナルやコマンドプロンプトとも呼ばれます。 映画に出てくるハッカーが使っているような、黒い背景に白い文字だけが表示される画面を思い浮かべてください。

現代のアプリケーションはボタンや画像が豊富なGUI（グラフィカルユーザーインターフェース）が主流ですが、開発の現場では今でも文字だけのCUI（キャラクターユーザーインターフェース）であるコンソールが活躍しています。

プログラマの聴診器

なぜわざわざ質素なコンソールを使うのでしょうか。コンソールは最も手軽で、ごまかしのきかない情報源だからです。

プログラムを作っていると、計算が合わない、途中で止まるといったトラブル（バグ）によく遭遇します。コンソールに途中の変数の値や「ここまでは動いた」というメッセージを表示させれば、プログラム内部の状態を診断できます。

これをデバッグ（虫取り）と呼びます。医者が聴診器で体の音を聞くように、プログラマはコンソールへの出力を見てプログラムの状態を確認します。



出力 = プログラムの実行結果

出力は単なる文字の表示ではありません。プログラムにとって、その処理が行われた証であり、目的そのものです。

情報を返す行為

プログラムは基本的に、入力を受け取り、処理を行い、出力を返すという流れで動きます（IPOモデル）。 出力とは、処理によって生まれた情報を外の世界へ送り出すことです。

- 電卓プログラムなら、計算した答えを表示するのが出力です。
- ゲームなら、現在のスコアや勝敗を表示するのが出力です。

Pythonでの表示（出力）の例

Pythonでは `print()` という命令を使って、画面（コンソール）に結果を表示します。

```
print("こんにちは")      # 文字を表示
print(100 + 200)       # 計算結果を表示
```

文字列（メッセージ）は `" "` で囲み、数値はそのまま書くのが基本のルールです。

様々な出力

出力というと画面への文字表示を思い浮かべがちですが、実際にはもっと広い意味があります。コンピュータから外へ情報が出ていくものはすべて出力です。

五感に訴える出力

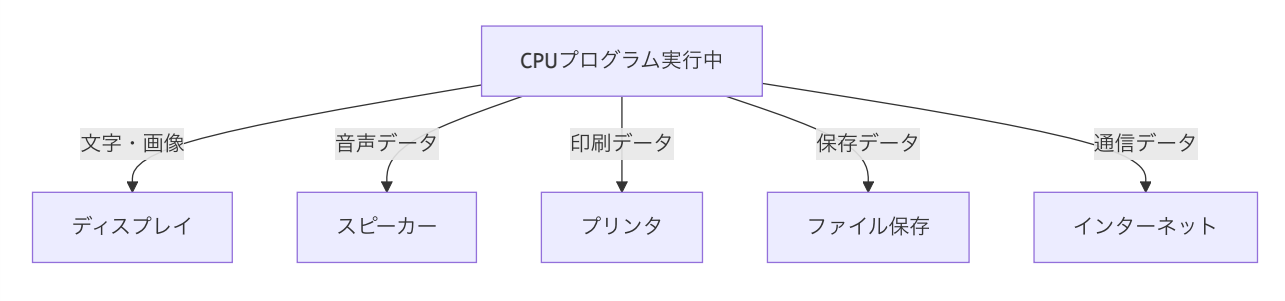
人間が受け取れる出力にはこんな種類があります。

- **視覚**
  - **ディスプレイ**: 文字だけでなく画像や動画の描画。
  - **LED**: 電源ランプの点灯や信号機の光。
  - **プリンタ**: 紙への印刷。
- **聴覚**
  - **スピーカー**: 音楽の再生、エラー時の警告音（ビープ音）、合成音声。
- **触覚**
  - **振動（バイブレーション）**: スマートフォンの通知やゲームコントローラーの振動。

データとしての出力

人間が直接見聞きするだけでなく、データとして保存・送信することも出力の一形態です。

- **ファイルへの書き込み**: メモ帳で文章を保存するとき、プログラムはストレージ上のファイルに文字データを出力しています。
- **ネットワーク送信**: ブラウザで検索ボタンを押すと、検索キーワードがインターネットを通じてサーバーへ送信されます。これも出力です。



ファイルへの書き出し（出力）

画面への表示だけでなく、計算結果を「ファイル」として保存することも立派な出力です。

```python
# 'memo.txt' という名前でファイルを保存する例
with open("memo.txt", "w", encoding="utf-8") as f:
    f.write("これはファイルに保存されたメッセージです。")
```

このプログラムを実行すると、パソコンの中に新しく `memo.txt` というファイルができあがります。

入力

出力がプログラムから外の世界へ情報を出すことだとすれば、入力（Input）はその逆です。外の世界からプログラムへ情報を取り込みます。

入力 = プログラムへの材料提供

プログラムにとっての入力は、処理を行うための材料や指示を受け取る行為です。

料理ロボットで考える

カレーを作るプログラムが組み込まれた料理ロボットがいるとします。

- **入力がない場合**: ロボットはあらかじめ決まったじゃがいもと豚肉を使い、毎回同じカレーを作ります。
- **入力がある場合**: 「今日は牛肉を使って」「辛口で」と指示を与えると、同じ手順を使いながらも、要望に合わせたビーフカレーを作ってくれます。

汎用性が生まれる

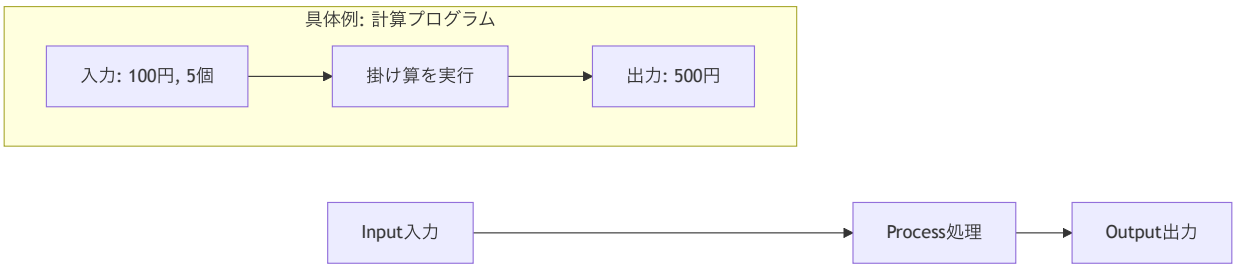
入力があることで、プログラムは毎回同じことしかできない機械から、状況に合わせて異なる結果を出せる汎用的なツールに変わります。

- 入力なしのプログラムは、常に1 + 1の結果を表示します（答えはいつも2）。
- 入力ありのプログラムは、渡された2つの数字を足して結果を表示します（10と20なら30、5と3なら8）。

入力と出力の関係（IPOモデル）

コンピュータシステムの動作は、どんなに複雑でも基本的に3ステップで説明できます。これをIPOモデルと呼びます。

1. **Input（入力）**: データを外部から受け取る。
2. **Process（処理）**: 受け取ったデータを加工・計算・判断する。
3. **Output（出力）**: 処理結果を外部へ返す。



設計の重要性

プログラミングでは、コードを書き始める前にIPOを整理することが大切です。

- Input: 何が必要か？（ユーザーの名前？ 商品の金額？）
- Output: 何がゴールか？（挨拶を表示する？ 合計金額を出す？）
- Process: 入力をどうすれば出力に変えられるか？（文字列を連結する？ 掛け算をする？）

IPOモデルの考え方

プログラムを作る前に、この3つのステップを整理することが大切です。

- **Input（入力）**：何が必要か？（例：単価150円、個数3個）
- **Process（処理）**：どう計算するか？（例：150 × 3 ＝ 450円）
- **Output（出力）**：何を出すか？（例：「合計450円です」と表示）

入力（Input）の数字を変えれば、プログラムの中身（Process）を変えなくても、自動的に結果（Output）が変わる。この仕組みこそがプログラムの便利な点です。

様々な入力

入力というとキーボードで文字を打ち込むイメージが強いですが、現代のプログラムはさまざまな方法で入力を受け取ります。

人間からの入力

- **キーボード**: 文字やコマンドの入力。
- **マウスやタッチパネル**: クリック位置やスワイプ方向。
- **マイクやカメラ**: 音声データや映像データ。

機械・環境からの入力

- **センサー**: スマホのGPS（位置情報）、加速度センサー（傾き）、温度計データなど。
- **ファイル読み込み**: 保存されているテキストや画像を取り込むのも、プログラムにとっては入力です。
- **インターネット**: Webサーバーが世界中からの「ページを見せて」というリクエストを受け取るのも入力です。

データの読み込み（入力）

保存されているファイルを読み込んで、プログラムの中で使うこともできます。

```
# 'memo.txt' というファイルの中身を読み取る例
with open("memo.txt", "r", encoding="utf-8") as f:
    content = f.read()
    print(content)
```

あらかじめ作成しておいたデータを読み込ませることで、プログラムの動作を外から変えることが可能になります。

想定外の入力への対応

プログラムに入力を与えるのは、常に正しい操作をする人だけとは限りません。押し間違いや勘違いで、プログラムが想定していない異常値が入力されることもあります。

Garbage In, Garbage Out（ゴミが入ればゴミが出る）

コンピュータ用語にGIGO（ギーゴ）という言葉があります。質の悪いデータを入力すれば、質の悪い結果しか出てこないという意味です。 それどころか、想定外の入力はプログラムをエラーで停止させてしまうこともあります。

バリデーション（入力チェック）

プログラムを壊さないためには、処理を始める前に入力データが正しいかどうかをチェックする必要があります。これを**バリデーション**と呼びます。

例えば、年齢を入力してもらうときに「-5歳」や「あいうえお」といった文字が入ってきたらどうでしょうか。そのまま計算を続けるとプログラムが止まってしまいます。 そのため、「0以上の数字であるか？」などを事前に確認し、正しくない場合は「もう一度入力してください」と促す仕組みを作ることが大切です。

データ型

プログラムで扱うデータには、必ず型（Type）という概念があります。 ここでは、コンピュータがデータをどのように分類して扱っているかを学びます。

データ型の種類

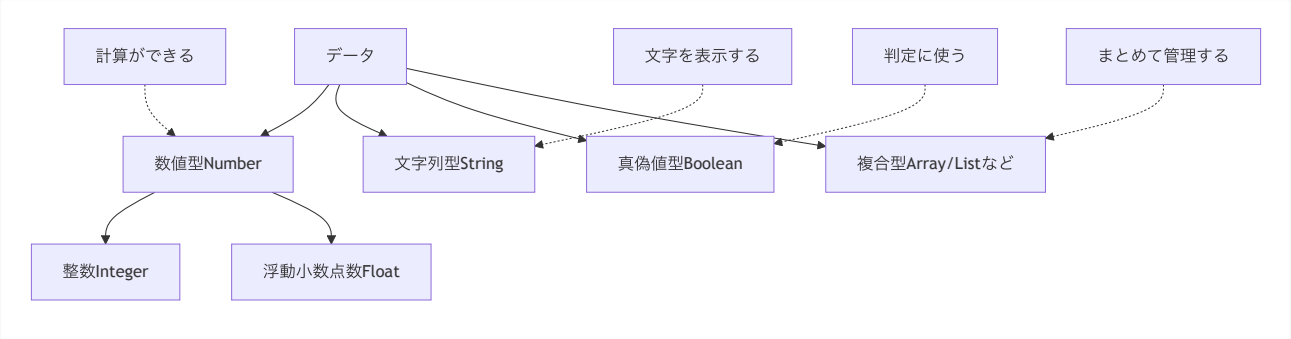
コンピュータにとってデータはすべて0と1の並びですが、人間が扱う際には「これは数値」「これは文字」といった意味づけが必要です。このデータの種類をデータ型（Data Type）と呼びます。

なぜ型を意識するのか？

型によって、できることとできないことが変わるからです。

- 数値なら足し算や引き算ができます。
- 文字列（テキスト）なら文字をつなげたり、特定の文字を探したりできます。
- しかし、文字と文字を掛け算することはできません（「あ」×「い」＝？ と言われても困ります）。

プログラミング言語はそれぞれのデータに型を割り当てることで、間違った操作（文字同士の割り算など）を防いだり、処理方法を選んだりしています。



数値（Number / Integer, Float）

計算に使えるデータ型です。

- **整数（Integer / int）**：`10`，`-5`，`0`
- **浮動小数点数（Float）**：`3.14`，`0.5`

数値と文字列の違いに注意

プログラムの中では、数値の `10` と、文字としての `"10"` は別物として扱われます。 数値であれば `10 + 20` は `30` になりますが、文字であれば `"10" + "20"` は単に文字がつながって `"1020"` になります。

文字列（String）

名前や文章などを扱うためのデータ型です。

- **表記**: `'こんにちは'` や `"Hello"` のように、引用符（クォート）で囲みます。
- **操作**: 文字同士をつなげたり、文字数を数えたりすることができます。

真偽値（Boolean）

`True`（正しい・ON）または `False`（間違い・OFF）の2つの状態だけを持つデータ型です。

「もし得点が80点以上なら（True）合格」といった、プログラムの判断に使われます。

配列（Array / List）

複数のデータをまとめて、順番に並べて管理するための箱です。Pythonでは**リスト（List）**と呼びます。

- **例**: `["Red", "Green", "Blue"]`
- **特徴**: それぞれのデータには `0` から始まる番号（インデックス）が振られており、その番号を使ってデータを取り出します。

演習（プログラミング問題）

この章の締めくくりとして、実際にプログラムを動かす演習を行います。 ゼロからコードを書く必要はありません。生成AIにコードを書かせて、人間がそれを確認するという現代的な開発プロセスを体験してみます。

AIを用いたコード生成の体験

これまでに学んだ入力、出力、変数、データ型が、実際のプログラムでどう使われているかを確認します。

テーマ：BMI（体格指数）計算プログラム

身長と体重からBMIを計算するプログラムを作ります。 自分でコードを書くのではなく、AI（ChatGPTやClaudeなど）に指示を出して作らせる想定です。

1. AIへの指示（プロンプト）

AIにプログラムを作ってもらうときは、やりたいことを具体的に伝えます。 今回は以下のような指示を出したとします。

> ***指示（プロンプト）：****PythonでBMIを計算するプログラムを書いてください。 以下の要件を満たすようにしてください。*
>
> 1. *身長（cm）と体重（kg）は、プログラムの先頭で**変数**として定義してください（入力の代わり）。*
> 2. *BMIの計算式は `体重(kg) ÷ (身長(m)の2乗)` です。身長の単位変換（cm → m）を忘れないでください。*
> 3. *計算結果（**出力**）は、「あなたのBMIは ○○ です」という形式で表示してください。*
> 4. *コードの各部分に、何をしているかの解説コメントを入れてください。*

2. 生成されたコード

上の指示に対してAIが生成したコードが以下です。 コピーして手元のPython環境で実行してみてください。

```
# --- 1. 入力（Input） ---
# ここで身長と体重を設定します
# この数字を書き換えることで、様々な人のBMIを計算できます
height_cm = 170.0  # 身長（センチメートル）
weight_kg = 65.0   # 体重（キログラム）

print(f"身長: {height_cm} cm")
print(f"体重: {weight_kg} kg")

# --- 2. 処理（Process） ---
# BMIの計算を行います

# 計算のために、身長を cm から m に変換します（100で割る）
# 例: 170cm -> 1.7m
height_m = height_cm / 100

# BMIの計算式: 体重 ÷（身長 × 身長）
bmi = weight_kg / (height_m * height_m)

# 補足: 結果を見やすくするために、小数点以下2桁に丸める処理
# round(数値, 桁数) という関数を使います
bmi_rounded = round(bmi, 2)

# --- 3. 出力（Output） ---
# 計算結果を画面に表示します
print("-----------------------")
print(f"あなたのBMIは {bmi_rounded} です。")
print("-----------------------")
```

```
# 追加機能: 簡単な肥満度判定（条件分岐）
if bmi < 18.5:
    print("判定: 低体重（痩せ型）です。")
elif bmi < 25:
    print("判定: 普通体重です。")
else:
    print("判定: 肥満気味です。")
```

3. コードの解説と確認

実行できたら、コードの中身を読んでみます。これまで学んだ用語がどこに隠れているか探してください。

1. **変数とデータ型** `height_cm = 170.0` の部分で、数値（小数なのでFloat型）を変数に入れています。これがプログラムへの入力です。数値を `180.0` や `60.0` に書き換えて再実行すると、結果が変わることを確認してください。
2. **演算（処理）** `height_cm / 100` や `weight_kg / ...` の部分で、算術演算子（割り算や掛け算）を使っています。コンピュータは指示された通りの計算を瞬時に行います。
3. **出力** `print(...)` で計算結果を人間に伝えています。 `f"..."` の中で変数を埋め込み、文章として整形して出力しています。
4. **論理（アルゴリズム）** 最後の `if` 文はまだ詳しくは学んでいませんが、「もしBMIが18.5より小さければ...」という条件分岐を行っています。数値の大小比較（Boolean型）がここで使われています。

プログラムは魔法ではなく、変数、演算、入出力という小さな要素の積み重ねでできています。AIにコードを書かせた場合でも、どの部分が入力で、どこで計算して、どこで出力しているかを読み解く力がエンジニアには求められます。

問題1 税込価格の計算

問題文

商品の税抜価格P（円）が与えられます。 消費税率10%で計算した税込価格を出力してください。

ただし、小数点以下は切り捨てて整数で出力すること。

入力データの特徴

- $1 \leq P \leq 1000000$
- 入力は整数である。

入力

入力は以下の形式で標準入力から与えられる。

```
P
```

出力

税込価格（整数）を出力せよ。

入力例1

```
100
```

出力例1

```
110
```

100円の商品の税込価格は 100 × 1.1 = 110円です。

入力例2

```
198
```

出力例2

```
217
```

198円の商品の税込価格は 198 × 1.1 = 217.8円、小数点以下を切り捨てて217円です。

入力例3

```
1
```

出力例3

```
1
```

1円の商品の税込価格は 1 × 1.1 = 1.1円、小数点以下を切り捨てて1円です。

問題2 文字列の長さと反転

問題文

文字列Sが与えられます。 以下の2つの情報を、それぞれ別の行に出力してください。

1. 文字列Sの長さ
2. 文字列Sを反転させた文字列

入力データの特徴

- $1 \leq |S| \leq 100$（$|S|$は文字列Sの長さ）
- Sは半角英数字のみからなる。

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

1行目に文字列の長さ、2行目に反転させた文字列を出力せよ。

入力例1

```
hello
```

出力例1

```
5
olleh
```

「hello」は5文字で、反転すると「olleh」になります。

入力例2

```
abc123
```

出力例2

```
6
321cba
```

入力例3

```
A
```

出力例3

```
1
A
```

1文字の場合、反転しても同じ文字になります。

問題3 配列の合計と平均

問題文

N個の整数が与えられます。 これらの整数の合計と平均を計算して出力してください。

ただし、平均は小数点以下を切り捨てて整数で出力すること。

入力データの特徴

- $1 \leq N \leq 100$
- 各整数は0以上1000以下である。

入力

入力は以下の形式で標準入力から与えられる。

```
N
A_1 A_2 ... A_N
```

1行目に整数の個数N、2行目にスペース区切りでN個の整数が与えられる。

出力

1行目に合計、2行目に平均（小数点以下切り捨て）を出力せよ。

入力例1

```
5
10 20 30 40 50
```

出力例1

```
150
30
```

合計は10+20+30+40+50=150、平均は150÷5=30です。

入力例2

```
3
7 8 9
```

出力例2

```
24
8
```

合計は7+8+9=24、平均は24÷3=8です。

入力例3

```
4
1 2 3 4
```

出力例3

```
10
2
```

合計は10、平均は10÷4=2.5を切り捨てて2です。

第3章 データとファイル

テキストデータ形式（CSV、JSON、YAML）

これまでの章では、プログラムの中で変数を使ってデータを扱ってきました。しかし、変数はメモリ上にあるため、プログラムが終了したりパソコンの電源を切ったりすると、中身はきれいに消えてしまいます。

データを永続的に保存したり、インターネットを通じて他のコンピュータへ送ったりするためには、データをファイルや通信データとして書き出す必要があります。

このとき重要になるのが、どのような形式で書き出すかというルールです。

データを渡すための共通ルール

友人に手紙を送るとき、自分にしか読めない暗号で書いたら、友人は内容を理解できません。データも同じで、送り手（書き出すプログラム）と受け手（読み込むプログラム）が共通のルールに従っていないと、正しく情報を伝えることができません。

テキスト形式の重要性

データの保存形式には大きく分けて2種類あります。

1. **バイナリ形式**: コンピュータにとって扱いやすい0と1のデータです。画像や音声、動画などがこれにあたります。人間が見ても意味不明な記号の羅列に見えます。
2. **テキスト形式**: 人間が読める文字で書かれたデータです。メモ帳などのエディタで開いて、中身を確認・編集できます。

プログラミングの世界、特に設定ファイルやデータのやり取りにおいては、テキスト形式が圧倒的に広く使われています。人間が目で見て内容を確認でき、修正もしやすいためです。

ここでは、代表的な3つのテキストデータ形式、**CSV**、**JSON**、**YAML**について学びます。

CSV（Comma-Separated Values）

CSVは、日本語でカンマ区切り値という意味です。その名の通り、データをカンマ `,` で区切って並べたシンプルな形式です。

表計算ソフトとの親和性

CSVは、ExcelやGoogleスプレッドシートのような表形式のデータを表現するのによく使われます。 1行が1つのデータ（レコード）を表し、カンマが列の区切りを表します。

表データのイメージです。

名前	年齢	出身地
佐藤	25	東京
鈴木	30	大阪
高橋	22	福岡

これをCSV形式にすると次のようになります。

```
名前,年齢,出身地
佐藤,25,東京
鈴木,30,大阪
高橋,22,福岡
```

CSVの特徴

- 非常にシンプルで、多くのソフトで読み書きできます。余計な記号が少ないため、データサイズも小さくなります。

- 一方で、データの種類（型）という情報を持てず、すべて文字扱いになることが多いです。複雑な階層構造を表現するのにも向いていません。

CSVの活用

プログラミングでは、リスト（配列）に保存されたデータをカンマでつなぎ合わせることで、簡単にCSV形式のデータを作り出すことができます。

JSON

JSON（ジェイソン）は、もともとJavaScriptという言語の書き方をベースに作られたデータ形式です。現在では言語を問わず、WebサービスやAPIで最も広く使われています。

キーと値のペア

JSONは、データの項目名である**キー**と、その内容である**値**をペアにして記述します。 また、波括弧 `{ }` でデータを囲み、角括弧 `[ ]` で配列（リスト）を表現します。

先ほどの表データをJSONで表現するとこうなります。

```
[
  {
    "名前": "佐藤",
    "年齢": 25,
    "出身地": "東京"
  },
  {
    "名前": "鈴木",
    "年齢": 30,
    "出身地": "大阪"
  }
]
```

JSONの特徴

- 階層構造（ネスト）を表現できます。住所の中に郵便番号と県名を入れるといった複雑なデータも、そのまま記述可能です。数値 `25` と文字列 `"25"` 、真偽値 `true/false` なども区別できます。
- 一方で、カッコや引用符が多くなるため、CSVに比べると人間が手書きするには少し手間がかかります。

JSONの活用

Pythonなどの言語では、専用の命令（ライブラリ）を使うことで、プログラム内の複雑なデータを一瞬でJSON形式に変換して保存したり、逆に読み込んだりすることができます。

YAML（YAML Ain't Markup Language）

YAML（ヤムル）は、人間にとっての読みやすさを最優先に設計された形式です。 主にソフトウェアの設定ファイル、たとえばサーバーのIPアドレスを指定するような場面でよく使われます。

インデントによる階層表現

JSONと似た構造を持ちますが、カッコ `{}` や `[]` を極力使いません。その代わり、インデント（行頭のスペース）を使ってデータの階層を表します。

先ほどのデータをYAMLで表現するとこうなります。

```
- 名前: 佐藤
  年齢: 25
  趣味:
```

```
    - 読書
    - 旅行

- 名前: 田中
  年齢: 19
  趣味:
    - ゲーム
```

- 先頭の `-` は、リストの項目であることを表します。
- `趣味:` の下の行が字下げされていることで、趣味に含まれるデータであることを示しています。

YAMLの特徴

- 記号が少なく、見た目がすっきりしていて読みやすいです。コメント（`#` で始まるメモ）を書くこともできます。JSONは標準ではコメントが書けないため、これは大きな利点です。
- 一方で、インデントの個数（スペースの数）を間違えるとエラーになるため、編集には注意が必要です。

各形式の使い分けまとめ

形式	主な用途	特徴	イメージ
CSV	データ分析、表計算	シンプル、構造なし	エクセルの表
JSON	Web API、データ通信	構造あり、機械寄り	アプリ間の小包
YAML	設定ファイル	構造あり、人間寄り	箇条書きのメモ

これらはどれが優れているというものではなく、目的によって使い分けるものです。 大量のデータを分析したいからCSVでもらおう、Webサイトのデータを取得したいからJSONを使おう、ツールの設定を書きたいからYAMLで保存しよう。プログラミングの中でこうした判断ができるようになると、開発がスムーズに進みます。

ファイル

前節では、データを表現するための形式について学びました。しかし、これらのデータ形式も、コンピュータの電源を切ると消えてしまうメモリの上にあるだけでは、長期的な利用ができません。

データを永続的に保存するための器が、ファイルです。

ファイルとは何か

ファイルとは、コンピュータの記憶装置（ストレージ）上に保存された、名前のついたひとまとまりのデータのことです。 普段「Wordファイル」や「画像ファイル」と呼んでいるものはすべてこれにあたります。

コンピュータの中の書類

現実世界で例えると、ファイルは書類そのものです。 そして、それらの書類を整理して入れておくバインダーや箱にあたるのが、フォルダ（ディレクトリ）です。

コンピュータは膨大な数の0と1のデータを扱いますが、それらを「ここからここまでが1つの写真」「ここからここまでは1つの文章」というように区切り、人間が管理しやすいように名前をつけたものがファイルです。

テキストファイルとバイナリファイル

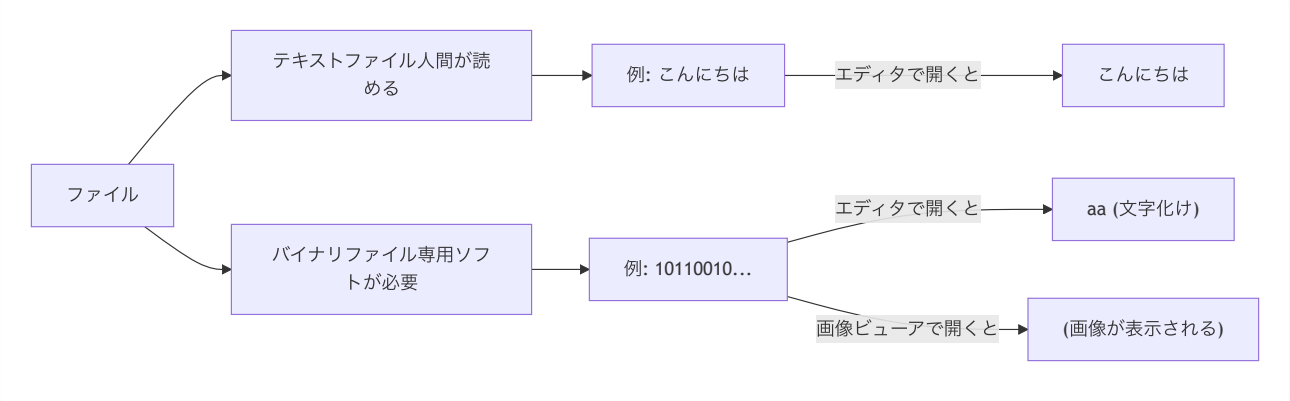
ファイルは、中身のデータの種類によって大きく2つに分けられます。前節で触れた内容をもう少し掘り下げます。

1. **テキストファイル**
    - 文字コード（UTF-8など）に従って記録されており、人間がエディタで開いて読むことができます。
    - メモ帳のデータ `.txt` 、ソースコード `.py` `.js` 、CSV、JSON、HTMLなどが該当します。
    - 行（改行）という概念があります。

2. **バイナリファイル**

- コンピュータが処理するためのデータが0と1の並びとして記録されています。専用のソフトを使わないと中身を正しく理解できません。無理やりメモ帳で開くと文字化けします。
- 画像 `.jpg` `.png` 、音声 `.mp3` 、動画 `.mp4` 、実行プログラム `.exe` などが該当します。
- データの先頭に「これは画像データです」といった識別情報（ヘッダ）が含まれていることが多いです。

ファイルの中身のイメージです。



なぜファイルが必要なのか

プログラムの中で変数を定義すればデータは扱えます。それなのに、なぜわざわざファイルとして保存する必要があるのでしょうか。主な理由は2つあります。

1. データの永続化（電源を切っても消えない）

これが最も重要な理由です。ハードウェアの章で学んだメモリとストレージの違いを思い出してください。

- **メモリ（RAM）**：プログラムが動いている間だけ使う作業机です。高速ですが、電源を切るとデータはすべて消えます（揮発性）。
- **ストレージ（HDD/SSD）**：データを保管しておく本棚や倉庫です。メモリより低速ですが、電源を切ってもデータは残ります（不揮発性）。

プログラムの実行結果（ゲームのセーブデータや作成した文書など）を次回も使いたい場合は、メモリ上のデータをストレージ上のファイルとして書き出す必要があります。これが保存（セーブ）です。

2. データの整理・管理と移動

ファイルという単位にまとめることで、データの扱いが格段に便利になります。

- 「2023年のデータ」「2024年のデータ」のように名前をつけてフォルダに分ければ、大量のデータを管理できます。
- ファイルというパッケージになっていれば、USBメモリに入れたりメールに添付したりして、他のコンピュータへデータを移動させることもできます。

ファイル入出力

プログラムがデータをファイルに保存したり、ファイルからデータを読み込んだりする操作をまとめて**ファイル入出力（I/O）**と呼びます。

前節の最後で少し触れましたが、ここではより詳しく、目的に合わせたファイルの扱い方を学んでいきます。

ファイル書き出し（Save / Write）

プログラムの中で計算した結果やユーザーが入力したデータをファイルとして保存する操作です。これを書き出し（Write）と呼びます。

書き出しを行う際、最も注意しなければならないのがモードの選び方です。Pythonなどの多くの言語では、ファイルを開く際にどういうつもりで開くかを指定します。

上書きと追記の違い

書き込みには主に2つのモードがあります。

1. **上書きモード `"w"`（Write）**
    - ファイルがまだなければ、新しく作ります。
    - すでに同じ名前のファイルがある場合、中身を一度すべて空にしてから書き込みます。古いデータは消えます。
2. **追記モード `"a"`（Append）**
    - ファイルがまだなければ、新しく作ります。
    - すでにファイルがある場合、既存の内容を残したまま末尾に新しいデータを書き足します。ログを残す場合などによく使われます。

この違いを図で示します。

ファイル読み込み（Load / Read）

保存されているファイルをプログラムに取り込み、変数として使えるようにする操作です。これを読み込み（Read）と呼びます。

ゲームのセーブデータをロードして続きから遊んだり、設定ファイルを読み込んでアプリの動作を変えたりする場面で使われます。

読み込みの流れ

読み込みの基本的な手順は書き出しと同じく `open` を使いますが、モードに `"r"`（Read）を指定します。

1. **Open**: ファイルを読み込みモードで開く。
2. **Read**: ファイルの中身を読み取って、変数に代入する。
3. **Close**: ファイルを閉じる。

ファイルが存在しない場合のエラーハンドリング

読み込み処理で最も警戒すべきなのは、読み込もうとしたファイルが存在しないケースです。 書き出しの場合はファイルがなければ勝手に作ってくれましたが、読み込みの場合は無いものは読めないので、 `FileNotFoundError` が発生してプログラムがクラッシュします。

実用的なプログラムでは、いきなり読み込むのではなく、読み込みに失敗したらどうするかという例外処理（エラーハンドリング）を記述するのが定石です。

例えば、「もしファイルがなかったら、エラーメッセージを出してプログラムを止めるのではなく、あらかじめ用意しておいたデフォルトのデータを読み込む」といった工夫をします。

演習（プログラミング問題）

この章で学んだデータ形式とファイル操作の知識を使って、実際に役立つプログラムを作ります。 今回は、コードをすべて自分一人で書くのではなく、AIにコードを書かせるという体験も行います。AIはプログラミングの強力なパートナーです。どのような指示（プロンプト）を出せばよいコードが返ってくるか、そのコツも学びましょう。

データ形式の変換

CSV形式のデータは人間が見る分には便利ですが、プログラムで複雑な処理をしたりWebアプリケーションで使ったりするにはJSON形式の方が扱いやすい場合があります。 そこで、CSVファイルを読み込んでJSONファイルに変換して保存するツールを作成します。

1. お題の設定

あなたは、あるお店の商品リストを管理しています。現在は以下のようなCSVファイルとして保存されています。

products.csv（元データ）

```
id,name,price,category
1,りんご,150,フルーツ
2,おにぎり,120,食品
3,ノート,200,文房具
4,ボールペン,100,文房具
```

これを、プログラムで扱いやすいJSON形式に変換します。

2. AIへの指示（プロンプト）

AIに対して、何をどうしたいかを明確に伝えます。以下のようなプロンプトを入力してみましょう。

> ***プロンプト例*** *Pythonで、'products.csv' という名前のCSVファイルを読み込み、それをJSON形式に変換して 'products.json' という名前で保存するプログラムを書いてください。 CSVの1行目はヘッダー（キー）として扱ってください。 また、JSONファイルは日本語が文字化けしないようにし、インデントを入れて読みやすくしてください。*

3. 生成されたコードのイメージ

AIは、以下のような「CSVを読み込んでJSONで保存する」プログラムを生成してくれます。

```python
import csv
import json

# 1. CSVファイルを読み込んで、リスト（データの並び）に変換する
data_list = []
with open("products.csv", "r", encoding="utf-8") as f:
    reader = csv.DictReader(f)
    for row in reader:
        data_list.append(row)

# 2. リストをJSON形式にしてファイルに保存する
with open("products.json", "w", encoding="utf-8") as f:
    json.dump(data_list, f, ensure_ascii=False, indent=2)
```

CSVで読み込み、辞書のリストにして、JSONで書き出す。この流れを作ることで、データ形式の変換が可能になります。

設定ファイルの読み込み

次に、設定ファイルを使ってプログラムの動作を外からコントロールする体験をしましょう。 プログラムの中身を書き換えなくても、設定ファイルを書き換えるだけで挙動が変わるようにします。

今回は、YAML形式を扱うためのライブラリ `PyYAML` を使った例を紹介します。

PythonでYAMLを扱うには、通常 `PyYAML` という外部ライブラリのインストールが必要です。 ターミナルで以下のコマンドを実行するとインストールできます。

```
pip install PyYAML
```

もしインストールが難しい場合は、読み込むファイルをJSON形式 `config.json` に変えて、標準の `json` モジュールを使っても同じことができます。

1. お題の設定

挨拶ボットプログラムを作ります。 このボットの挨拶の言葉や繰り返す回数を、プログラムを修正せずに変更できるようにします。

config.yaml（設定ファイル）

```yaml
app_name: '挨拶ボット'
settings:
  message: 'こんにちは、世界！'
  count: 3
  show_timestamp: true
```

2. コードのイメージ

設定ファイル（YAML）を読み込んで動作を変えるプログラムの例です。

```python
import yaml

# 1. 設定ファイル(config.yaml)を読み込む
with open("config.yaml", "r", encoding="utf-8") as f:
    config = yaml.safe_load(f)

# 2. 読み込んだ設定（メッセージや回数）を使って処理する
msg = config["settings"]["message"]
count = config["settings"]["count"]

for i in range(count):
    print(msg)
```

ファイルの有用性を体感する

この仕組みのポイントは、プログラムの**「ロジック（仕組み）」**と**「パラメータ（設定）」**を分離できたことです。

- エンジニアは「メッセージを繰り返す」という仕組みを作ることに集中できます。
- ユーザーは、設定ファイルを書き換えるだけで、プログラミングの知識がなくてもアプリの挙動を変えることができます。

これが、現代のソフトウェアがファイルやデータベースを重要視している大きな理由の一つです。

問題1 CSVの行数と列数

問題文

CSV形式のデータが与えられます（ヘッダー行を含む）。 このCSVデータの行数（ヘッダーを含む）と列数を出力してください。

入力データの特徴

- CSVデータは1行以上100行以下である。
- 各行は1列以上10列以下のカンマ区切りである。
- すべての行で列数は同じである。

入力

入力は以下の形式で標準入力から与えられる。

```
CSV形式のデータ（複数行）
```

出力

1行目に行数、2行目に列数を出力せよ。

入力例1

```
name,age,city
Alice,25,Tokyo
Bob,30,Osaka
```

出力例1

```
3
3
```

ヘッダーを含めて3行、3列（name, age, city）のCSVデータです。

入力例2

```
id,value
1,100
2,200
3,300
4,400
```

出力例2

```
5
2
```

5行2列のCSVデータです。

入力例3

```
item
apple
banana
```

出力例3

```
3
1
```

3行1列のCSVデータです。

問題2 JSONからの値抽出

問題文

JSON形式のオブジェクトが1行で与えられます。 指定されたキーの値を出力してください。

入力データの特徴

- JSONオブジェクトは有効な形式である。
- 指定されたキーは必ずオブジェクト内に存在する。
- 値は文字列または数値である。

入力

入力は以下の形式で標準入力から与えられる。

```
JSON文字列
キー名
```

1行目にJSONオブジェクト（1行）、2行目に抽出したいキー名が与えられる。

出力

指定されたキーの値を出力せよ。

入力例1

```
{"name":"Alice","age":25}
name
```

出力例1

```
Alice
```

キー「name」の値は「Alice」です。

入力例2

```
{"id":123,"status":"active"}
id
```

出力例2

```
123
```

キー「id」の値は123です。

入力例3

```
{"product":"Apple","price":150,"stock":50}
price
```

出力例3

```
150
```

キー「price」の値は150です。

問題3 CSVの特定列の合計

問題文

CSV形式のデータ（ヘッダー行を含む）と列番号が与えられます。 指定された列（0始まり）の数値の合計を計算して出力してください。 ヘッダー行は計算に含めません。

入力データの特徴

- CSVデータは2行以上（ヘッダー + データ1行以上）である。
- 指定された列には数値のみが含まれる。

入力

入力は以下の形式で標準入力から与えられる。

```
列番号
CSV形式のデータ（複数行）
```

1行目に列番号（0始まり）、2行目以降にCSVデータが与えられる。

出力

指定された列の数値の合計を出力せよ。

入力例1

```
1
name,score
Alice,80
Bob,90
Charlie,85
```

出力例1

```
255
```

1列目（score列）の合計は 80 + 90 + 85 = 255 です。

入力例2

```
0
id,name
1,Apple
```

```
2,Banana
3,Cherry
```

出力例2

```
6
```

0列目（id列）の合計は 1 + 2 + 3 = 6 です。

入力例3

```
2
product,quantity,price
Apple,5,100
Banana,3,80
Orange,10,120
```

出力例3

```
300
```

2列目（price列）の合計は 100 + 80 + 120 = 300 です。

第4章 プログラミング言語

プログラミング言語

言語の種類と役割

これまで、コンピュータの仕組みやデータの扱い方について学んできました。ここからは、コンピュータへの命令書であるプログラミング言語そのものについて深く学んでいきます。

世界には、名前がついているものだけでも数百種類以上のプログラミング言語が存在します。なぜ、日本語や英語のように1つの共通言語に統一されていないのでしょうか。

それは、プログラミング言語が道具だからです。 家を建てるとき、柱を切るにはノコギリ、釘を打つには金槌を使うように、作りたいものや環境によって最適なプログラミング言語は異なります。

- **Webサイトを作りたい**: JavaScript, PHP, HTML/CSS
- **人工知能（AI）を開発したい**: Python
- **スマホアプリを作りたい**: Swift（iPhone）, Kotlin（Android）
- **高速なゲームを作りたい**: C++, C#

このように、それぞれの言語には得意な分野と役割があります。全ての分野で完璧な言語は存在しません。プログラマは目的に応じて言語を選び、使い分けています。

高水準言語と低水準言語

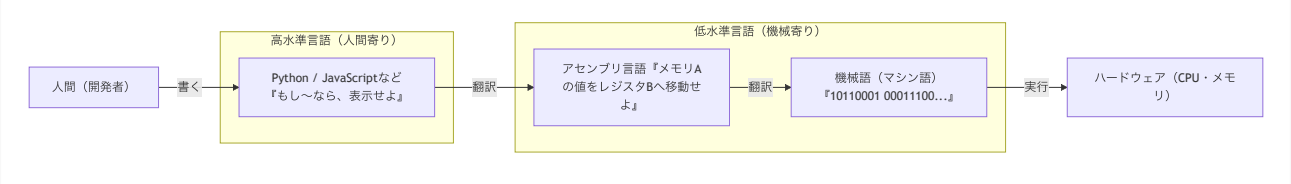
数あるプログラミング言語を分類する最も基本的なものさしに、水準（レベル）という考え方があります。 ここでいう水準とは機能の優劣ではなく、人間にとっての分かりやすさ（抽象度）を指します。

大きく分けて、高水準言語と低水準言語の2つがあります。

1. **高水準言語（High-level language）**
    - 人間の言葉（主に英語）に近い文法で書かれています。
    - 人間にとって読み書きしやすく、学習コストが比較的低いです。また、1行の命令で多くの処理を行えます。
    - コンピュータが理解できるように翻訳する処理が必要なため、実行速度がわずかに遅くなる場合があります。
    - Python, JavaScript, Ruby, Java など、現在使われているほとんどの言語がこれにあたります。
2. **低水準言語（Low-level language）**
    - コンピュータ（CPU）の機械的な動作命令に近い言葉で書かれています。
    - ハードウェア（メモリやCPU）を直接細かく制御でき、非常に高速に動作します。
    - 人間には暗号のように見え、習得が非常に難しいです。また、処理内容に対して記述量が膨大になります。
    - 機械語（マシン語）、アセンブリ言語がこれにあたります。

人間とコンピュータの距離

この関係を図で表すと、以下のようになります。 私たちが普段書くコード（高水準言語）は、何段階もの翻訳を経て、最終的にコンピュータが理解できる0と1に変換されます。



コードで見る水準の違い

実際に、同じ「文字を表示する」という処理を、高水準言語と低水準言語のイメージで比較してみます。

高水準言語（Python）の場合

人間が直感的に理解できる単語（print = 印刷する、表示する）が使われています。

```
print("Hello")
```

低水準言語（アセンブリ言語）のイメージ

データの移動先やメモリの番地などを一つひとつ指定する必要があります。CPUの種類によって書き方も変わります。 （※これは架空の例です）

```
MOV AX, DATA_SEGMENT
MOV DS, AX
MOV DX, OFFSET MESSAGE
MOV AH, 09H
INT 21H
```

機械語（マシン語）の場合

最終的にコンピュータが見ているのはこれだけです。人間がこれを直接読み書きすることは、現代ではほとんどありません。

```
10110100 00001001 10111010 ...
```

高水準言語の威力

普段私たちは高水準言語を使うことで、コンピュータの複雑な内部動作（メモリのどこにデータを置くか、CPUのどの回路を使うかなど）を気にせずに、やりたいこと（ロジック）だけに集中できています。 プログラミング学習は、まずはこの高水準言語を使って、自分のアイデアを形にする方法を学ぶことから始まります。

JavaScript（Node.js）

数あるプログラミング言語の中で、現在最も広く使われている言語の一つが**JavaScript（ジャバスクリプト）**です。 名前が似ているJavaという言語がありますが、これらはまったく別の言語です（メロンとメロンパンくらい違います）。ここでは、現代のソフトウェア開発において中心的な役割を果たすJavaScriptと、その活動領域を広げたNode.jsについて学びます。

JavaScriptの用途

Webブラウザで動く唯一の標準言語

私たちが普段見ているWebサイトやWebアプリケーションは、主に3つの技術で構成されています。

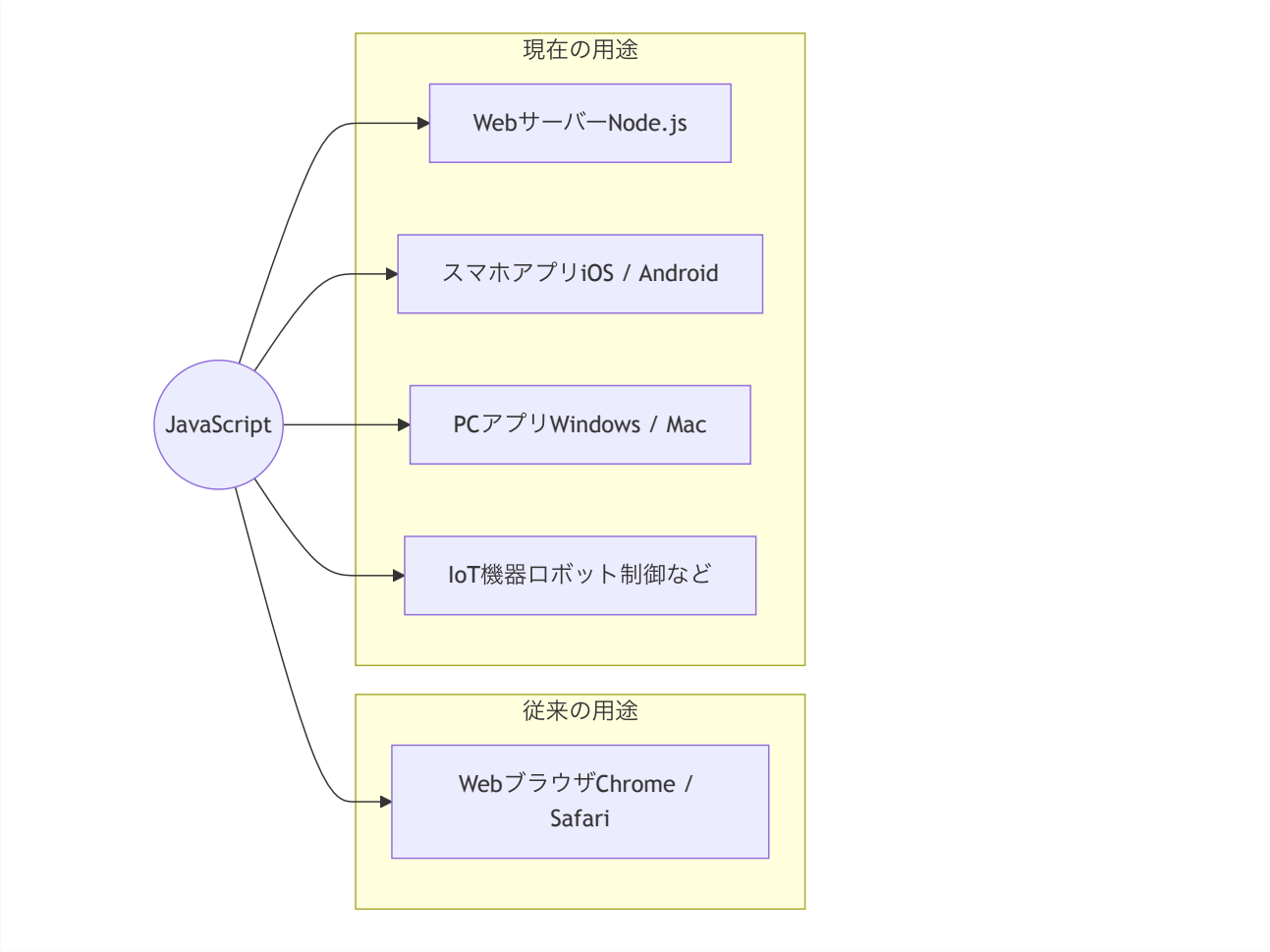
1. **HTML**: ページの骨組みを作ります（見出し、文章、画像など）。
2. **CSS**: ページの見た目を整えます（色、レイアウト、フォントなど）。
3. **JavaScript**: ページに動きをつけます（ポップアップ、アニメーション、データのやり取りなど）。

この中で、JavaScriptはWebブラウザ（Chrome, Safari, Edgeなど）の上で動作する唯一の標準プログラミング言語です。Googleマップで地図をドラッグして動かせるのも、Gmailでページを移動せずにメールが届くのも、すべてJavaScriptがブラウザの中で動いているおかげです。Webエンジニアを目指すなら、避けては通れない言語です。

サーバーサイドやアプリ開発への広がり

かつてJavaScriptはWebブラウザの中だけで動く、Webページ専用の言語でした。しかし現在では、その活躍の場はブラウザの外にも大きく広がっています。

- **サーバーサイド開発**: 後述するNode.jsの登場により、Webサーバーの裏側の処理もJavaScriptで書けるようになりました。
- **スマホアプリ開発**: React Nativeなどの技術を使えば、JavaScriptでiPhoneやAndroidのアプリを作ることができます。
- **デスクトップアプリ開発**: Electronという技術を使えば、VS CodeやSlackのようなPC用アプリもJavaScriptで作れます。



この図は、かつてはWebブラウザの中だけで使われていたJavaScriptが、現在ではサーバー、スマートフォン、PC、さらにはIoT機器まで、あらゆる場所で動作するようになっていることを示しています。

JavaScriptの歴史

誕生と進化

JavaScriptは、1995年にNetscape Communications社のBrendan Eich（ブレンダン・アイク）氏によって開発されました。驚くべきことに、そのプロトタイプはわずか10日間で作られたと言われています。 当初は簡単な動きをつけるための補助的な言語でしたが、Googleマップ（2005年）の登場などで「ブラウザ上で高度なアプリケーションが動く」ことが証明され、急速に進化しました。

標準規格「ECMAScript」

JavaScriptが普及するにつれ、「ブラウザAでは動くけど、ブラウザBでは動かない」という互換性の問題が発生しました。そこで、言語の仕様を統一するための標準規格が作られました。それが**ECMAScript（エクマスクリプト）**です。

- **ECMAScript**: 言語の仕様書（ルールブック）です。
- **JavaScript**: ECMAScriptの仕様に基づいて実装された実際のプログラミング言語です。

普段「JavaScript」と呼んでいるのは、実質的にはこのECMAScriptのバージョンのことです（例としてES2015, ES6など）。この規格によって毎年新しい機能が追加され、言語として成長し続けています。

Node.jsとは何か

ブラウザの外へ飛び出したJavaScript

2009年、Ryan Dahl（ライアン・ダール）氏によって**Node.js（ノード・ジェイエス）**が開発されました。 これは、ブラウザの外（OS上）でJavaScriptを動かすための実行環境（ランタイム）です。

これまでブラウザの中に閉じ込められていたJavaScriptのエンジン（V8エンジン）を取り出し、WindowsやmacOSなどのOS上で直接動かせるようにしたものです。

革命的な意義

Node.jsの登場は、Web開発の世界に革命をもたらしました。

1. **言語の統一**: フロントエンド（ブラウザ側）もバックエンド（サーバー側）も、JavaScriptというたった一つの言語で開発できるようになりました。学習コストが下がり、開発効率が劇的に向上しました。
2. **強力なツール**: CLIツール（コマンドラインツール）を作成したり、パソコン内のファイルを操作したりすることがJavaScriptで可能になりました。

Node.jsでの実行例

Node.jsを使うと、以下のようなJavaScriptプログラムをパソコン上で直接動かせます。

```
// JavaScriptの例
const language = 'Node.js';
console.log(`こんにちは、${language} の世界へ！`);
```

TypeScript

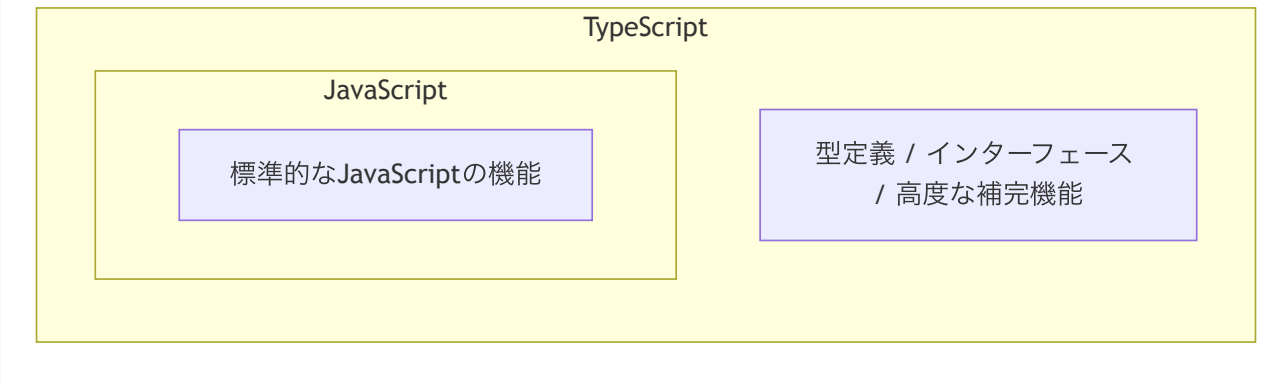
JavaScriptは非常に自由で強力な言語ですが、その自由さゆえに、開発者が意図しないミス（バグ）を犯しやすいという側面も持っています。その課題を解決し、より安全に効率よく開発するために生まれたのが**TypeScript（タイプスクリプト）**です。

JavaScriptとの関係

JavaScriptに型の概念を加えたスーパーセット

TypeScriptは、Microsoftによって開発された言語です。最大の特徴は、JavaScriptに**型（Type）**という概念を追加したことです。

TypeScriptはJavaScriptの**スーパーセット（上位互換）**と呼ばれます。JavaScriptで書かれたコードは、そのままTypeScriptのコードとしても正しく動作します。JavaScriptという基礎の上に、便利な追加機能や安全装置（型）が乗っているイメージです。



この図は、TypeScriptがJavaScriptを完全に内包していることを示しています。JavaScriptのすべての機能を使いつつ、TypeScript独自の強力な機能を追加で利用できるという関係性です。

コンパイル（トランスパイル）するとJavaScriptになる

コンピュータやWebブラウザは、実はTypeScriptをそのまま直接実行することはできません。TypeScriptを実行するには、一度JavaScriptの形式に変換する必要があります。

この変換作業を**コンパイル**（またはトランスパイル）と呼びます。

1. **書く**: 開発者がTypeScriptでコードを書く。

2. **変換**: コンパイラというツールが、コードをチェックしながらJavaScriptへ変換する。
3. **動かす**: 変換されたJavaScriptを、ブラウザやNode.jsが実行する。

結局JavaScriptになるなら最初からJavaScriptで書けばいいのでは、と思うかもしれません。しかし、この変換するタイミングでコードのミスを厳しくチェックしてくれることが、開発者にとって非常に大きな助けになるのです。

型のメリット

エラーを事前に発見できる

JavaScript（動的型付け言語）では、プログラムを実際に動かしてみるまでエラーに気づけないことがよくあります。数値が入っているはずの変数にうっかり文字を入れてしまい、計算結果が `NaN`（Not a Number）になって初めて気づく、といったケースです。

TypeScript（静的型付け言語）では、変数に「この箱には数値しか入れません」というルール（型）を指定します。もし間違った種類のデータを入れようとすると、プログラムを動かす前、コードを書いている最中にエディタが赤線を引いて警告してくれます。

大規模な開発とチーム開発の必須ツール

プログラミングは、一人の人間がすべてを把握できる小さな規模から、何百人もの開発者が協力して数百万行のコードを書く巨大なシステムまで様々です。

大規模な開発において、TypeScriptは絶大な威力を発揮します。

- **ドキュメント代わりになる**: この関数にはどんなデータが必要で、何が返ってくるのかがコード自体に明記されるため、他人が書いたコードでも使い方が一目でわかります。
- **自動補完が強力**: エディタがデータの形を正確に把握しているため、次に何を打つべきかを完璧に予測して提案してくれます。
- **修正が怖くない**: コードの一部を変更したとき、その影響で動かなくなる場所があれば、コンパイラが即座にすべて教えてくれます。

TypeScriptの威力を体験する

以下のコードは、TypeScriptの特徴である型指定の効果を擬似的に体験するためのものです。JavaScriptと比較しながら、どのようにエラーを防げるのかを見てみます。

```
// TypeScriptの例（数値型を指定）
function add(a: number, b: number) {
  return a + b;
}

// 数値以外を渡そうとすると、動かす前にエラーを教えてくれる
// add("1", "2");
```

現代の開発現場では、TypeScriptを使わない理由がないと言われるほど普及しています。これから学ぶ皆さんも、型の守られた世界で開発することで、よりスムーズにプログラミングを習得できるはずです。

Python

JavaScriptがWebの世界で王者であるならば、**Python（パイソン）**はデータサイエンスとAIの世界における絶対的な王者です。プログラミング言語の人気ランキングで常にトップを争うこの言語について、その特徴と人気の理由を学びます。

Pythonの用途

データ分析、AI・機械学習、Web開発など多目的に活躍

Pythonは、非常に汎用性の高い多目的プログラミング言語です。特定の分野に縛られることなく、あらゆる場面で活躍しています。

- **データ分析・統計**: 膨大なビジネスデータから売り上げの傾向を分析したり、グラフを描画したりする作業が得意です。
- **AI・機械学習**: ChatGPTのようなAIモデルの開発や、画像認識、自動翻訳などの最先端技術は、ほとんどがPythonで作られています。
- **Web開発**: InstagramやYouTube、Dropboxといった有名なWebサービスの裏側（サーバーサイド）でもPythonが使われています。
- **業務自動化（スクリプト）**: Excelのデータを集計してメールを送るといった毎日の単純作業を自動化する小さなプログラム作りにも最適です。

初学者に最も優しい言語

Pythonがこれほど人気な最大の理由は、文法がシンプルで圧倒的に読みやすいことです。

他の言語ではカッコやセミコロンなどの記号をたくさん書かなければならない処理でも、Pythonなら英単語を並べるような感覚で、短くスッキリと書くことができます。 そのため、プログラミングの学習を始めるならまずはPythonから、と推奨されることが多いです。

Pythonの歴史

読みやすさを重視する哲学（The Zen of Python）

Pythonは1991年、オランダ人のグイド・ヴァン・ロッサム氏によって公開されました。 開発当初から一貫している哲学があります。コードは書かれる時間よりも、読まれる時間の方が圧倒的に長い、という考え方です。

プログラムは一度書いたら終わりではなく、後から自分や他人が何度も読み返し、修正や機能追加を行います。そのとき、複雑で難解なコードはバグの温床になります。 Pythonには**The Zen of Python（Pythonの禅）**と呼ばれる設計指針があり、その中には以下のような言葉があります。

> *Simple is better than complex.（複雑であるより、シンプルであるほうが良い。） Readability counts.（読みやすさは重要である。）*

この哲学のおかげで、誰が書いても似たような、読みやすいコードになりやすいのがPythonの大きな強みです。

強力なライブラリのエコシステム

Python自体はシンプルな言語ですが、その周りには世界中の開発者が作ったライブラリ（便利な道具箱）が豊富に揃っています。これが、現在のAIブームを支えている正体です。

- **NumPy（ナムパイ）**: 高度な数学計算を一瞬で行うライブラリです。
- **Pandas（パンダス）**: 表形式のデータを自由自在に加工・集計するライブラリです。
- **PyTorch / TensorFlow**: AI（ディープラーニング）を作るための専門的なライブラリです。

研究者やデータサイエンティストたちは、ゼロからプログラムを作るのではなく、これらの強力なライブラリをPythonという接着剤で組み合わせることで、最先端の研究成果をすぐに形にできるのです。

Pythonのシンプルさ

Pythonでは、やりたいことを非常に短いコードで、英語を読むような感覚で記述できます。

```
# Pythonの例：70点以上の人だけを表示する
scores = [65, 80, 92, 55, 78]
passed = [s for s in scores if s >= 70]
print(f"合格者: {passed}")
```

代表的なその他の言語

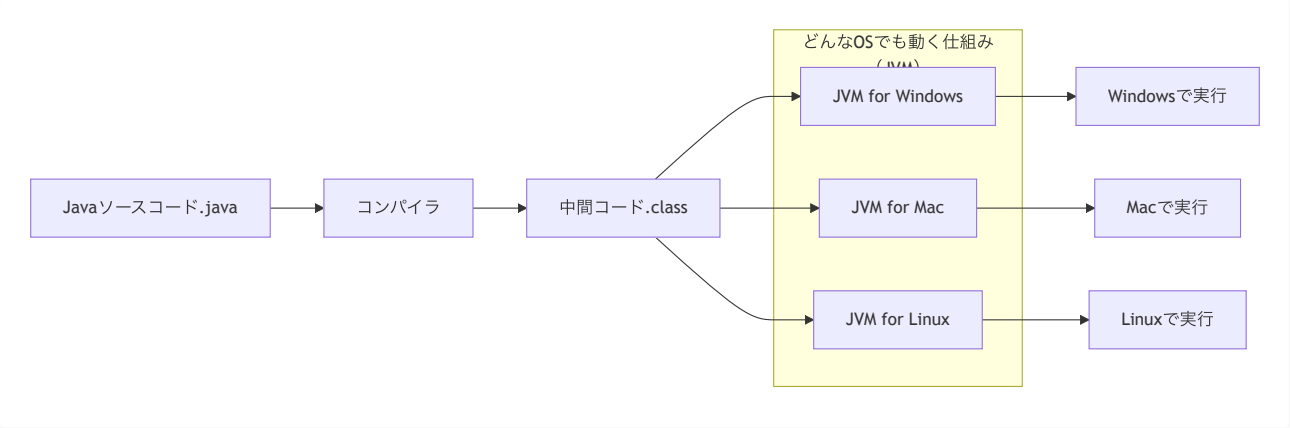
これまで、WebやAIの分野で活躍する現代的な言語（JavaScript, Python）を見てきました。しかし、プログラミングの世界には、それらとは異なる歴史や強みを持つ重要な言語がまだまだあります。 ここでは、特に企業のシステム開発やコンピュータの基礎を支える分野で使われている2つの代表的な言語を紹介します。

Java

特徴

「一度書けば、どこでも動く（Write Once, Run Anywhere）」

これは、Java（ジャバ）が登場したときの有名なキャッチコピーです。 通常、コンパイルが必要な言語（C言語など）は、Windows用にコンパイルしたプログラムはMacでは動きません。しかしJavaは、**Java仮想マシン（JVM）**という仕組みを導入することで、OSに関係なく同じプログラムを動かすことを実現しました。



Javaのプログラムは、まず中間コードという共通の形式に変換されます。それぞれのOS専用のJava仮想マシン（JVM）がその中間コードを翻訳して実行するため、プログラマはOSごとの違いを気にする必要がありません。

堅牢なオブジェクト指向言語

Javaはオブジェクト指向という考え方を厳密に取り入れています。システムを部品（オブジェクト）の組み合わせとして作る考え方です。 また、TypeScriptのように厳格な静的型付けを採用しており、エラーが起きにくく、大人数での開発でもコードが壊れにくいという特徴があります。この頑丈さ（堅牢性）が、Javaの最大の武器です。

用途

その信頼性の高さから、絶対に止まってはいけないシステムでよく選ばれます。

- **大規模システム**: 銀行のATMシステム、証券取引システム、大企業の基幹業務システムなど。
- **Androidアプリ**: 長い間、Androidアプリ開発の標準言語でした。現在はKotlinも普及していますが、Javaも現役です。
- **Webサービス**: Twitter（現X）などの大規模なWebサービスの裏側でも、処理速度と安定性を求めてJavaが使われています。

Javaのコード例

Javaは記述量が多いですが、その分ルールが厳格で、大規模な開発でも壊れにくい頑丈なプログラムが作れます。

```
// Javaの例
public class Main {

  public static void main(String[] args) {
    String name = "山田";
    System.out.println(name + "さんの口座です");
  }
}
```

C言語 / C++

特徴

コンピュータの生の力を引き出す

C言語（シー言語）は、1972年に開発された非常に歴史ある言語です。C++（シープラスプラス）は、C言語に機能を追加して拡張した言語です。 これらの言語の最大の特徴は、ハードウェアを直接制御できることです。

PythonやJavaは、メモリ管理などを言語自体が自動でやってくれます（ガベージコレクション機能など）。しかしC言語では、プログラマ自身がメモリの確保や解放を管理する必要があります。 これは部屋の片付けに例えられます。

- **Python/Java**: 自動お掃除ロボットが勝手に片付けてくれます。楽ですが、細かい融通は利きません。
- **C/C++**: 自分で一つひとつ片付けます。手間はかかりますが、極限まで効率的に収納できます。

このため習得難易度は高いですが、使いこなせればコンピュータの性能限界まで引き出した爆速のプログラムを作ることができます。

用途

速さと軽さが求められる場所で、他の言語の追随を許しません。

- **OS（オペレーティングシステム）**: Windows、macOS、LinuxといったOS自体が、主にC/C++で作られています。
- **組み込みシステム**: 家電（炊飯器、エアコン）、自動車のエンジン制御、ロボットなど、使えるメモリが極端に少ない機械の中で動くプログラムです。
- **ハイエンドゲーム**: プレステやSwitchなどの家庭用ゲーム機や、高画質な3Dゲームの描画エンジンなど、一瞬の遅延も許されない世界です。

C言語のコードを見てみる

C言語のコード例を見てみます。一見シンプルに見えますが、メモリ上の場所（アドレス）やサイズを意識して書かれています。

```
// C言語の例
#include <stdio.h>
int main() {
    int num = 100;
    // &num で、データがメモリの「何番地」にあるかを表示できる
    printf("住所: %p\n", &num);
    return 0;
}
```

C言語やC++を学ぶことは、単にコードを書くこと以上に、コンピュータが裏側で実際に何をしているのかを深く理解する旅となります。

言語比較の実践

「1から100までを足し算する」という同じ処理を、PythonとJavaScriptで比較してみましょう。

1. Pythonの書き方

```
total = 0
for i in range(1, 101):
    total = total + i
print(total)
```

2. JavaScriptの書き方

```
let total = 0;
for (let i = 1; i <= 100; i++) {
  total = total + i;
}
console.log(total);
```

3. 違いを比較する

2つのコードを見比べてみます。

項目	Python	JavaScript
変数の宣言	`total = 0` (型推論のみ)	`let total = 0;` (宣言が必要)
繰り返しの範囲	`range(1, 101)` (関数を使う)	`i = 1; i <= 100` (条件式を書く)
ブロックの囲み	インデント（字下げ）で表現	波括弧 `{ }` で表現
画面出力	`print(...)`	`console.log(...)`

Pythonはシンプルさを追求しており、インデント（行頭の空白）で処理のまとまりを表現します。JavaScriptは記号（カッコやセミコロン）が多く、より厳密な構造を持っています。

すべての言語を暗記する必要はありません。AIに「このPythonコードをJavaScriptに書き換えて」と頼めば、一瞬で翻訳してくれます。大切なのは、それぞれの言語が持つ「得意分野」を知っておくことです。

4. AIによる翻訳体験（バイブコーディング）

プログラミング言語の文法は似ている部分も多いですが、細かいルールの違いをすべて暗記するのは大変です。 そこで、生成AIの力を借りてみましょう。AIは、あるプログラミング言語を別の言語に翻訳するのが非常に得意です。

演習手順

1. 普段使っているAIチャット（ChatGPT、Claude、Copilotなど）を開きます。
2. 先ほどのPythonのコードをコピーします。
3. 以下のようなプロンプト（指示）と一緒にAIに送信します。

> ***プロンプト例*** *以下のPythonコードと同じ動作をするプログラムを、JavaScriptで書いてください。また、それぞれの行が何をしているか、コメントで詳しく解説してください。*
>
> *（ここにPythonコードを貼り付ける）*

4. AIが出力したJavaScriptコードを、実際に自分の環境（エディタ）にコピー＆ペーストして実行してみましょう。

体験のポイント

- AIが `print` を `console.log` に、 `for ... in` を `for (...)` に正しく変換できているか確認してください。
- もし「C言語で書いて」や「Javaで書いて」と頼んだらどうなるでしょうか。興味があれば試してみてください。

すべての言語の文法を完璧に暗記する必要はありません。やりたい処理さえ明確であれば、AIという優秀な通訳者が最適な言語に翻訳してくれます。

問題1 偶数の抽出（Python）

問題文

この問題では **Python** を使用してプログラムを作成してください。提出ファイルは `main.py` です。

N個の整数が与えられます。 その中から偶数のみを抽出し、入力された順番で出力してください。

入力データの特徴

- $1 \leq N \leq 100$
- 各整数は-1000以上1000以下である。

- 少なくとも1つは偶数が含まれる。

入力

入力は以下の形式で標準入力から与えられる。

```
N
A_1 A_2 ... A_N
```

1行目に整数の個数N、2行目にスペース区切りでN個の整数が与えられる。

出力

偶数のみをスペース区切りで1行に出力せよ。

入力例1

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

出力例1

```
2 4
```

1, 3, 5は奇数なので除外し、2と4のみを出力します。

入力例2

```
6
10 21 32 43 54 65
```

出力例2

```
10 32 54
```

入力例3

```
3
0 -2 4
```

出力例3

```
0 -2 4
```

0は偶数として扱います。負の偶数も含めます。

問題2 1からNまでの合計（JavaScript）

問題文

この問題では **JavaScript** を使用してプログラムを作成してください。提出ファイルは `main.mjs` です。

正の整数Nが与えられます。 1からNまでの整数の合計を計算して出力してください。

入力データの特徴

- $1 \leq N \leq 10000$

入力

入力は以下の形式で標準入力から与えられる。

```
N
```

出力

1からNまでの合計を出力せよ。

入力例1

```
5
```

出力例1

```
15
```

1 + 2 + 3 + 4 + 5 = 15 です。

入力例2

```
10
```

出力例2

```
55
```

1 + 2 + ... + 10 = 55 です。

入力例3

```
100
```

出力例3

```
5050
```

1から100までの合計は5050です。

問題3 FizzBuzz（TypeScript）

問題文

この問題では **TypeScript** を使用してプログラムを作成してください。提出ファイルは `main.ts` です。

正の整数Nが与えられます。 1からNまでの各整数について、以下のルールに従って出力してください。

- 3の倍数かつ5の倍数なら「FizzBuzz」
- 3の倍数なら「Fizz」
- 5の倍数なら「Buzz」
- それ以外はその数字をそのまま出力

各出力は1行ずつ出力してください。

入力データの特徴

- $1 \leq N \leq 100$

入力

入力は以下の形式で標準入力から与えられる。

```
N
```

出力

1からNまでの各整数について、ルールに従って1行ずつ出力せよ。

入力例1

```
5
```

出力例1

```
1
2
Fizz
4
Buzz
```

入力例2

```
15
```

出力例2

```
1
2
Fizz
4
Buzz
Fizz
7
8
Fizz
Buzz
11
Fizz
13
14
FizzBuzz
```

15は3の倍数かつ5の倍数なので「FizzBuzz」となります。

第5章 アルゴリズム

アルゴリズム

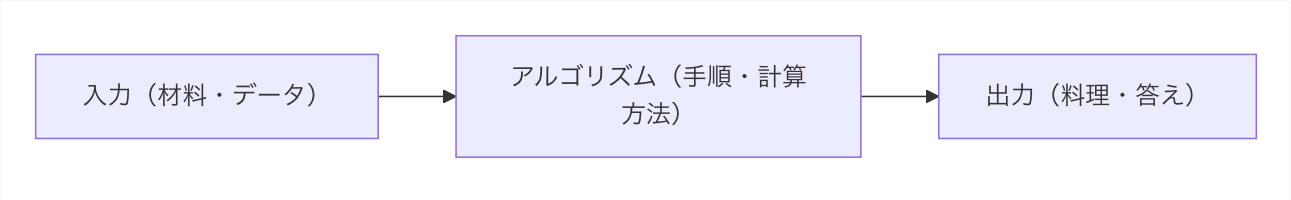
アルゴリズムとは何か

やり方のレシピ

「アルゴリズム（Algorithm）」という言葉を聞くと、高度な数学やAIの専門用語のように聞こえるかもしれません。しかし、その本質はとてもシンプルです。

アルゴリズムとは、ある問題を解決するための明確で具体的な手順のことです。

もっと平たく言えば、やり方やレシピのことです。 私たちは、入力を受け取り、それを何らかの処理（アルゴリズム）に通すことで、欲しい結果（出力）を得ています。



- **入力**: 解きたい問題や、材料となるデータ。
- **アルゴリズム**: 正解にたどり着くための計算手順やロジック。
- **出力**: 手順通りに実行した結果得られる答え。

プログラミングにおいてアルゴリズムを学ぶということは、単にコードの書き方を覚えるだけではありません。どういう手順で計算すれば正しく効率的に答えが出せるか、というロジックの組み立て方を学ぶことを意味します。

手順を考える重要性

「とりあえず書く」は失敗のもと

プログラミング初心者がやりがちな失敗の一つに、パソコンに向かっていきなりコードを書き始めてしまうことがあります。 設計図なしで家を建て始めたり、地図を見ずに知らない土地を歩き始めたりするようなものです。

手順（アルゴリズム）を先に考えることが重要な理由は2つあります。

1. **結果が間違ってしまうから** プログラミング言語の文法がどれだけ正しくても、書かれている内容（手順）が間違っていれば、正しい結果は得られません。 たとえば「カレーを作る」という目的のために、「野菜を煮る」「野菜を切る」という順番でプログラムを書いてしまったら、文法的にエラーは出なくても、出来上がるのは茹でられた丸ごとの野菜です。カレーにはなりません。

2. **効率が悪くなるから** 行き当たりばったりでコードを書くと、無駄な処理が増えたり、後から修正が大変になったりします。事前に「まずAをして、次にBをして、もしCならDをする」と整理しておくことで、最短ルートで完成に近づけます。

身近な例（料理・道順）

アルゴリズムはコンピュータの中だけでなく、日常生活の中にも溢れています。

料理のレシピ

料理のレシピは、まさにアルゴリズムそのものです。

- **入力**: 野菜、肉、カレールー、水
- **アルゴリズム**
    1. 野菜と肉を一口大に切る。
    2. 鍋で具材を炒める。
    3. 水を加えて15分煮込む。

4. 火を止めてルーを溶かす。
5. とろみがつくまで弱火で煮込む。

- **出力**: カレーライス

この手順のどれか一つでも抜けたり、順番が入れ替わったりすると、美味しいカレーにはなりません。

目的地への道順

「駅からレストランまで行く」という問題も、アルゴリズムで解決しています。

- **入力**: 現在地（駅）、目的地（レストラン）
- **アルゴリズム**
  1. 改札を出て北口に向かう。
  2. 大通りを直進し、2つ目の信号まで歩く。
  3. 信号を右に曲がる。
  4. 50メートル進んだ左手にある建物に入る。
- **出力**: レストランへの到着

人間の言葉からコンピュータの言葉へ

人間相手なら「適当に炒めて」や「いい感じに味付けして」で通じるかもしれませんが、コンピュータは曖昧な命令を一切理解できません。 コンピュータにアルゴリズムを伝えるには、条件分岐（もし〜なら）や繰り返し（〜になるまで）を使って、徹底的に具体化する必要があります。

例えば、「お湯を沸かす」という動作をコンピュータに指示する場合、以下のような手順に分解します。

1. **繰り返しの条件**:「水温が100度になるまで」以下の作業を繰り返す。
2. **処理**: 火力を強める（水温を上げる）。
3. **判断（もし〜なら）**:「もし水温が100度に達したら」、火を止める。

日常の何気ない手順を数値、条件、繰り返しに分解して再構築することが、アルゴリズムを考える第一歩です。

有名なアルゴリズム（探索と整列）

アルゴリズムの世界には、王道と呼ばれる有名な問題と解決法があります。 その代表格が、特定のデータを見つける**探索（サーチ）**と、データを順番に並べる**整列（ソート）**です。 これらは、Web検索エンジンからECサイトの商品一覧まで、あらゆるシステムの基礎となって動いています。

探索アルゴリズム（線形探索・二分探索）

データの中から目的の値を見つける

探索とは、たくさんのデータの中から、自分が探している特定のデータ（ターゲット）を見つけ出すことです。 トランプの山からハートのエースを探す、電話帳から友人の番号を探す、といった行為です。

ここでは、最も基本的な2つの方法を紹介します。

1. **線形探索（リニアサーチ）**: 端から順番にしらみつぶしに探す方法。
2. **二分探索（バイナリサーチ）**: データの真ん中を見て、探す範囲を半分に絞り込んでいく方法。

線形探索: しらみつぶしの力技

一番単純な方法は、データの先頭から一つずつ順番に「これですか？」「違います」「じゃあ次は？」を繰り返していくことです。 これを線形探索と呼びます。

- データがバラバラに並んでいても使えます。
- データ量が増えると、探す時間もそのまま比例して増えます。

二分探索: 賢い絞り込み

もしデータが小さい順（または大きい順）に綺麗に並んでいる（ソートされている）なら、もっと効率的な方法が使えます。それが二分探索です。

辞書を引くとき、いきなり1ページ目から順に見る人はいないでしょう。大体のあたりをつけて真ん中を開き、目的の単語より前か後ろかを判断して、探す範囲を一気に半分にします。これを繰り返すのが二分探索です。

以下の図で、1から9までの数字の中から「7」を探す手順を比較してみます。



- データ量が爆発的に増えても、二分探索なら探す回数はあまり増えません（非常に高速）。
- ただし、データが事前に整列（ソート）されていないと使えません。

データ量が増えたときの影響

この2つのアルゴリズムの性能差は、データ量が増えれば増えるほど顕著になります。

- **データが100個のとき**
  - 線形探索: 最大100回チェックが必要です（運が悪ければ）。
  - 二分探索: 最大でも7回程度で特定可能です。
- **データが100万個のとき**
  - 線形探索: 最大100万回。
  - 二分探索: たったの20回程度。

100万回の作業と20回の作業では、完了までの時間は天と地ほどの差があります。適切なアルゴリズムを選ぶべき理由がここにあります。

整列アルゴリズム（ソート）

データを整理整頓する

探索を効率よく行うため（二分探索を使うため）にも、また人間がデータを見やすくするためにも、データをある規則（小さい順など）に従って並べ替えることが重要です。これを**整列（ソート）**と呼びます。

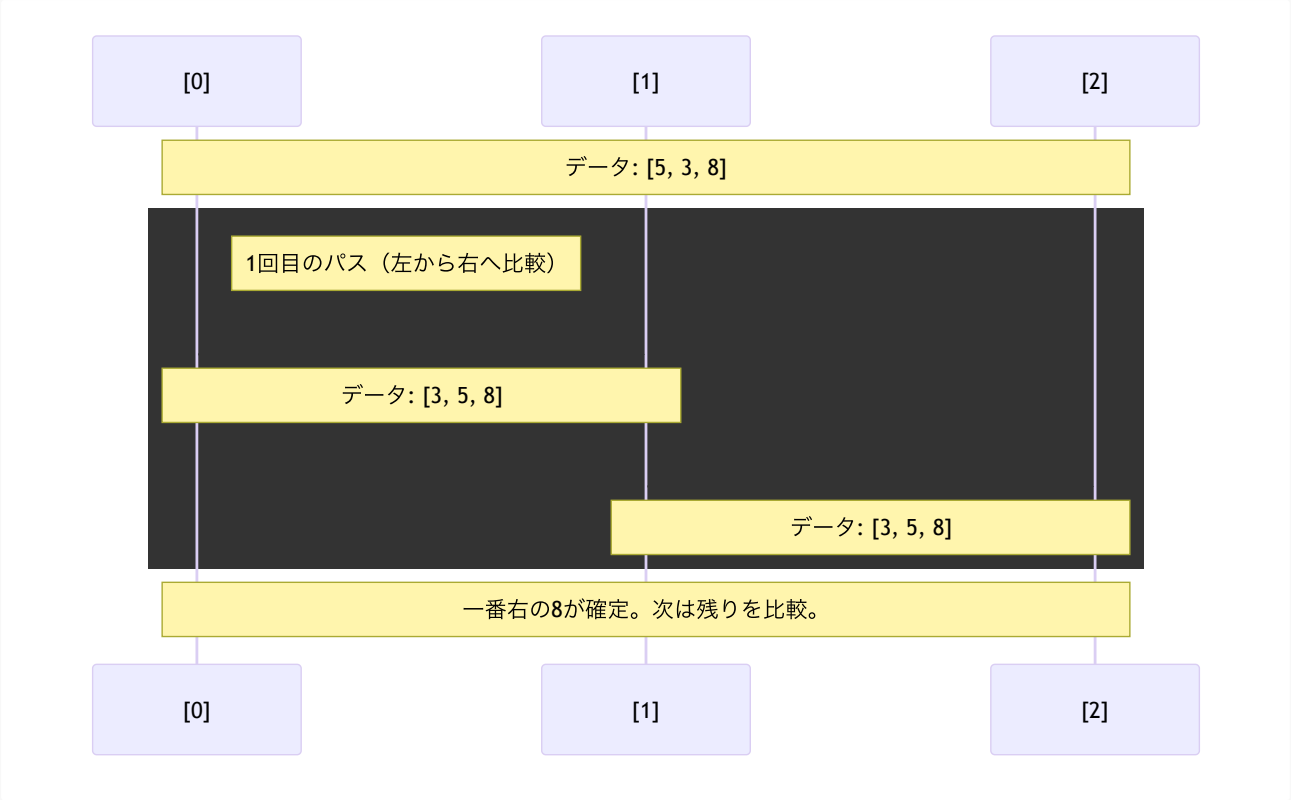
ソートにもたくさんのアルゴリズムが存在しますが、ここでは代表的なものを紹介します。

1. **バブルソート**: 単純だが遅い。
2. **クイックソート**: 複雑だが速い。

バブルソート: 基本の並べ替え

隣り合う2つのデータを比較し、順番が逆なら入れ替える、という操作を端から端まで何度も繰り返す方法です。 データの列の中で、大きな値が泡（バブル）のように徐々に後ろ（右側）へ移動していく様子から名付けられました。

仕組みが単純で理解しやすい反面、データ量が増えると急激に時間がかかるようになるため、大量データの処理には向きません。



クイックソート: 高速な並べ替え

基準となる値（ピボット）を一つ適当に決め、それより小さいグループと大きいグループにデータを分割します。そして、それぞれのグループの中でまた同じことを繰り返します。

分割統治法という考え方を使っており、非常に高速です。現代の多くのプログラミング言語の標準ソート機能（ `sort()` など）は、このクイックソートやその改良版を採用していることが多いです。

プログラムの効率（時間計算量、空間計算量）

処理時間の違い

「動けばいい」だけではない

プログラミングを始めたばかりの頃は、エラーが出ずに動いたというだけで大きな達成感があります。それはとても素晴らしいことです。しかし、プログラミングの世界にはもう一つ上のレベルがあります。それが効率です。

同じ結果（出力）を出すプログラムであっても、その中身（アルゴリズム）の書き方によって、処理が終わるまでの時間が1秒で済むのか、100年かかってしまうのか、という極端な差が生まれることがあります。

無駄な繰り返しを避ける

コンピュータは計算が非常に速いので、データが少ないうちは効率の悪い書き方をしていても気づきません。しかし、扱うデータが増えた途端に、その差が牙を剥きます。

以下の例を見てみます。「1からNまでの数字を全部足す」という単純な問題を考えます。 （例として、1から100まで足すと答えは5050になります）

これを2つの異なるアプローチで解いてみます。

1. **地道に足す方法（ループ）**: 1+2+3... と順番に計算する。
2. **公式を使う方法（数学）**: 等差数列の和の公式 $(1+N) \times N / 2$ を使う。

もし1億までの数字を足す場合、地道に足すとコンピュータでも数秒かかりますが、公式を使えば一瞬（0.0000秒レベル）で終わります。 どう計算するか（アルゴリズム）の選び方一つで、パフォーマンスは劇的に変わるのです。

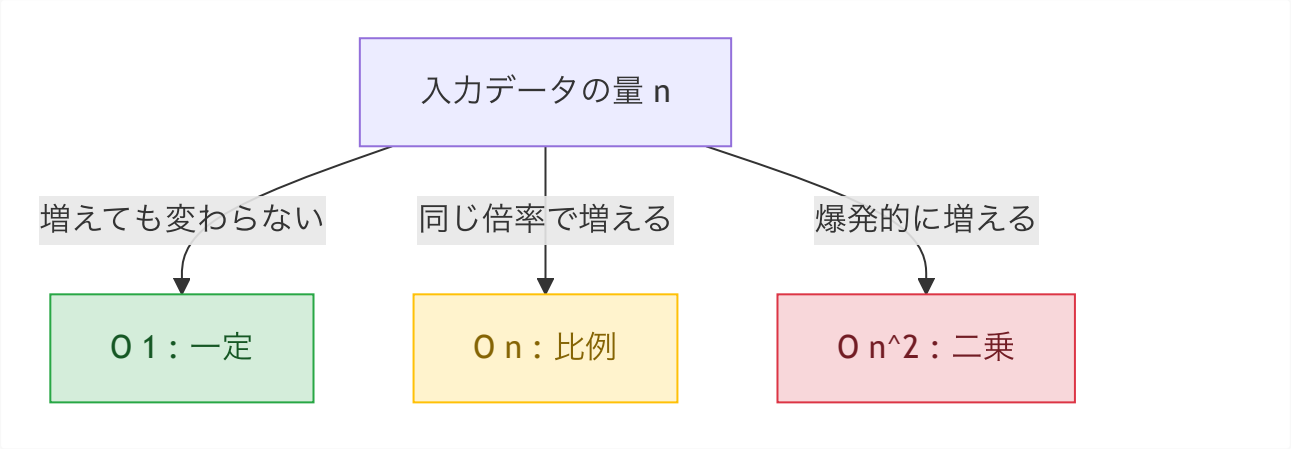
データ量が増えたときの影響（オーダー記法）

計算回数の増え方を測るモノサシ

プログラムの効率を議論するとき、具体的な「何秒」という時間はあまり使いません。使うコンピュータの性能によって変わってしまうからです。 その代わりに、入力データ（N）が増えたとき計算回数がどれくらいの勢いで増えるか、という変化の割合を使います。

これを表すのが**オーダー記法（ビッグオー記法）**です。 数学的な厳密さは一旦置いておいて、感覚的に理解しましょう。

よく出てくる3つのパターンを紹介します。



1. **$O(1)$（オー・ワン）：とても速い**
    - データが1個でも1億個でも、計算にかかる手間は一定で変わりません。
    - 先ほどの等差数列の和の公式や、配列の3番目のデータにアクセスする処理がこれにあたります。
2. **$O(n)$（オー・エヌ）：普通**
    - データが2倍になれば、計算時間も2倍になります（比例する）。
    - 先ほどのループで足し算する処理や、リストの中から最大値を探す処理がこれにあたります（全部見ないとわからない）。
3. **$O(n^2)$（オー・エヌ2乗）：遅い（注意が必要）**
    - データが2倍になれば計算時間は4倍、データが10倍になれば100倍になります。
    - 二重ループ（ループの中でさらにループする処理）やバブルソートがこれにあたります。

$O(n)$ と $O(n^2)$ の違いを体感する

例えば、データの数が1000個あるとします。

- **$O(n)$ の場合**: 1,000回の計算で終わります。
- **$O(n^2)$ の場合**: 1,000,000回（100万回）の計算が必要になります。

データ量が10倍になると、$O(n^2)$ の計算回数は100倍に膨れ上がります。 アルゴリズムを考えるときは、なるべく $O(n^2)$ を避けて、より効率的な方法にできないかと考える癖をつけることが重要です。

効率を意識する理由

なぜ、そんなに急ぐのか

「今のコンピュータは高性能なんだから、少しくらい効率が悪くてもいいじゃないか」と思うかもしれません。 確かに、個人で使う小さなツールなら問題にならないことも多いです。しかし、以下の理由からプロは常に効率を意識します。

1. **扱うデータ量が巨大だから** GoogleやAmazonのようなシステムは、数億、数十億というデータを扱います。検索ボタンを押して結果が出るまでに10秒もかかったら、誰も使ってくれません。非効率なアルゴリズムは、サービスの死を意味しま

す。

2. **システムダウンを防ぐため** アクセスが集中したとき、効率の悪いプログラムはCPUやメモリを食いつぶし、サーバーを停止させてしまう原因になります。

3. **コストと環境への配慮** 計算に時間がかかるということは、それだけ電気を使います。クラウドサーバー（AWSなど）を利用する場合、処理時間はそのまま利用料金に直結します。 効率の良いアルゴリズムを書くことは、お財布にも地球環境にも優しいのです。

空間計算量（メモリの節約）

ここまで時間の速さ（時間計算量）の話をしてきましたが、メモリをどれだけ使うか（空間計算量）も大切です。 いくら速くても、実行するのに1TB（テラバイト）のメモリが必要なプログラムは、普通のパソコンでは動きません。

- **時間計算量**: どれくらい待たされるか。
- **空間計算量**: どれくらい作業机（メモリ）を広げるか。

良いアルゴリズムとは、この時間と空間のバランスが取れたものを指します。 基本的には「時間は速く、メモリ消費は少なく」を目指しますが、時には「メモリをたくさん使う代わりに、爆速で処理する（キャッシュなど）」というトレードオフ（交換条件）を選ぶこともあります。

演習（プログラミング問題）

ここまで学んだアルゴリズムの概念を、実際にコードを動かして体験してみます。 今回は、自分でゼロからコードを書くだけでなく、AIにコードを書かせてその動きを観察するという現代的な学習スタイルで進めていきます。

アルゴリズムのトレース

トレース（Trace）とは、プログラムが実行される過程を1行ずつ追いかけて、変数の変化などを確認する作業のことです。アルゴリズムを理解するには、完成したコードを見るだけでなく、データがどのように変化していくかを目撃するのが一番の近道です。

課題1: 選択ソートの動きを可視化せよ

先ほどバブルソートを紹介しましたが、ここでは別の整列アルゴリズムである**選択ソート（Selection Sort）**を題材にします。 選択ソートは、残っているデータの中から最小値を見つけて先頭に持ってくる、という作業を繰り返すアルゴリズムです。

ステップ1: AIにコードを生成させる

お使いのAI（ChatGPTやClaudeなど）に、以下のプロンプト（指示）を入力して、Pythonコードを生成させてみましょう。

> ***プロンプト例*** *Pythonで選択ソートのアルゴリズムを書いてください。ただし、学習用に使いたいので、数字が入れ替わるたびにその時のリストの中身をprintで出力して、どのように並び替えが進んでいるか目で見えるようにしてください。*

ステップ2: コードを実行して観察する

AIが生成したコードを実行してみると、以下のように「最小値を探して、先頭と入れ替える」様子が順番に表示されます。

```
開始状態: [29, 10, 14, 37, 13]
1. 最小値 10 を発見、29 と交換: [10, 29, 14, 37, 13]
2. 最小値 13 を発見、29 と交換: [10, 13, 14, 37, 29]
3. 最小値 14 を発見、14 と交換: [10, 13, 14, 37, 29]
...
最終結果: [10, 13, 14, 29, 37]
```

実行結果の読み解き方

1. 最初は `[29, 10, 14, 37, 13]` です。
2. この中で一番小さいのは `10` です。先頭の `29` と `10` を交換します。 → `[10, 29, 14, 37, 13]` （10が確定）

3. 次は、残りの `[29, 14, 37, 13]` の中で一番小さい `13` を探します。これを `29` と交換します。 → `[10, 13, 14, 37, 29]` （10, 13が確定）
4. これを最後まで繰り返します。

途中経過を出力させる（プリントデバッグといいます）ことで、ブラックボックスだったアルゴリズムの中身が手に取るように分かるようになります。

ステップ3: AIに解説させる

もしコードの中で分からない部分があれば、その行をAIに貼り付けて「この行は何をしているの？」と聞いてみましょう。 AIは優秀なメンターとして、複雑な処理も分かりやすく解説してくれます。

FizzBuzz問題の応用

FizzBuzz（フィズ・バズ）は、プログラミングの学習において世界で最も有名な練習問題の一つです。条件分岐（if文）のロジックを考えるのに最適です。

課題2: 3と5と7の倍数

まずは基本のFizzBuzzをおさらいします。

- 3の倍数のときは "Fizz"
- 5の倍数のときは "Buzz"
- 両方の倍数（15など）のときは "FizzBuzz"
- それ以外は数字そのまま

これを少し複雑にして、7の倍数のときは "Pop" というルールを追加してみます。 つまり、

- 3の倍数 → Fizz
- 5の倍数 → Buzz
- 7の倍数 → Pop
- 3と5 → FizzBuzz
- 3と7 → FizzPop
- 5と7 → BuzzPop
- 3と5と7 → FizzBuzzPop

このロジックをどう組み立てるかが、アルゴリズムの腕の見せ所です。

アプローチA: 正直にif文を重ねる（大変な例）

全ての組み合わせを `if` と `elif` で書こうとすると、条件の順番や抜け漏れに非常に気を使います。

```
# 悪い例ではありませんが、拡張するのが大変な書き方
if n % 3 == 0 and n % 5 == 0 and n % 7 == 0:
    print("FizzBuzzPop")
elif n % 3 == 0 and n % 5 == 0:
    print("FizzBuzz")
# ... これを延々と書く必要がある
```

アプローチB: 文字列連結を使う（賢い例）

ここで、アルゴリズムの工夫をします。 「3の倍数ならFizzを追加」「5の倍数ならBuzzを追加」という風に、出力する文字を積み上げていくと考えれば、条件分岐をスッキリさせることができます。

```
# アルゴリズムの工夫：文字列を積み上げる
output = ""
if i % 3 == 0: output += "Fizz"
if i % 5 == 0: output += "Buzz"
if i % 7 == 0: output += "Pop"
```

```
if output == "": output = str(i)
print(output)
```

このアルゴリズムの優れた点は、新しいルール（例：11の倍数はBang）が増えても、`if` 文を一つ追加するだけで対応できる「拡張性の高さ」にあります。

どういう手順で処理するかを変えるだけで、プログラムの分かりやすさや変更のしやすさが大きく改善することを体感してください。

問題1 線形探索のトレース

問題文

N個の整数からなる配列と、探したい値Tが与えられます。 配列の中からTを線形探索（先頭から順番に探す方法）で探してください。

探索の**各ステップ**で、現在確認している要素とTの比較結果を出力してください。 一致した場合は `値 == T` の形式で出力し、そこで探索を終了してください。 一致しなかった場合は `値 != T` の形式で出力し、次の要素に進んでください。

最後の行には、見つかった場合はそのインデックス（0始まり）を、見つからなかった場合は `-1` を出力してください。

同じ値が複数ある場合は、最初に見つかったもののインデックスを出力してください。

入力

```
N T
a_1 a_2 ... a_N
```

- 1行目にNとTが空白区切りで与えられます
- 2行目にN個の整数が空白区切りで与えられます

制約

- $1 \leq N \leq 100$
- $-1000 \leq a_i \leq 1000$
- $-1000 \leq T \leq 1000$

出力

各ステップの比較結果を1行ずつ出力し、最後の行に結果（インデックスまたは-1）を出力してください。

入力例1

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

出力例1

```
1 != 3
4 != 3
3 == 3
2
```

先頭から順に確認し、インデックス2で値3が見つかりました。

入力例2

```
4 10
1 2 3 4
```

出力例2

```
1 != 10
2 != 10
3 != 10
4 != 10
-1
```

全ての要素を確認しましたが10は見つかりませんでした。

問題2 バブルソートのトレース

問題文

N個の整数からなる配列が与えられます。 この配列をバブルソート（隣り合う要素を比較して入れ替える方法）で昇順に整列してください。

まず**初期状態**の配列を出力し、その後バブルソートの**各パス（外側ループの1回分）が終わるごとに**、その時点での配列の状態を出力してください。

バブルソートのパスは$N-1$回行ってください（途中でソートが完了しても省略しないでください）。

入力

```
N
a_1 a_2 ... a_N
```

- 1行目にNが与えられます
- 2行目にN個の整数が空白区切りで与えられます

制約

- $1 \leq N \leq 100$
- $-1000 \leq a_i \leq 1000$

出力

初期状態を含め、合計N行を出力してください。各行は配列の要素を空白区切りで出力してください。

入力例1

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

出力例1

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

1行目は初期状態、2行目以降は各パス後の状態です。

入力例2

```
4
9 7 8 6
```

出力例2

```
9 7 8 6
7 8 6 9
7 6 8 9
6 7 8 9
```

問題3 最大値探索のトレース

問題文

N個の整数からなる配列が与えられます。 配列を先頭から順に走査して最大値を求めてください。

走査の**各ステップ**で、現在の暫定最大値を出力してください。 つまり、i番目（0始まり）の要素まで見た時点での最大値を、$i = 0, 1, \ldots, N-1$ の順に1行ずつ出力してください。

入力

```
N
a_1 a_2 ... a_N
```

- 1行目にNが与えられます
- 2行目にN個の整数が空白区切りで与えられます

制約

- $1 \leq N \leq 100$
- $-1000 \leq a_i \leq 1000$

出力

N行出力してください。i行目（1始まり）には、先頭からi個の要素の中での最大値を出力してください。

入力例1

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

出力例1

```
3
3
4
4
5
```

要素を順に見ていくと、最大値は 3→3→4→4→5 と変化します。

入力例2

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

出力例2

```
-2
-2
-1
-1
```

全て負の数でも、暫定最大値は -2→-2→-1→-1 と変化します。

第6章 データ構造：情報を「賢く」扱うためのコンピュータサイエンスの核心

1. はじめに：なぜデータ構造を学ぶのか？

これまでに皆さんは、変数、条件分岐、ループといったプログラミングの基礎を学んできました。これらはプログラムの「動き」を作るための基本的な道具です。

しかし、実際のソフトウェア開発において、コードを動かすことと同じくらい、あるいはそれ以上に重要なのが、**「データをどのように持たせるか」**という設計の視点です。今回は、プログラムをより高速に、そして効率的に動かすための鍵となる「データ構造」について深く学んでいきましょう。

1.1 データの整理整頓：本棚のメタファー

あなたの部屋に1,000冊の本があると想像してください。これらをどのように管理するのが最も効率的でしょうか？

1. **床に無造作に積み上げている状態（データ構造がない状態）**：あなたが「夏目漱石の『こころ』を読みたい」と思ったとき、どこから手を付ければよいでしょうか？ おそらく、一番上の本から順に一冊ずつ手に取って、タイトルを確認し、違えばどけるという作業を繰り返すしかありません。運が良ければ数秒で見つかるかもしれませんが、最悪の場合は1,000冊すべてを確認する必要があり、大変な時間と労力がかかります。この「端から順番に全部見ていく」という非効率な探し方は、コンピュータの世界では「線形探索（リニアサーチ）」と呼ばれ、データが大量にある場合には致命的な遅さの原因となります。
2. **図書室のように整理された本棚（適切なデータ構造がある状態）**：もし、本がジャンル別に棚分けされ、さらに著者名の五十音順に並んでいたらどうでしょうか。「日本文学」の棚に行き、「な」の場所を探すだけで、ものの数秒で目的の本を手に取ることができます。1,000冊の本を一枚ずつめくる必要はどこにもありません。この「あらかじめ整理されたルールに従って、瞬時に目的地にたどり着く」仕組みこそが、データ構造の真髄です。

コンピュータの世界でも全く同じことが言えます。 **データ構造（Data Structure）**とは、データをメモリ上にどのように配置し、どのように関連付けるかという「情報の整理ルール」のことです。適切なデータ構造を選ぶことで、プログラムの実行速度は劇的に向上し、同時にコードの読みやすさ（保守性）も高まります。データ構造を学ぶことは、コンピュータをより効率的に、そして賢く使いこなすための第一歩なのです。

1.2 適切な構造を選ぶ重要性と「トレードオフ」の理解

プログラミングにおいて「どんな場面でも最強のデータ構造」というものは存在しません。これを「銀の弾丸はない（No Silver Bullet）」と呼んだりします。それぞれのデータ構造には得意分野と苦手分野があり、一報を良くしようとすると他方が犠牲になる「トレードオフ」の関係にあります。

- **参照（中身を見るの）は速いが、途中への挿入は遅い構造**: 例えば「配列」はこれに当たります。番号を指定すれば一瞬でデータが見つかりますが、割り込みをしようとすると他のデータをどかす手間がかかります。
- **挿入や削除は一瞬だが、特定の場所のデータを探すのは遅い構造**: 「連結リスト」が代表例です。前後の繋がりを書き換えるだけでいいので追加は簡単ですが、中身を探すには先頭から辿る必要があります。
- **メモリを多く消費するが、特定の計算が極めて速い構造**: 「ハッシュテーブル」などは、検索速度を極限まで高めるために、わざとメモリに余裕を持たせてデータを配置します。

料理をするとき、スープを作るにはお玉を使い、パスタを茹で上げるときにはトングを使います。これと同じように、プログラマは実現したい機能や処理の内容に合わせて、手持ちの「データ構造という道具箱」の中から最適なものを選び抜くスキルが求められます。

---

2. コンピュータの記憶装置とメモリの仕組み

データ構造を深く理解するためには、コンピュータがどのようにデータを記憶しているか、その舞台裏を少しだけ知っておく必要があります。

2.1 メモリ (RAM) は「番号付きの巨大なロッカー」

コンピュータのメインメモリ（RAM）は、通し番号（アドレス）がついた巨大なロッカーのようなものが、一列に果てしなく並んでいるイメージです。それぞれのロッカーには、1単位のデータを保存することができます。

- **アドレス**: それぞれのロッカーにつけられた「0番地」「1番地」という住所です。コンピュータはこの住所を頼りにデータを読み書きします。
- **データ**: ロッカーの中には、0か1の組み合わせ（ビット）で表現された情報が入ります。
- **連続したアクセス**: 隣り合うロッカーを順番に開けていく作業は非常に速いです。コンピュータはこの「並び」を扱うのが得意です。
- **ランダムアクセス**: 1番のロッカーを開けた次に、突然100万番のロッカーを開けるといった「飛び飛び」の作業は、ロッカーの場所を探す手間がかかるため、連続したアクセスに比べるとわずかに効率が落ちる場合があります。

2.2 CPUキャッシュと空間的局所性

実は、現代のCPUには「キャッシュメモリ」という、RAMよりもさらに高速なメモリが内蔵されています。CPUはメモリからデータを取ってくる際、そのデータだけでなく、**「その周辺にあるデータ」もまとめてキャッシュにコピー**します。 これを「空間的局所性」と呼びます。隣り合うデータが並んでいる「配列」が、「連結リスト」よりも実効速度で勝ることが多いのは、このハードウェアの仕組みのおかげです。データ構造は、この物理的な特性すら考慮して設計されています。

---

3. 基本構造：配列 (Array) と連結リスト (Linked List)

データを一列に並べて管理する方法には、大きく分けて「配列」と「連結リスト」の2つの流派があります。

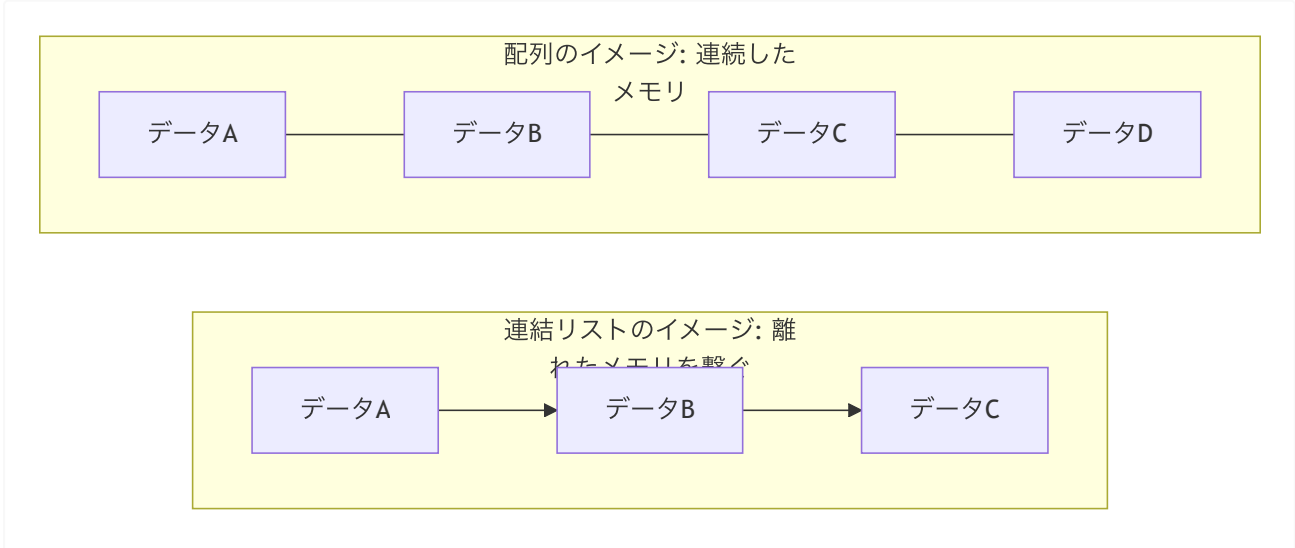
3.1 配列 (Array)：一列に並んだ指定席

配列は、メモリ上の「連続した場所」にデータを隙間なく、順番に並べる構造です。

- **特徴**: ロッカーが1番から10番まで、一続きのブロックとして確保されている状態です。
- **メリット (アクセスの速さ)**: 最大の長所は、データの場所を即座に特定できることです。「100番目のデータが見たい」と言われれば、コンピュータは「先頭の場所 + 100」という簡単な計算によって、その場所を一瞬で割り出せます。この「どこでも一瞬でアクセスできる」特性を **$O(1)$** と呼びます。
- **デメリット (編集の遅さ)**: 列の途中に新しいデータを割り込ませたり、削除したりするのが非常に大変です。例えば、100人並んでいる列の2番目に新しい人を割り込ませるには、3番目から100番目までの全員に「一歩ずつ後ろに下がってスペースを空けてもらう」必要があります。この移動作業は、データが多ければ多いほど時間がかかります。

3.2 連結リスト (Linked List)：宝探しゲームのメモ

連結リストは、データがメモリ上のバラバラな場所に配置されていても、「次のデータがどこにあるか」というメモ（矢印やポインタ）で繋がっている構造です。



- **特徴**: データ本体と、「次のデータの場所はここですよ」という情報がセットになって保存されています。
- **メリット (編集の速さ)**: 列の途中にデータを追加したり、削除したりするのが非常に速いです。新しいデータを空いているロッカーに入れ、前のデータの「次へのメモ」を書き換えるだけで済みます。他の大量のデータを物理的に移動させる必要はありません。
- **デメリット (アクセスの遅さ)**: 特定の場所にあるデータを見るのが遅いです。1番目のデータから順に「次の人はどこ？」「その次は？」とメモを辿っていくしかないからです。目的地にショートカットすることは不可能なのです。

---

4. アルゴリズムと計算量：効率を測るための共通言語

プログラマがデータ構造を選ぶとき、感覚ではなく「どれくらい効率的か」を数学的な基準で判断します。そのための共通言語が**計算量（Complexity）**です。

4.1 ビッグオー記法 (Big O Notation) の種類

- **$O(1)$（定数時間）**: データの数が10個でも100億個でも、処理にかかる時間が一定です。配列の要素アクセスやハッシュテーブルの検索がこれに当たります。まさに理想的な速さです。
- **$O(\log n)$（対数時間）**: データが2倍、4倍と増えても、手間はわずか（1回、2回...）しか増えません。非常に効率的なアルゴリズムです。二分探索がこの代表例です。100万件のデータも、わずか20回程度の処理で見つけられます。
- **$O(n)$（線形時間）**: データが10倍になれば、かかる時間も10倍になります。予測しやすいですが、データが膨大になると目に見えて時間がかかるようになります。線形探索がこれに当たります。
- **$O(n \log n)$**: $O(n)$ よりは少し遅いですが、実用的なソートアルゴリズム（クイックソートやマージソート）の多くはこの範囲に収まります。
- **$O(n^2)$（二乗時間）**: データが10倍になると、手間は100倍になります。二重ループを使った単純なアルゴリズム（バブルソートなど）で見られます。データが数万件を超えると実用性を失います。

4.2 時間計算量と空間計算量

計算量には、実行時間に関わる「時間計算量」と、どれだけメモリを使うかに関わる「空間計算量」の2種類があります。時には、メモリを贅沢に使うことで計算時間を短縮する、という高度な設計判断が行われます。

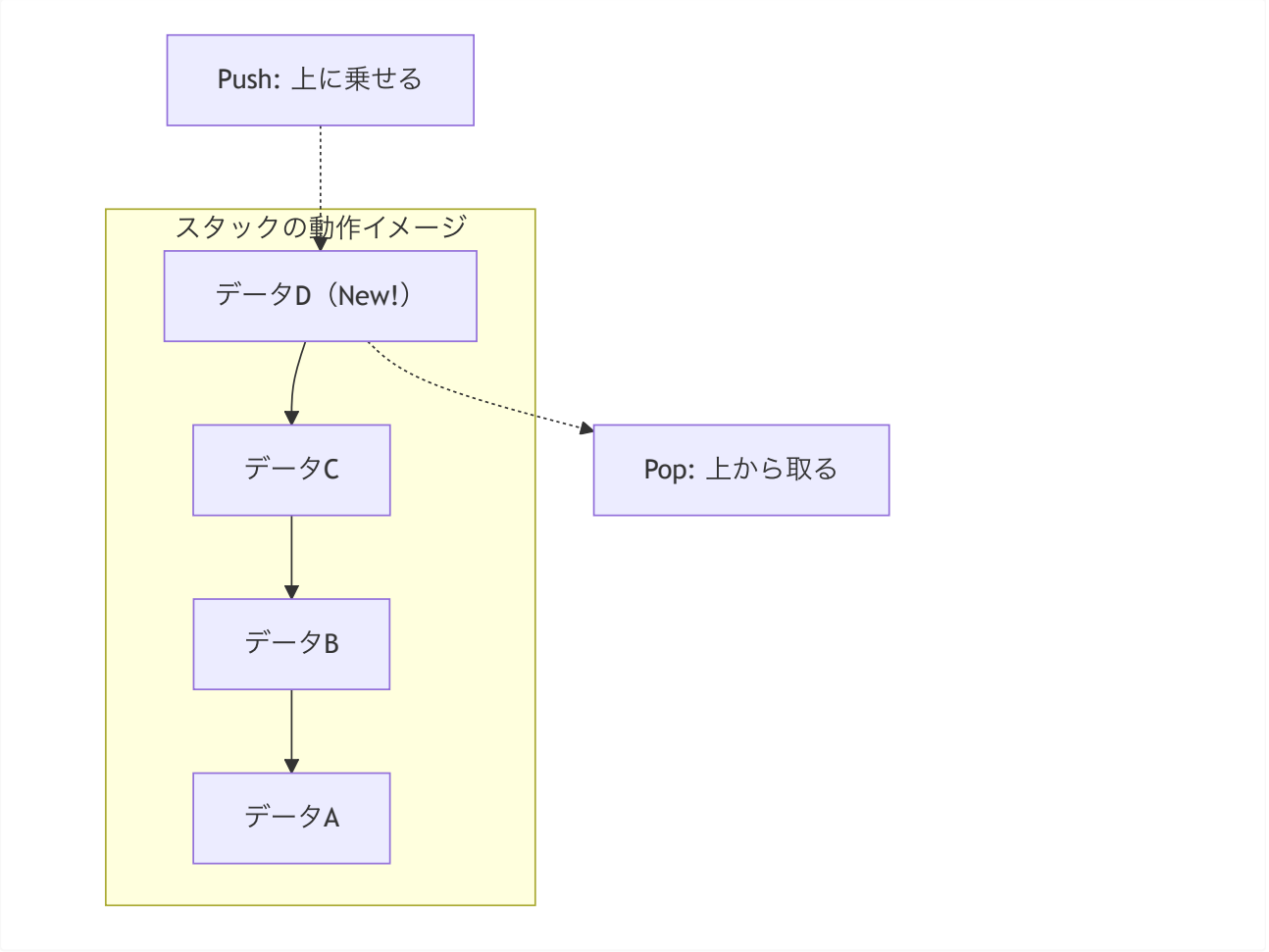
---

5. 線形データ構造の応用：スタック (Stack) とキュー (Queue)

データを単に並べるだけでなく、出し入れの順番にルールを設けることで、複雑な機能をシンプルに実現できます。

5.1 スタック (Stack)：後入れ先出し (LIFO)

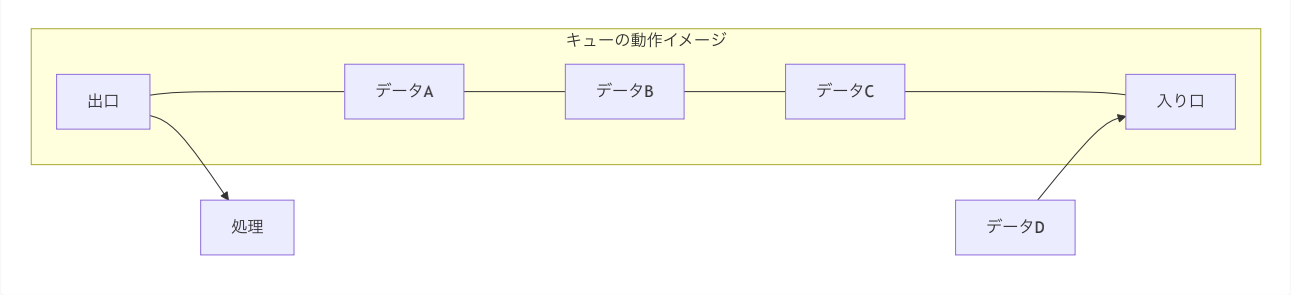
- **ルール**: **LIFO (Last In, First Out)**。
- **イメージ**: 机の上に積み上げた本、洗った後のお皿の山。
- **操作**: `Push`（一番上に積む）、`Pop`（一番上を取り出す）。



- **活用シーン**:
  - **Webブラウザの「戻る」ボタン**: 最後に見たページが一番上にあり、それを最初に取り出します。
  - **エディタのUndo（元に戻す）**: 直前の操作がスタックの頂上に保存されており、それを最初に取り消します。
  - **関数の呼び出し**: プログラムがある関数を呼び出す際、現在地をスタックに保存して、終わったら戻ります。

5.2 キュー (Queue)：先入れ先出し (FIFO)

- **ルール**: **FIFO (First In, First Out)**。
- **イメージ**: レジの行列、チケット購入の待ち列。
- **操作**: `Enqueue`（列の後ろに並ぶ）、`Dequeue`（列の先頭から出る）。



- **活用シーン**:
  - **プリンタの印刷待ち**: 送られた順序を守って一枚ずつ印刷します。
  - **イベント処理**: マウスクリックやキーボード入力などの信号を届いた順に処理します。
  - **タスクスケジューリング**: CPUが複数のプログラムを交互に動かす際の順番待ち。

5.3 両端キュー (Deque / Double-Ended Queue)

スタックとキューの両方の性質を持ち、前からも後ろからも出し入れができる構造です。Pythonでは `collections.deque` として非常に高速な実装が提供されています。

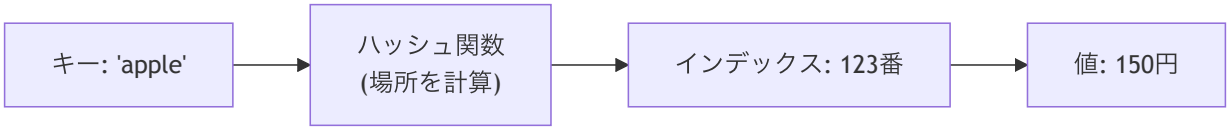
---

6. ハッシュテーブル（連想配列）：一瞬でデータを見つける魔法

「1番目のデータ」という番号ではなく、「apple」という「名前（キー）」でデータを管理したいときに登場します。

6.1 なぜハッシュテーブルは $O(1)$ なのか？

キー（例：'apple'）を**ハッシュ関数**という魔法の計算式に通すと、瞬時に「メモリの123番に保存せよ」という数値（ハッシュ値）が返ってきます。探すときも同じ計算をするだけで場所がわかるため、データが100万件あっても計算一回分、すなわち一瞬で見つけられるのです。



6.2 ハッシュ衝突と解決策

稀に、違うキーなのに同じハッシュ値（場所）が計算されてしまうことがあります。これを「ハッシュ衝突」と呼びます。

- **チェイン法**: 同じ場所に来たデータを「連結リスト」で繋いでいく方法。
- **オープンアドレス法**: もし場所が埋まっていたら、隣の空いている場所を探して入れる方法。

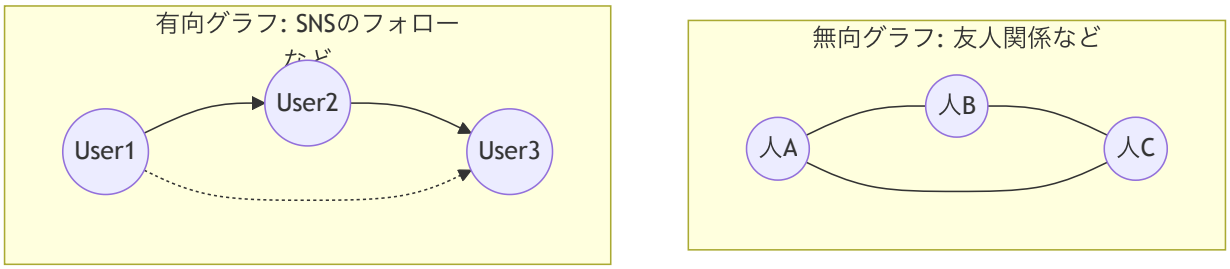
現代のプログラミング言語（Pythonの辞書 `dict` など）は、これらを内部で極めて高度に解決しており、エンジニアは衝突を意識することなく、高速な検索の恩恵を受けられます。

---

7. グラフ (Graph)：複雑なつながりをモデル化する

世の中の複雑な「つながり」を表現するための、もっとも汎用的な構造です。

- **ノード (Node)**: 頂点（点）。人、駅、Webページ、ルーターなど。
- **エッジ (Edge)**: 辺（線）。友達関係、路線、リンク、ケーブルなど。



7.1 グラフの種類

1. **無向グラフ**: 矢印がなく、つながりが双方向なもの。（例：相互の友人関係）
2. **有向グラフ**: 矢印があり、つながりに向きがあるもの。（例：SNSの「フォロー」、一方通行の道路）
3. **重み付きグラフ**: エッジに数値（距離、時間、コスト）がついているもの。（例：路線図の所要時間）

7.2 グラフ探索の代表的な手法

- **幅優先探索 (BFS)**: 今いる場所から「近い順」に探索範囲を広げていく方法。最短経路（一番近い駅など）を探すのに適しています。

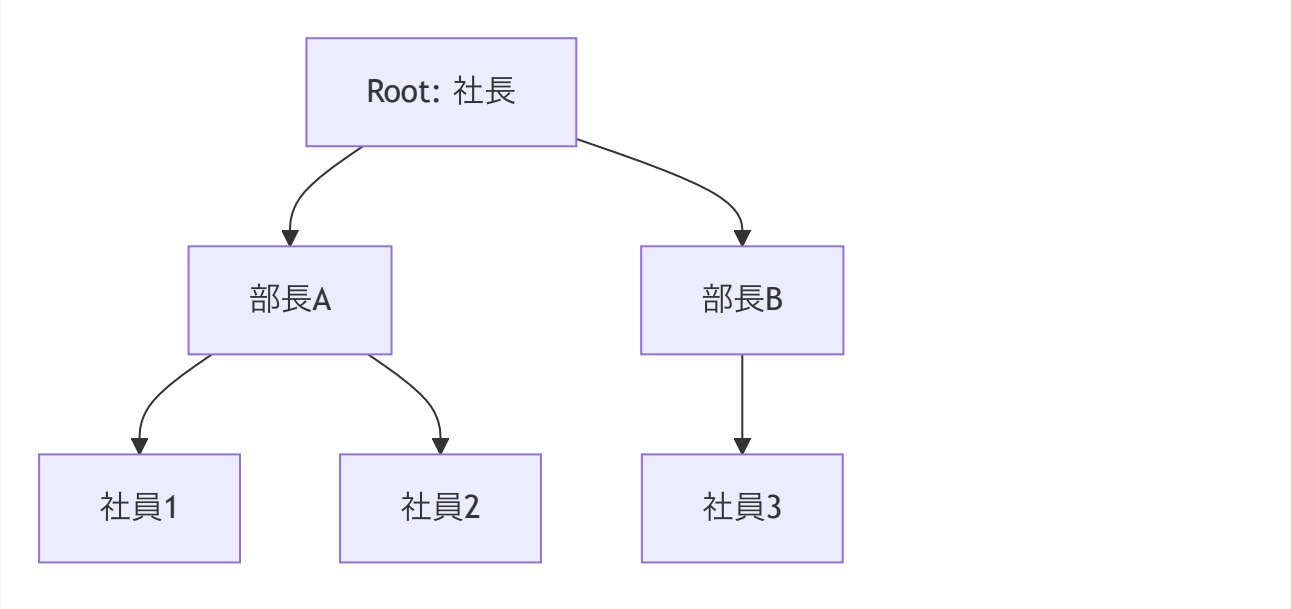
- **深さ優先探索 (DFS)**: 行けるところまで奥深く進み、行き止まったら戻る方法。迷路の脱出経路探しや、すべてのパターンを網羅する際に使われます。

---

8. 木 (Tree)：整理整頓された階層構造のマスター

親子関係を持つ、ループのない階層的なグラフです。自然界の木を逆さまにしたような形をしています。

- **ルート (Root)**: 唯一の親（根）。すべての出発点です。
- **リーフ (Leaf)**: 子を持たない末端（葉）。



- **活用例**:
  - **ファイルシステム**: Windowsの `C:\` や Macの `/` がルートです。
  - **HTMLの構造 (DOM)**: ブラウザはHTMLを解析して、木構造（DOMツリー）としてメモリに展開します。
  - **意思決定ツリー**: 質問にYes/Noで答えていき、結論に達する仕組み。

8.1 二分探索木 (Binary Search Tree)

「左の子は親より小さく、右の子は親より大きい」というルールを徹底した木です。このルールのおかげで、10億件のデータからでも、わずか30回程度の比較で目的のデータを見つけ出せます（計算量は $O(\log n)$）。

---

9. 探索とソートのアルゴリズム：データをどう操るか

データ構造が決まったら、次はそのデータをどう動かすかという手順（アルゴリズム）が必要です。

9.1 線形探索 vs 二分探索

- **線形探索**: 1ページ目から順に探す方法。データが並んでいなくても使えますが、遅いです。
- **二分探索**: ソート済みのデータを半分ずつ切り捨てる方法。辞書を引くような速さですが、**「あらかじめ並べ替えておく」**必要があります。

9.2 主要なソートアルゴリズム

1. **バブルソート ($O(n^2)$)**: 隣同士を比較して入れ替える。理解は簡単ですが、遅いです。
2. **クイックソート ($O(n \log n)$)**: 基準を決めて、大きいグループと小さいグループに分けることを繰り返す。非常に高速です。
3. **マージソート ($O(n \log n)$)**: データを最小単位までバラバラにして、順番を整えながら合体（マージ）させる。安定した速さが特徴です。

---

10. 【中級編】高度なデータ構造の世界

実務で使われる、さらに洗練されたデータ構造をいくつか紹介します。

10.1 優先度付きキューとヒープ (Heap)

通常のキューは「先入れ先出し」ですが、現実はそうとは限りません。病院の救急外来のように、「後に来た人でも重症なら先に診る」という仕組みが必要です。これを実現するのが**優先度付きキュー**です。 これを高速に実現するためのデータ構造が**「ヒープ」**で、常に「一番優先度の高いデータ」を木の頂上に持ってくるように自動で整列します。

10.2 トライ木 (Trie)：文字列検索の達人

文字列の共通する「接頭辞」を共有して保存する木です。 スマホのキーボードで文字を打っているときに表示される「予測変換」の裏側では、このトライ木が動いており、入力された数文字から始まる単語を一瞬でリストアップしています。

10.3 ブルームフィルタ (Bloom Filter)

「ある要素が集合に含まれているか」を、100%の正確さを捨てる代わりに、圧倒的な省メモリで判定する確率的なデータ構造です。 「含まれていない」と判定されたら100%含まれていませんが、「含まれている」と判定された場合は稀に間違いがあります。Webブラウザの有害サイト判定などに使われ、まずはブルームフィルタで高速にフィルタリングする、という使い方をされます。

---

11. 【実践】Pythonによるデータ構造の実装とシミュレーション

データ構造の理論を学んだ後は、実際にコードを書いてその動きを体験しましょう。

11.1 スタックによるカッコの対応チェック

プログラミング言語の構文チェックなどで実際に使われている手法です。

```
# python
# スタックを使ってカッコが正しく閉じられているかチェックする

def is_balanced(text):
    stack = []
    # 開きカッコと閉じカッコの対応表
    mapping = {")": "(", "}": "{", "]": "["}

    for char in text:
        if char in "({[":
            # 開きカッコならスタックに積む（Push）
            stack.append(char)
        elif char in ")}]":
            # 閉じカッコなら、スタックの一番上を取り出す（Pop）
            if not stack:
                return False # スタックが空なら対応する開きカッコがない

            top = stack.pop()
            if top != mapping[char]:
                return False # 対応が一致しない

    # 最後にスタックが空なら、すべて正しく閉じられている
    return len(stack) == 0

# テスト
```

```
print(is_balanced("( [ { } ] )")) # True
print(is_balanced("( [ ) ]"))     # False
```

11.2 再帰を使った木構造（ディレクトリ）の全探索

「フォルダの中にフォルダがある」という同じ構造が繰り返されるデータを扱うには、自分自身を呼び出す「再帰」がもっともスマートです。

```
# python
# ディレクトリ構造を再帰的に探索して、ファイルパスを表示する

def show_tree(node, current_path="Root"):
    # nodeが辞書なら「フォルダ」
    if isinstance(node, dict):
        for name, content in node.items():
            new_path = f"{current_path}/{name}"
            print(f"📂 フォルダ: {new_path}")
            # さらに深く（再帰呼び出し）
            show_tree(content, new_path)
    # nodeがリストなら「ファイルリスト」
    elif isinstance(node, list):
        for filename in node:
            print(f"  📄 ファイル: {current_path}/{filename}")

# サンプルデータ
file_system = {
    "Documents": {
        "Work": ["resume.pdf", "data.xlsx"],
        "Private": ["diary.txt"]
    },
    "Pictures": ["cat.jpg", "dog.png"]
}

show_tree(file_system)
```

---

12. 【応用】AI、Web開発、ゲームとデータ構造

データ構造は、皆さんが普段使っている最先端の技術を支えています。

12.1 AIと「テンソル (Tensor)」

現代のAI（ディープラーニング）の正体は、膨大な数の数値計算です。 これらの数値は「テンソル」と呼ばれる多次元配列（多次元のロッカー）に格納されます。画像データは (縦, 横, 色) という3次元の配列として扱われ、これをGPU（画像処理装置）で超高速に並列計算することで、AIは画像の内容を認識します。

12.2 Web開発と「仮想DOM」

Reactなどの最新のWeb開発ツールでは、ブラウザの木構造（DOM）を操作する前に、メモリ上に「仮想の木（仮想DOM）」を作ります。2つの木を比較して「どこが新しくなったか」という差分だけを計算し、最小限の更新を行うことで、ヌルヌルと動く快適なWebページを実現しています。

12.3 ゲーム開発と「空間分割」

オープンワールドのゲームでは、広大なマップに数万個のオブジェクトが存在します。すべての敵キャラと自分の距離を計算すると、ゲームはフリーズしてしまいます。 そこで「四分木」や「八分木」といった木構造を使って、自分の周りのエリアだけに計算

範囲を絞り込むことで、快適なゲームプレイを支えています。

---

13. データ構造の歴史：メモリとの戦いの記録

データ構造の歴史は、いかに限られたメモリを節約するか、という挑戦の歴史でもあります。

- **1950年代（黎明期）**：コンピュータのメモリは極めて貴重で、数KBしかありませんでした。配列という単純な構造が主流でした。
- **1960年代（LISPの誕生）**：ジョン・マッカーシーによってLISP言語が発明され、「連結リスト」が世に広まりました。これにより、プログラムの実行中に動的にサイズが変わるデータを扱えるようになりました。
- **1970年代（理論の確立）**：ドナルド・クヌースらによってアルゴリズム解析（Big O）の理論が体系化されました。B木などの高度な木構造が登場し、今日のデータベースの基礎となりました。
- **現在（ビッグデータ時代）**：インターネット上の膨大なデータを分散して保持する「分散ハッシュテーブル」や、SNSの巨大なつながりを解析する「グラフデータベース」など、データ構造は今も進化し続けています。

---

14. 詳細用語解説 (Ultimate Glossary)

理解を深めるための完全版用語集です。

- **ADT (Abstract Data Type / 抽象データ型)**: 「何ができるか」という機能の定義のことです。例えばスタックなら「PushとPopができるもの」という定義を指します。これを配列で作るか、連結リストで作るかは実装者の自由であり、この「定義と実装の分離」がソフトウェア開発では非常に重要です。
- **再帰 (Recursion)**: 関数が自分の処理の中で自分自身を呼び出すことです。大きな問題を、それより一回り小さな同じ形の問題に分割して解くときに使われます。
- **ハッシュ値 (Hash Value)**: ハッシュ関数によって生成される、データの「指紋」のような数値です。データが少しでも変わればハッシュ値も大きく変わります。
- **ポインタ (Pointer)**: メモリ上の特定の場所（住所）を指し示す「指」のような情報です。C言語などの低レイヤーな言語では直接操作しますが、Pythonなどでは裏側で自動的に管理されています。
- **トラバース (Traverse)**: データ構造内のすべての要素を、漏れなく重複なく順番に巡回することです。
- **動的配列 (Dynamic Array)**: 実行中に自動でサイズが変わる配列です。Pythonのリストがこれに当たります。内部では、サイズが足りなくなると、さらに大きな新しい配列を確保して全データを引っ越しさせるという大掛かりなことを裏側で行っています。
- **平衡木 (Balanced Tree)**: 左右の枝の高さが揃うように自動でバランスを調整する賢い木構造です。データを入れる順番が悪いと、木はただの一列（連結リスト）のようになってしまい検索速度が落ちるため、それを防ぐ工夫です。
- **隣接リスト (Adjacency List)**: グラフの各ノードが、自分と繋がっている相手のリストを個別に持つ形式です。SNSの友人リストのような管理方法です。
- **空間計算量 (Space Complexity)**: プログラムを実行するのにどれくらいのメモリが必要かを表す尺度です。
- **時間計算量 (Time Complexity)**: プログラムの実行にどれくらいの手間がかかるかを表す尺度です。
- **安定ソート (Stable Sort)**: 同じ値を持つデータの元々の順番を崩さない優しいソート方法です。例えば「年齢」でソートした後に「名前」でソートしても、年齢の順番が維持されるような性質です。
- **優先度付きキュー**: 単なる順番待ちではなく、緊急度の高い要素から先に取り出されるキューです。
- **インデックス (Index)**: 配列などで、何番目の要素かを指定するための番号（通常0から始まります）。
- **イテレータ (Iterator)**: データ構造の要素を一つずつ順番に取り出すための仕組みです。
- **多次元配列**: 配列の中に配列が入っている構造です。2次元配列はExcelのシートのような「表」を表せます。

---

15. まとめ：エンジニアとしての設計思想

データ構造を学ぶことは、コンピュータという魔法の箱を操るための「知恵」を学ぶことです。

どのようなデータ構造を選ぶかという決定は、その後の開発のしやすさ、システムのパフォーマンス、そしてユーザー体験（UX）のすべてに直結します。 これからは、新しい技術やライブラリに触れるとき、「その裏側ではどんなデータ構造が動いているのだろう？」と想像してみてください。 その問いこそが、あなたを優れたエンジニアへと導く羅針盤となるでしょう。

---

16. 演習問題：学んだ知識を定着させよう

演習1：最適なデータ構造を提案しよう

以下のシステムを作る際、どのデータ構造が最も適しているか理由とともに考えてください。

1. **スマートフォンの通知履歴**: 新しい順に並び、古いものは下に追いやられる。
2. **人気ラーメン店の予約システム**: 受付をした順番に客を案内する。
3. **英和辞典アプリ**: 英単語を入力したら一瞬で日本語の意味を表示する。
4. **家系図アプリ**: 親、子、孫の関係を可視化・管理する。

演習2：アルゴリズムの思考実験

100個のデータが順番に並んで（ソートされて）います。ここから特定の1つを探すとき、

- 「線形探索」を使った場合、最悪で何回確認が必要ですか？
- 「二分探索」を使った場合、最大で約何回の確認で済みますか？（ヒント：2の何乗が100を超えますか？）

演習3：AIを使ってさらに深く学ぼう

生成AI（ChatGPT, Claude, Gemini など）に以下のプロンプトを投げて、さらに高度なトピックに触れてみましょう。

プロンプト

```
IT初心者です。データ構造について勉強しています。『ヒープ（Heap）』というデータ構造について、日常生活の例を使ってわかりやすく教えてください。また、それが『優先度付きキュー』という仕組みの中でどう役に立っているかも詳しく説明してください。
```

問題1 スタックの操作シミュレーション

問題文

スタック（後入れ先出し: LIFO）のデータ構造をシミュレーションします。

N 個の操作が順番に与えられます。操作は以下の2種類です。

- `PUSH x` : 整数 x をスタックの一番上に積む
- `POP` : スタックの一番上の要素を取り除く（スタックが空の場合は何もしない）

すべての操作を実行した後、スタックの一番上にある要素を出力してください。スタックが空の場合は `EMPTY` と出力してください。

入力データの特徴

- 1 ≤ N ≤ 100
- PUSH で与えられる整数 x は -1000 以上 1000 以下の整数

入力

入力は以下の形式で標準入力から与えられる。

```
N
op_1
op_2
...
op_N
```

各 op_i は `PUSH x` または `POP` のいずれかである。

出力

すべての操作を実行した後のスタックの一番上の要素を出力せよ。スタックが空の場合は `EMPTY` と出力せよ。

入力例1

```
3
PUSH 5
PUSH 3
POP
```

出力例1

```
5
```

PUSH 5 でスタックに 5 を積み、PUSH 3 で 3 を積みます。POP で一番上の 3 を取り除くと、スタックの一番上は 5 になります。

入力例2

```
2
PUSH 42
POP
```

出力例2

```
EMPTY
```

PUSH 42 でスタックに 42 を積み、POP で取り除くと、スタックは空になります。

問題2 キューの操作シミュレーション

問題文

キュー（先入れ先出し: FIFO）のデータ構造をシミュレーションします。

N 個の操作が順番に与えられます。操作は以下の2種類です。

- `ENQUEUE x` : 文字列 x をキューの末尾に追加する
- `DEQUEUE` : キューの先頭の要素を取り出す

DEQUEUE 操作が行われるたびに、取り出された要素を出力してください。キューが空の状態で DEQUEUE が呼ばれた場合は `EMPTY` と出力してください。

入力データの特徴

- 1 ≤ N ≤ 100
- ENQUEUE で与えられる文字列 x は英小文字またはアルファベットからなる長さ 1 以上 20 以下の文字列

入力

入力は以下の形式で標準入力から与えられる。

```
N
op_1
op_2
...
op_N
```

各 op_i は `ENQUEUE x` または `DEQUEUE` のいずれかである。

出力

DEQUEUE 操作が行われるたびに、取り出された要素を1行に1つずつ出力せよ。キューが空の場合は `EMPTY` と出力せよ。

入力例1

```
5
ENQUEUE Alice
ENQUEUE Bob
DEQUEUE
ENQUEUE Charlie
DEQUEUE
```

出力例1

```
Alice
Bob
```

ENQUEUE で Alice、Bob の順にキューに追加します。最初の DEQUEUE で先頭の Alice を取り出し、次に Charlie を追加した後、DEQUEUE で先頭の Bob を取り出します。

入力例2

```
3
ENQUEUE X
DEQUEUE
DEQUEUE
```

出力例2

```
X
EMPTY
```

ENQUEUE で X を追加し、最初の DEQUEUE で X を取り出します。2回目の DEQUEUE ではキューが空なので EMPTY を出力します。

問題3 辞書検索

問題文

辞書（キーと値のペア）を使った検索をシミュレーションします。

まず N 個のキーと値のペアが与えられます。次に Q 個のクエリが与えられるので、各クエリのキーに対応する値を出力してください。キーが辞書に存在しない場合は `NOT_FOUND` と出力してください。

入力データの特徴

- 1 ≤ N ≤ 100
- 1 ≤ Q ≤ 100
- キーと値は英数字からなる長さ 1 以上 20 以下の文字列
- 同じキーが複数回登録されることはない

入力

入力は以下の形式で標準入力から与えられる。

```
N
key_1 value_1
key_2 value_2
...
key_N value_N
Q
query_1
query_2
...
query_Q
```

出力

各クエリについて、対応する値を1行に1つずつ出力せよ。キーが存在しない場合は `NOT_FOUND` と出力せよ。

入力例1

```
3
apple 150
banana 100
orange 200
2
apple
grape
```

出力例1

```
150
NOT_FOUND
```

apple はキーとして登録されているので値 150 を出力します。grape は登録されていないので NOT_FOUND を出力します。

入力例2

```
2
dog woof
cat meow
3
cat
dog
bird
```

出力例2

```
meow
woof
NOT_FOUND
```

cat の値は meow、dog の値は woof です。bird は登録されていないので NOT_FOUND を出力します。

第8章 データ分析

データの集計

これまでの講義で、プログラムを使ったデータの生成や通信によるデータ取得の方法を扱いました。ただし、データは持っているだけでは意味がありません。 そこに隠された意味を見つけ出し、次の行動につなげるプロセスがデータ分析です。

その第一歩として、バラバラのデータをまとめて全体像をつかむ集計（しゅうけい）について扱います。

合計・平均・件数

データの全体像を把握する基本統計量

手元に、あるお店の1週間の売上データや、クラス全員のテストの点数など、たくさんの数字があるとします。数字の羅列をただ眺めていても、全体の調子がどうなのかは分かりません。

そこで、データの特徴をひとつの数字で表す指標を使います。これを基本統計量（きほんとうけいりょう）と呼びます。特に重要なのが以下の3つです。

1. **件数（Count）**：データがいくつあるか。データの規模を表します。
2. **合計（Sum）**：全部の値を足したもの。総量を表します。
3. **平均（Mean / Average）**：合計を件数で割ったもの。だいたいこれくらいという基準を表します。

平均値の罠と外れ値

平均は便利な指標ですが、万能ではありません。**外れ値**（はずれち）と呼ばれる、極端に大きい（または小さい）値に引っ張られやすいという弱点があります。

例えば、5人の貯金額のデータがあるとします。

- Aさん: 10万円
- Bさん: 20万円
- Cさん: 30万円
- Dさん: 40万円
- **Eさん（大富豪）: 1000万円**

この5人の平均貯金額は **220万円** です。 しかし、Aさん〜Dさんの4人にとっては「自分は平均よりはるかに少ない」と映ります。実際には、Eさん以外の4人はごく標準的です。

このことから、**平均値が必ずしも「普通」を表すわけではない**と常に意識しておく必要があります。

Pythonによる基本統計量の計算

Pythonを使って実際に計算します。 あるRPGゲームのキャラクター5人の攻撃力データをリストとして用意し、その特徴を分析します。

```python
# python
# 基本的な統計量（件数、合計、平均）を計算するプログラム

def analyze_power(power_list):
    """
    攻撃力のリストを受け取り、分析結果を表示する関数
    """
    print(f"データ: {power_list}")

    # 1. 件数（要素の数）を求める: len()関数
    count = len(power_list)
    print(f"人数: {count} 人")
```

```
    # 2. 合計を求める: sum()関数
    # sum() はリストの中身を全部足してくれる便利な関数です
    total = sum(power_list)
    print(f"合計攻撃力: {total}")

    # 3. 平均を求める: 合計 ÷ 件数
    # 件数が0だと割り算エラーになるので、念のためチェックします
    if count > 0:
        average = total / count
        print(f"平均攻撃力: {average:.1f}") # 小数点第1位まで表示
    else:
        print("データがありません")

    print("-" * 20)

# --- 分析の実行 ---

# ケース1: バランスの良いパーティ
group_a = [100, 120, 90, 110, 130]
print("【グループAの分析】")
analyze_power(group_a)

# ケース2: 1人だけレベルMAXがいるパーティ（外れ値の影響）
# 999という極端な値が平均値をどう変えるか見てみましょう
group_b = [100, 120, 90, 110, 999]
print("【グループBの分析】")
analyze_power(group_b)
```

コードの解説です。

1. `len(リスト)`：リストに入っているデータの個数を返します。
2. `sum(リスト)`：リストに入っている数値をすべて合計して返します。自分で `for` ループを使って足し算を書く必要はありません。
3. `total / count`：平均の計算です。
4. 実行結果の比較:
    - グループAの平均は `110.0` になります。データ全体の実感と合っています。
    - グループBの平均は `283.8` になります。しかし、ほとんどのメンバーは100前後しかありません。「平均攻撃力283」という数字は、このチームの実力を正しく表しているとは言えません。これが外れ値の影響です。

グループごとの集計

分類して比較する

データ分析では、全体を見るだけでは不十分なことがよくあります。 例えば、「お店全体の売上が落ちている」という事実だけでは対策が立てられません。しかし、「食品売り場は好調だが、衣料品売り場が激減している」と分かれば、対策が見えてきます。

データを何らかの基準（カテゴリ）で分けて集計することを**グループ化（グルーピング）**と言います。

- カテゴリ別: 商品ジャンル別、店舗別、男女別など。
- 期間別: 月別、曜日別、時間帯別など。

比較することで初めて、「AはBよりも優れている」「Cは週末に伸びる傾向がある」といった特徴が浮き彫りになります。

Pythonによるグループ集計

複数の店舗を持つお店の売上データを分析します。 データは「どの店で（shop）」「いくら売れたか（sales）」という情報のリストとして与えられます。これを店舗ごとに合計して比較します。

```python
# python
# 店舗ごとの売上を集計するプログラム

# 売上データのリスト（辞書のリスト）
# 本来はデータベースやCSVファイルから読み込みますが、ここでは直接書きます
sales_data = [
    {"shop": "東京店", "sales": 100},
    {"shop": "大阪店", "sales": 200},
    {"shop": "東京店", "sales": 150}, # 東京店の2件目
    {"shop": "名古屋店", "sales": 120},
    {"shop": "大阪店", "sales": 80},  # 大阪店の2件目
    {"shop": "東京店", "sales": 300}, # 東京店の3件目
]

def aggregate_by_shop(data):
    print("--- 店舗別売上の集計開始 ---")

    # 集計結果を保存するための空の辞書を用意
    # キーを「店舗名」、値を「合計売上」にします
    # 例: {"東京店": 550, "大阪店": 280, ...}
    shop_totals = {}

    for record in data:
        shop_name = record["shop"]
        amount = record["sales"]

        # その店舗が辞書にまだ登録されていなければ、0で初期化
        if shop_name not in shop_totals:
            shop_totals[shop_name] = 0

        # 売上を加算する
        shop_totals[shop_name] += amount

    # 結果を表示
    # items() を使うと、辞書のキーと値をセットで取り出せます
    for name, total in shop_totals.items():
        print(f"店舗: {name} | 合計売上: {total}万円")

    return shop_totals

# --- 分析の実行 ---
result = aggregate_by_shop(sales_data)

# 最も売上が高い店舗を見つける応用
# max関数に辞書を渡すと、キー（店舗名）の最大値などを探せます
# key=result.get と指定すると、「値（売上）」の大きさで比較してくれます
best_shop = max(result, key=result.get)
print(f"\n🏆 最も売上が高かったのは: {best_shop} ({result[best_shop]}万円)")
```

コードの解説です。

1. データの構造: `[{"shop": "...", "sales": ...}, ...]` という形式は、JSONデータやデータベースの検索結果としてよく使われる形式です。
2. 集計ロジック:

- `shop_totals` という辞書（ハッシュテーブル）を集計表として使います。
- データを1件ずつ取り出し、その店舗の枠がまだなければ作り、枠があれば金額を足す、という処理を繰り返します。
- これはプログラミングにおける基本的な集計アルゴリズムです。

3. 結果の活用: 集計さえできてしまえば、あとは最大値を探したりグラフにしたりと、自由に分析を進められます。

集計結果の読み取り

計算が終わって数字が出ても、それで終わりではありません。むしろ、そこからが人間の出番です。

数字の意味を解釈する

出てきた数字を、ビジネスや生活の文脈（コンテキスト）に当てはめて**解釈**（かいしゃく）する力が求められます。

例えば、あるWebサイトのアクセス解析で、先月より訪問者数が2倍になった（合計が増えた）という結果が出たとします。これは単純に喜んでよいのか、立ち止まって考える必要があります。

- ポジティブな解釈: 広告の効果が出て、ファンが増えたのかもしれない。
- ネガティブな解釈: サイトに不具合があり、ユーザーが何度も再読み込みを連打してしまったのかもしれない。あるいは、炎上して野次馬が集まっただけかもしれない。

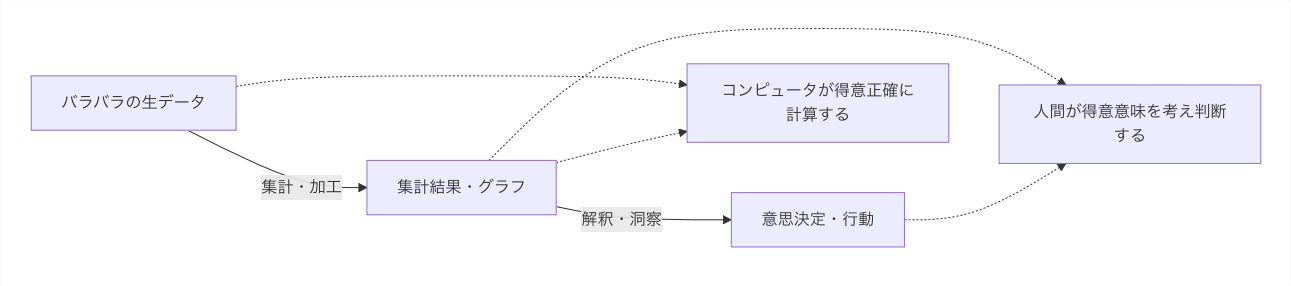
「なぜその数字になったのか」という背景を想像し、仮説を立てることが重要です。

疑う視点を持つ

データ分析では、数字を鵜呑みにせず、健全な疑いを持つことが大切です。

- データの偏り（バイアス）はないか
  - 「ユーザー満足度90%」というアンケート結果が、実はそのサービスを大好きなファンだけに聞いた結果だとしたら、全体を代表していません。
- 比較対象は適切か
  - アイスクリームの売上が増えたとき、カイロの売上と比較しても意味がありません。季節や条件を揃えて比較する必要があります。

集計はあくまで事実の要約に過ぎません。そこから真実を見抜くのは、AIではなく、分析者自身の洞察力です。



データの前処理

データを集めて集計すれば分析完了、と思って実際にやってみると、多くの場合うまく計算できなかったり、おかしな結果が出たりして失敗します。**現実世界のデータは汚い**からです。

料理に例えると分かりやすいです。畑で採れた野菜は、泥がついていたり虫食いがあったりします。そのまま鍋に入れたら料理が台無しです。洗って、皮を剥いて、傷んだ部分を取り除く下ごしらえが必要です。

データ分析でも同じことが言えます。集めたばかりの生データ（Raw Data）を、分析できる状態に綺麗に整える作業を**データの前処理**（ぜんしょり）と呼びます。 データサイエンティストは、実は分析そのものよりも、この前処理に多くの時間（全体の8割とも言われます）を使っています。

データのクリーニング（浄化）

コンピュータは融通がきかない

人間なら、「iPhone」と「iphone」と「ｉＰｈｏｎｅ」を見れば、全部同じものだと分かります。しかし、コンピュータにとってこれらは完全に別の文字列です。もしこれらが混ざったまま集計すると、本来は合計されるべきデータが別々に集計されてしまいます。これを**表記揺れ**（ひょうきゆれ）と言います。

データを正しく扱うために、表記を統一しゴミを取り除く作業がデータクリーニングです。

- 全角・半角の統一: アルファベットや数字は半角に揃えるのが一般的です。
- 大文字・小文字の統一: すべて小文字（または大文字）に変換して揃えます。
- 空白の削除: ユーザーが入力する際、末尾に見えないスペース（空白）が入ってしまうことがよくあります。「`Apple `」と「`Apple`」は別物として扱われるため、この余分な空白を削除（トリミング）する必要があります。

Pythonによるデータクリーニング

ユーザーから入力されたアンケートの回答データを想定して、汚いデータを綺麗にするプログラムを書きます。

```
# python
# 文字列の表記揺れを直すデータクリーニングのプログラム

def clean_text(text):
    """
    汚い文字列を受け取り、クリーニングして返す関数
    """
    if text is None:
        return None

    # 1. 前後の空白を削除する (stripメソッド)
    # 例: "  Apple  " -> "Apple"
    cleaned = text.strip()

    # 2. 小文字に統一する (lowerメソッド)
    # 例: "Apple" -> "apple"
    cleaned = cleaned.lower()

    # 3. 全角英数字を半角に直すなどの高度な処理も可能ですが、
    # ここでは基本的な「文字の置き換え」で対応する例を示します
    # 例: 全角スペースを半角スペースに変換
    cleaned = cleaned.replace("　", " ")

    return cleaned

# --- クリーニングの実験 ---

# ユーザーからの入力データ（表記がバラバラ）
raw_inputs = [
    "iPhone",
    "iphone",
    "  iPhone  ",   # 前後に空白がある
    "IPHONE",
    "Android",
    " android "
]

print("【処理前】")
# そのまま集計（カウント）しようとすると...
raw_counts = {}
```

```
for item in raw_inputs:
    if item not in raw_counts:
        raw_counts[item] = 0
    raw_counts[item] += 1
print(raw_counts)
# 結果: 全部バラバラにカウントされてしまいます

print("\n【処理後】")
clean_counts = {}
for item in raw_inputs:
    # ここでクリーニング関数を通す！
    clean_item = clean_text(item)

    if clean_item not in clean_counts:
        clean_counts[clean_item] = 0
    clean_counts[clean_item] += 1
print(clean_counts)
# 結果: {'iphone': 4, 'android': 2} と正しく集計されます
```

コードの解説です。

1. `text.strip()`：文字列の先頭と末尾にある空白文字（スペース、タブ、改行など）をすべて取り除きます。これだけで多くのトラブルが解決します。
2. `text.lower()`：アルファベットをすべて小文字に変換します。逆に大文字にする `upper()` もあります。
3. 効果: クリーニング関数を通すことで、見た目が少し違うだけのデータを同じものとして扱えるようになり、正しい集計結果が得られました。

欠損値の扱い

データの一部が空っぽ

データセットを見ていると、ある行の特定の項目だけ値が入っていないことがあります。これを**欠損値**（けっそんち）と呼びます。プログラミング言語では `null`、`None`、`NaN` (Not a Number) などと表現されます。

欠損値が含まれたまま計算（足し算や平均など）をしようとすると、プログラムがエラーで止まったり、計算結果がおかしくなったりします。そのため、分析前に何らかの対処が必要です。

主な対処法

1. **リストワイズ削除（除去）**：
    - 欠損値が含まれている行（データ）をまるごと捨ててしまう方法です。
    - 簡単ですが、データ数が減ったり、特定の傾向を持つデータだけが消えて結果が歪む（バイアスがかかる）リスクがあります。
2. **代入法（穴埋め）**：
    - 欠損している部分を、何らかの値で埋める方法です。
    - 平均値埋め: 全体の平均値を入れておく。当たり障りのない値にする手法です。
    - 0埋め: 売上データなどでデータがない場合、売上ゼロとみなして0を入れる場合もあります。

欠損の理由を考える

どの方法を選ぶべきかは、なぜ欠損しているのかという理由によります。

- 入力忘れ: 本当はあるはずの値なら、平均値などで埋めてもよいかもしれません。
- 該当なし: アンケートで「配偶者の有無」が「無」の人の「配偶者の年齢」欄が空欄の場合。これは0歳や平均年齢で埋めるのは間違いです。対象外という特別な値として扱う必要があります。

Pythonによる欠損値処理

テストの点数データの中に欠損（未受験など）があった場合を想定し、除去と平均値埋めの両方を試します。

```python
# python
# 欠損値（None）の処理方法を比較するプログラム

# テストの点数データ（None は欠損を表す）
scores = [80, 95, None, 70, None, 90]

def process_missing_values(data):
    print(f"元のデータ: {data}")

    # --- 方法1: 欠損があるデータを除外する ---
    # リスト内包表記を使って、Noneでないものだけを集めた新しいリストを作る
    # 「x is not None」は「xがNoneではない」という条件です
    dropped_data = [x for x in data if x is not None]

    print("\n【方法1: 除外】")
    print(f"データ: {dropped_data}")
    print(f"平均点: {sum(dropped_data) / len(dropped_data):.1f}")


    # --- 方法2: 平均値で埋める ---
    # まず、存在するデータだけで仮の平均を計算します
    # ここでも「Noneでないデータ」を使います
    valid_scores = [x for x in data if x is not None]
    mean_val = sum(valid_scores) / len(valid_scores)

    # None の場所を、計算した平均値に置き換えたリストを作る
    # （三項演算子を使っています: 「x if 条件 else 代わり」）
    filled_data = [x if x is not None else mean_val for x in data]

    print("\n【方法2: 平均値埋め】")
    print(f"埋める値: {mean_val:.1f}")
    print(f"データ: {filled_data}")
    # 見た目は平均値と同じになりますが、データ数は減っていません
    print(f"平均点: {sum(filled_data) / len(filled_data):.1f}")

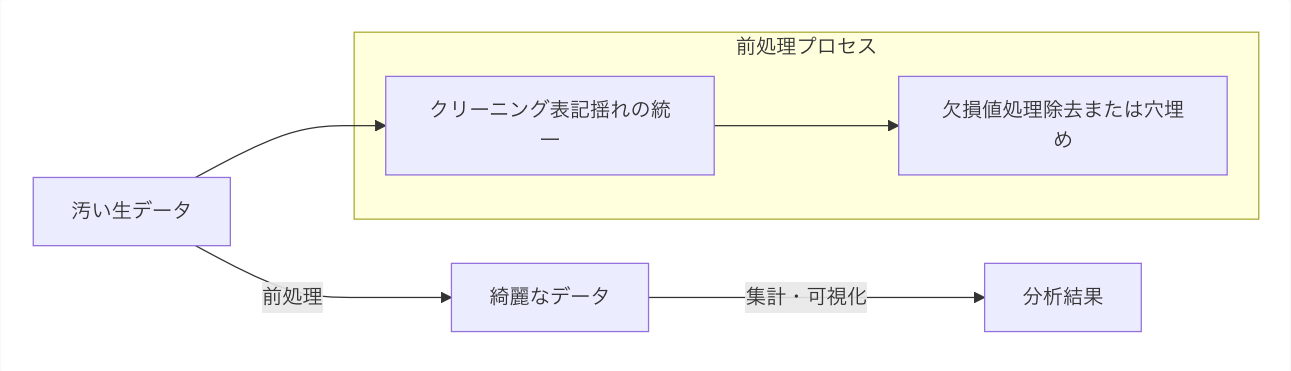
# --- 実行 ---
process_missing_values(scores)
```

コードの解説です。

1. `[x for x in data if x is not None]`：Pythonの強力な機能であるリスト内包表記を使っています。dataの中から、Noneではないxだけを取り出して新しいリストにする、という意味です。これでクリーニングされたリストが一瞬で作れます。
2. 結果の違い:
    - 除外: データ数が減ります（6個から4個）。
    - 埋め: データ数は変わりません（6個のまま）。欠損していた場所には、それらしい値（平均値）が入っています。
    - 今回の例では最終的な平均点は同じになりますが、その後の分析（例えば合計点の計算や偏差値の計算など）では結果が変わってきます。

前処理の流れまとめ

データ分析における前処理の位置づけを図で確認します。



図の解説です。 左側の汚い生データは、そのままでは分析に使えません。中央の前処理プロセスを通して、表記を揃えたり欠損値を埋めたりすることで、右側の綺麗なデータになります。この綺麗なデータを使って初めて、正しい分析結果が得られます。

データの可視化

データの集計や前処理が終わったら、次はその結果を人間に伝えるステップです。 数字の羅列をそのまま見せるのではなく、図や図表にして目に見える形にすることを**データの可視化**（かしか / Data Visualization）と呼びます。

コンピュータは大量の数字を扱うのが得意ですが、人間は数字のリストをパッと見ただけでは意味を理解しにくい。一方、長さや色や形といった視覚的な情報を処理する能力は高いです。可視化は、コンピュータが得意な数字を、人間が得意な視覚情報に翻訳する作業と言えます。

表とグラフの違い

データを提示する方法には、大きく分けて表とグラフの2つがあります。それぞれ得意な役割があるため、目的や相手に合わせて使い分けることが重要です。

表（Table）: 正確な数値を伝える

表は、行と列に数字を並べたものです。

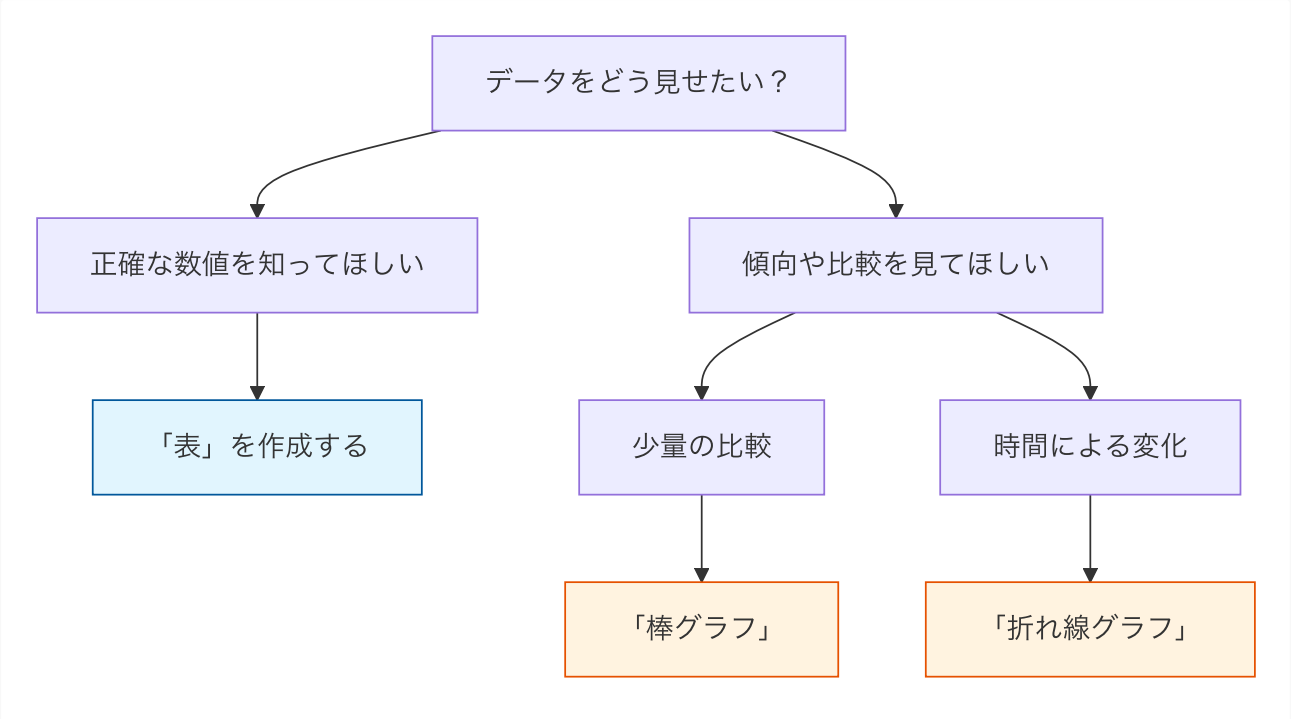
- 得意なこと: 個々のデータの正確な値を伝えること。特定の項目をピンポイントで調べること。
- 向いている場面: 領収書、名簿、詳細な売上報告など。
- 弱点: データ量が多くなると、全体の傾向や「どこが一番大きいか」を瞬時に判断するのが難しくなります。

グラフ（Graph）: 傾向と変化を伝える

グラフは、数字を長さや位置などの図形に置き換えたものです。

- 得意なこと: 全体の傾向（右肩上がりなど）やデータの大きさを直感的に比較すること。
- 向いている場面: プレゼン資料、ニュースの統計、状況のモニタリングなど。
- 弱点: 1つ1つの正確な数値（例えば100.45など）を読み取るのには適していません。

以下の図は、どちらを選ぶべきかの判断基準を表しています。



知りたいことが具体的な数字なら表を、全体の関係性や変化ならグラフを選ぶのが基本です。

棒グラフ・折れ線グラフ

グラフには多くの種類がありますが、まずは最も基本的な棒グラフと折れ線グラフの使い分けを押さえます。

棒グラフ（Bar Chart）：量の大小を比較する

それぞれのデータを棒の長さで表すグラフです。

- 使い所: カテゴリ（種類）ごとの違いを比べたいときに使います。
- 例: 店舗別の売上比較、好きな食べ物アンケートの結果、国別の人口など。
- ポイント: 棒の高さがそのまま量を表すため、一目でどこが一番多いか、どれくらい差があるかが分かります。

折れ線グラフ（Line Chart）：変化の流れを追う

データを点で打ち、それらを線で結んだグラフです。

- 使い所: 時間の経過とともに値がどう変化しているかを見たいときに使います。これを**時系列**（じけいれつ）データと呼びます。
- 例: 1日の気温の変化、株価の推移、Webサイトの訪問者数の変化など。
- ポイント: 線の傾きを見ることで、急激に増えているのか、緩やかに減っているのかといった変化の勢いを直感的に掴めます。

その他の代表的なグラフ

- 円グラフ（Pie Chart）: 全体を100%としたときの、各項目の割合を示すのに適しています。
- 散布図（Scatter Plot）: 2つのデータの関係性（例: 気温とアイスの売上の関係）を、点（プロット）の散らばりで見るのに適しています。

可視化のメリット

手間をかけてグラフを作る理由は、大きく3つあります。

1. **意思決定のスピードが上がる** 膨大な数字の羅列から結論を出すには時間がかかりますが、グラフなら問題箇所や好調な部分を一瞬で判断できます。

2. **異常値やパターンを発見しやすい** 先ほど取り上げた外れ値も、グラフにすると他のデータから大きく飛び出しているため、すぐに見つけられます。また、数字だけでは気づかなかった「週末だけ決まって数字が落ちる」といった法則性にも気づきやすくなります。
3. **人間の脳の特性を活かせる** 人間の脳は、論理的な思考をする前に、視覚的な刺激に対して直感的に反応します。この直感を味方につけることで、情報の理解度を格段に高められます。

Pythonによる簡易グラフの作成

本格的なグラフ作成には専用のライブラリ（Matplotlibなど）を使いますが、ここではプログラミングの基礎を活かして、文字を使ってコンソール画面に棒グラフを描画するプログラムを作成します。 数値を視覚的な長さ（文字の数）に変換する、可視化の核心部分を体験できます。

```
# python
# データを棒グラフとして表示するプログラム

def display_bar_chart(data_dict):
    """
    辞書形式のデータを受け取り、コンソールに棒グラフを表示する
    data_dict: {"カテゴリ名": 数値, ...}
    """
    print("--- アンケート結果（好きなプログラミング言語） ---")

    # グラフの見た目を調整するための設定
    # 棒グラフに使う文字
    bar_char = "■"

    # 1. データの最大値を見つける（グラフのスケールを調整するため）
    if not data_dict:
        print("データがありません")
        return

    # values() で数値だけを取り出し、max() で最大値を求めます
    max_value = max(data_dict.values())

    # 2. 各データを順番に取り出して描画する
    for label, value in data_dict.items():
        # ラベル（名前）を左側に表示
        # f-string の {label:10} は「10文字分の幅で表示する」という意味です
        row_label = f"{label:10}: "

        # 数値に応じた長さの棒を作る
        # 文字列に数値を掛けると、その回数分繰り返されます
        bar = bar_char * value

        # 最後に数値も添えて表示
        print(f"{row_label}{bar} ({value})")

# --- 可視化の実行 ---

# 集計済みのデータ（本当は前処理や集計ステップで作られます）
results = {
    "Python": 15,
    "JavaScript": 12,
    "TypeScript": 8,
    "Java": 5,
    "C++": 3
```

```
}

display_bar_chart(results)

print("\n【考察】")
print("グラフを見ると、Pythonが最も人気があり、C++の約5倍の規模であることが一目で分かりますね。")
```

コードの解説です。

1. `bar_char * value` : Pythonでは文字列に数値を掛けることで、その文字を繰り返すことができます。例えば `

演習（プログラミング問題）

データ分析の実践

データ分析の面白さは、実際に生データに触れ、そこから発見を得る瞬間にあります。 これまでに学んだファイルの読み込み、データの前処理、集計、可視化のすべてを使って、気象データの分析に挑戦します。

ステップ1: データの準備

まず、分析するためのデータが必要です。今回は気象庁などが公開しているオープンデータを模した、架空の2025年月別平均気温データを使用します。 リアルな練習のために、このデータには以下の汚れが含まれています。

1. 欠損値: データが取れていない月がある。
2. 型: CSVから読み込んだ直後はすべて文字列になっているため、数値への変換が必要。

まず、以下のコードを実行して、分析対象となる `weather.csv` ファイルを作成してください。

```
# python
# 演習用のダミーデータ(CSV)を作成する準備プログラム

# CSVファイルの中身になる文字列
csv_content = """month,temperature
1月,5.2
2月,6.5
3月,
4月,15.0
5月,20.2
6月,24.5
7月,28.3
8月,30.5
9月,26.8
10月,
11月,14.5
12月,8.8"""

# ファイルに書き込む
filename = "weather.csv"
# "w" は書き込みモード、encoding="utf-8" は文字化けを防ぐ設定です
with open(filename, "w", encoding="utf-8") as f:
    f.write(csv_content)

print(f"準備完了: {filename} を作成しました。")
```

ステップ2: AIへの指示とコード生成

この `weather.csv` を分析するプログラムを、AIと一緒に作ります。 プログラミングの現場では、すべてをゼロから書くのではなく、AIに大枠を作ってもらい、人間が微調整する方法が主流になりつつあります。

以下のようなプロンプト（指示）をAIに投げかけることを想像してください。

> ***プロンプト例:*** *「`weather.csv` というファイルがあります。1列目は月、2列目は気温です。 このファイルを読み込んで、以下の処理を行うPythonコードを書いてください。*
>
> 1. *気温が空欄（欠損）になっている行は除外してください。*
> 2. *月ごとの気温を数値として読み込んでください。*
> 3. *読み込んだデータを棒グラフ（テキスト）で表示してください。」*

AIは以下のようなコードを提案してくるはずです。コードを読み、これまでに学んだ知識を使って処理の流れを理解しましょう。

```python
# python
# AIと一緒に作成したデータ分析プログラム

def analyze_weather(filename):
    print(f"--- {filename} の分析開始 ---")

    data = {{}} # 読み込んだデータを格納する辞書

    try:
        # ファイルを開く（文字コードに注意）
        with open(filename, "r", encoding="utf-8") as f:
            # 1行目（ヘッダー: month,temperature）を読み飛ばす
            header = f.readline()

            # 2行目以降を1行ずつ処理
            for line in f:
                # 行末の改行を削除し、カンマで区切る
                line = line.strip()
                if not line: # もし空行ならスキップ
                    continue

                parts = line.split(",")

                # データ形式が正しいかチェック（要素が2つあるか）
                if len(parts) != 2:
                    continue

                month_name = parts[0]
                temp_str = parts[1]

                # 【前処理】
                # 1. 欠損値（空文字）のチェック
                if temp_str == "":
                    print(f"スキップ: {month_name} はデータがありません")
                    continue

                # 2. 型変換（文字列 -> 小数）
                try:
                    temperature = float(temp_str)
                except ValueError:
                    print(f"スキップ: {month_name} の気温データが不正です")
                    continue
```

```
                # 有効なデータとして保存
                data[month_name] = temperature

    except FileNotFoundError:
        print("エラー: ファイルが見つかりません。")
        return

    # 【可視化】
    print("\n--- 気温グラフ ---")
    for month, temp in data.items():
        # グラフの棒を作る（気温1度につきブロック1つ）
        # int()で整数にして、その数だけブロックを表示
        bar = "■" * int(temp)
        print(f"{month:4}: {bar} {temp}℃")

    # 【集計】
    if data:
        avg_temp = sum(data.values()) / len(data)
        print(f"\n年間平均気温（観測できた月のみ）: {avg_temp:.1f}℃")
    else:
        print("有効なデータがありませんでした。")

# 実行
analyze_weather("weather.csv")
```

コードの解説です。

1. ファイル読み込み: `open()` 関数でファイルを開き、`readline()` でヘッダーを飛ばし、`for` ループで1行ずつデータを取得しています。
2. 前処理（クリーニング）:
   - `strip()` で改行などの余計な空白を削除。
   - `split(",")` でCSVのルールに従ってデータを分割。
   - `if temp_str == "":` で欠損値を検出し、その行を無視（スキップ）しています。
3. 型変換: ファイルから読んだデータはすべて文字として扱われるため、`float()` を使って計算できる数値に変換しています。
4. 可視化: テキスト文字（ ■ ）を使って棒グラフを描画し、暑い月と寒い月が一目でわかるようにしています。

実行結果の確認

上記のコードを実行すると、以下のような出力が得られます。

```
--- weather.csv の分析開始 ---
スキップ: 3月 はデータがありません
スキップ: 10月 はデータがありません

--- 気温グラフ ---
1月 : ■■■■■■■■ 5.2℃
2月 : ■■■■■■■■■ 6.5℃
4月 : ■■■■■■■■■■■■■■■■■■■■■ 15.0℃
5月 : ■■■■■■■■■■■■■■■■■■■■■■■■■■■ 20.2℃
6月 : ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ 24.5℃
7月 : ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ 28.3℃
8月 : ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ 30.5℃
9月 : ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ 26.8℃
```

```
11月 : ████████████████████ 14.5℃
12月 : ██████████ 8.8℃

年間平均気温（観測できた月のみ）: 18.0℃
```

欠損していた3月と10月が自動的に無視され、データがある月だけでグラフと平均値が計算されていることが確認できます。 プログラムを使えば、汚れたデータが含まれていても瞬時に正しい分析結果を導き出せます。

問題1 基本統計量の計算

問題文

N 個の整数が与えられます。これらの整数について、個数・合計・平均を求めてください。

平均は小数第 1 位まで出力してください。

入力

入力は以下の形式で与えられます。

```
N
a_1 a_2 ... a_N
```

1 行目に整数の個数 N が与えられます。 2 行目に N 個の整数がスペース区切りで与えられます。

制約

- 1 ≤ N ≤ 1000
- -1000 ≤ a_i ≤ 1000

出力

以下の 3 行を出力してください。

```
Count: (個数)
Sum: (合計)
Average: (平均)
```

平均は小数第 1 位まで出力してください。

入力例1

```
5
10 20 30 40 50
```

出力例1

```
Count: 5
Sum: 150
Average: 30.0
```

入力例2

```
3
1 2 3
```

出力例2

```
Count: 3
Sum: 6
Average: 2.0
```

問題2 グループ別集計

問題文

N 件のデータが与えられます。各データはカテゴリ名と数値の組です。カテゴリごとに数値の合計を求め、カテゴリ名の辞書順に出力してください。

入力

入力は以下の形式で与えられます。

```
N
category_1 value_1
category_2 value_2
...
category_N value_N
```

1 行目にデータの件数 N が与えられます。 続く N 行にカテゴリ名と数値がスペース区切りで与えられます。

制約

- 1 ≤ N ≤ 1000
- カテゴリ名は英小文字のみからなる長さ 1 以上 20 以下の文字列
- 1 ≤ value_i ≤ 10000

出力

カテゴリ名の辞書順に、各カテゴリの合計値を以下の形式で出力してください。

```
category total
```

入力例1

```
5
apple 100
banana 200
apple 150
banana 50
cherry 300
```

出力例1

```
apple 250
banana 250
cherry 300
```

入力例2

```
3
x 10
y 20
x 30
```

出力例2

```
x 40
y 20
```

問題3 欠損値の処理

問題文

N 個の値が与えられます。各値は整数または欠損値を表す "NA" です。欠損値を除いた有効なデータについて、個数・合計・平均を求めてください。

平均は小数第 1 位まで出力してください。有効なデータが 1 つも存在しない場合は "NO DATA" と出力してください。

入力

入力は以下の形式で与えられます。

```
N
v_1 v_2 ... v_N
```

1 行目に値の個数 N が与えられます。 2 行目に N 個の値がスペース区切りで与えられます。各値は整数または "NA" です。

制約

- 1 ≤ N ≤ 1000
- 整数の値は -1000 以上 1000 以下

出力

有効なデータが存在する場合、以下の 3 行を出力してください。

```
Count: (有効な値の個数)
Sum: (有効な値の合計)
Average: (有効な値の平均)
```

平均は小数第 1 位まで出力してください。

有効なデータが 1 つも存在しない場合は、以下の 1 行を出力してください。

```
NO DATA
```

入力例1

```
5
10 NA 30 NA 50
```

出力例1

```
Count: 3
Sum: 90
Average: 30.0
```

入力例2

```
3
NA NA NA
```

出力例2

```
NO DATA
```

第10章 CLIアプリケーションの開発方法

前節までで開発環境を整え、AIを活用する準備ができました。 本節からは実際に動くソフトウェアを作っていきます。 最初に取り組むのは、CLIアプリケーションです。

CLI（コマンドラインインターフェース）

普段、スマートフォンやパソコンを使うとき、画面上のアイコンをタップしたりクリックしたりして操作しています。 しかしプログラミングの世界、とくに開発の現場では、アイコンやボタンがない、テキストだけでコンピュータとやり取りする画面（コンソール画面）を目にする機会が急増します。

この文字を使ってコンピュータを操作する画面こそが、CLIです。

CLIとは何か

CLIはCommand Line Interface（コマンドライン・インターフェース）の略称です。 インターフェースとは「接点」「境界面」という意味で、人間とコンピュータが情報をやり取りする窓口を指します。

文字で対話する

CLIの最大の特徴は、文字（コマンド）だけを使ってコンピュータに命令を出す点です。 マウスでフォルダをダブルクリックして開く代わりに、フォルダを開けという意味の単語（コマンド）をキーボードで打ち込みます。

一見すると不便で古臭い方法に見えるかもしれません。しかし、プロのエンジニアがCLIを愛用するには理由があります。 慣れてしまえば**圧倒的に高速**かつ**正確**に作業できるからです。

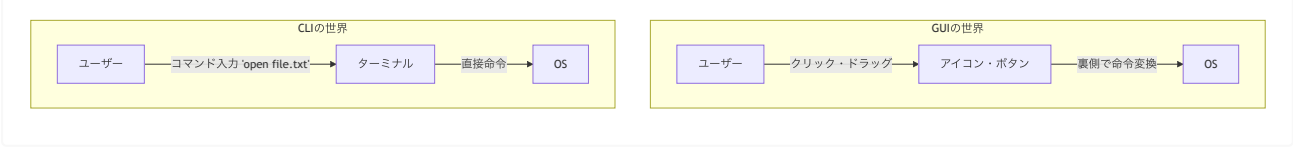
例えば、1000個ある画像ファイルの名前をすべて撮影日時に変更する作業をマウス操作で行うと、何時間もかかります。 しかしCLIで適切な命令を1行書けば、数秒で完了します。この自動化のしやすさこそ、CLIが現代でも必須スキルとされる理由です。

GUI（グラフィカルユーザーインターフェース）との違い

普段慣れ親しんでいる、グラフィック（絵やアイコン）を中心とした操作画面をGUI（Graphical User Interface）と呼びます。CLIとGUIの違いを比較します。

特徴	GUI（グラフィカル）	CLI（コマンドライン）
操作方法	マウス、タッチ、ジェスチャー	キーボード入力のみ
直感性	見ればなんとなく使い方がわかる	コマンド（単語）を知らないと何もできない
正確性	操作ミス（押し間違い）が起きやすい	言語化して命令するため、正確に伝わる
リソース	画面描画のために多くのメモリやCPUを使う	文字だけなので軽量
主な用途	Web閲覧、動画視聴、お絵かき	開発、システム管理、大量データ処理

違いをイメージ図で理解する



GUIは案内係（アイコン）を通してコンピュータにお願いするようなものです。直感的ですが、案内係ができることしか頼めません。 一方CLIは、コンピュータと直接言葉で話すようなものです。言葉を覚える必要はありますが、細かい要望を直接伝えることができ、案内係がいない場所（サーバー内部など）でも操作できます。

また、多くの開発者向けツールは、まずCLIとして作られます。GUIを作るには画面のデザインやボタンの配置を考える手間がかかりますが、CLIなら機能そのものの開発に集中できるからです。

CLIが使われる場面

CLIが活躍する具体的な場面を挙げます。

1. **サーバーの設定・管理**: Webサイトやアプリを動かしているサーバーの多くは、画面（ディスプレイ）が接続されておらず、遠隔地から文字だけで操作します。
2. **大量ファイルの処理**: 特定の拡張子のファイルだけを探して削除する、ログファイルから特定のエラーだけを抜き出す、といった処理はCLIが得意とするところです。
3. **開発ツールの実行**: 前節で使った `node` や `python`、`npm`、`pip` といったコマンドはすべてCLIツールです。例えば、`node` はNode.js（JavaScriptを実行する環境）を起動するコマンドを指します。
4. **バージョン管理 (Git)**: プログラムの変更履歴を記録するGitというツールも、複雑な操作を行う際はCLIを使うのが一般的です。

CLIは大量の処理、自動化、開発効率に強く、エンジニアにとって欠かせない道具です。

コンソール画面への恐怖心を払拭する

初心者にとって、コンソール画面は「何か間違えたらパソコンが壊れるのでは」と恐怖を感じる場所かもしれません。 確かに強力な権限を持っていますが、正しく恐れ、基本ルールを守れば決して怖いものではありません。

むしろCLIを使えるようになると、コンピュータの中身を直接コントロールしている感覚が得られ、エンジニアとしての視座が高まります。 車のボンネットを開けてエンジンの仕組みを理解し、自分でメンテナンスできるようになる感覚に似ています。

実際にCLIを体験する

百聞は一見にしかずです。ターミナルで簡単なコマンドを実行し、その便利さを体感します。 ここでは画面に文字を表示する `echo`（エコー）コマンドを使います。

以下のコマンドを、ご自身のターミナル（PowerShell や Terminal.app）に入力して Enter キーを押してください。

```
# bash
# echoコマンド: 指定した文字列をそのまま画面に「こだま（echo）」のように返します
echo "Hello, CLI World!"

# 計算結果を表示するわけではありません（文字として扱われます）
echo "1 + 1"
```

解説です。

`echo` の動作を3点に整理します。

- `echo`：後に続く文字を表示せよという命令（コマンド）です。
- このコマンドを打った瞬間、キーボード入力はシェル（命令の解釈係）に渡され、コンピュータがそれを実行し、結果を画面に返します。
- これがCLIにおける入力、処理、出力の最小単位です。

この流れを理解すると、CLIの操作は会話のキャッチボールだと捉えられるようになります。

たったこれだけですが、マウスを使わずにコンピュータに命令を下しました。 次節からは、このCLI上で動く便利なアプリケーションを、AIと協力して実際に作っていきます。

CLIアプリケーションの例

CLIという環境に少し慣れたところで、これから作っていくCLIアプリケーションの具体的なイメージを膨らませていきます。

プログラミングを学ぶと、OSに最初から入っている基本的なコマンドだけでなく、自分だけの特別な道具を作れるようになります。代表的な3つのパターンを紹介します。

ファイル操作ツール

コンピュータを使っていると、単純だけど数が多い作業に遭遇することがあります。これを自動化するのがファイル操作ツールの役割です。

面倒な作業を自動化する

例えば、デジカメやスマートフォンからパソコンに写真を転送すると、`IMG_0001.jpg`, `IMG_0002.jpg` ... といった味気ないファイル名になることがあります。これらを「2026-01-28_旅行_01.jpg」のように、撮影日とイベント名を含んだ名前に書き換えたいとします。

ファイルが10枚なら手作業でも終わりますが、1000枚あったらどうなるか。GUIで一つ一つ右クリックして名前を変更するのは苦行です。 CLIアプリケーション（スクリプト）を書けば、この1000回の操作を**一瞬で**終わらせることができます。

他にも、以下のようなツールがよく作られます。

- 掃除ツール: ダウンロードフォルダの中で、1ヶ月以上開いていないファイルだけをゴミ箱に移動する
- 整理ツール: 散らかったファイルを、拡張子（.jpg, .txt, .pdf）ごとにフォルダを分けて自動で振り分ける

ここでいう**拡張子**とは、ファイル名の末尾につく「.jpg」「.txt」のような部分で、ファイルの種類を表します。

どちらも面倒な作業を機械に任せるための典型例で、CLIの強みがよく表れます。

【コード例】特定のファイルを探し出す（Python）

現在いるフォルダの中に、どんなテキストファイル（.txt）があるかを探し出して一覧表示する、シンプルなPythonプログラムです。 ファイル操作の第一歩であるファイルを見つけるという処理を行っています。

```python
# python
import glob  # ファイルのパターンマッチングを行うためのライブラリ（モジュール）
import os    # OS（オペレーティングシステム）の機能を使うためのライブラリ

def list_text_files():
    print("--- テキストファイル検索ツール ---")

    # テスト用に一時的なファイルを作成します（学習用）
    # 実際には、すでに存在するファイルを操作することが多いです
    with open("memo.txt", "w") as f:
        f.write("これはメモです。")
    with open("todo.txt", "w") as f:
        f.write("買い物に行く")

    # 現在のフォルダ（.）にある .txt で終わるファイルを探します
    # glob.glob は、指定したパターンに一致するファイル名のリストを返します
    files = glob.glob("*.txt")

    if not files:
        print("テキストファイルは見つかりませんでした。")
    else:
        print(f"{len(files)} 個のテキストファイルが見つかりました:")
        for file_name in files:
            print(f"- {file_name}")

    # 後始末（作成したテストファイルを削除）
    # os.remove はファイルを削除する命令です
```

```
    os.remove("memo.txt")
    os.remove("todo.txt")
    print("\n検索と後始末が完了しました。")

if __name__ == "__main__":
    list_text_files()
```

解説です。

ファイル探索の仕組みを4点で確認します。

- `import glob` : `glob` （グロブ）という、ワイルドカード（*）を使ってファイルを探すための便利な道具箱を準備しています。
- `glob.glob("*.txt")` : 拡張子が .txt であるすべてのファイルという意味です。 `*` は任意の文字を表します。
- `for file_name in files` : 見つかったファイルを一つずつ取り出して処理（ここでは画面に表示）しています。
- プログラムを使えば、探す、表示する、消すといった操作を自由に組み合わせることができます。

この例はファイルを探すという最小単位の処理を体験するためのものです。

データ変換ツール

データには様々な形式（フォーマット）があります。ある形式を別の形式に変換するツールも、CLIでよく作られます。これらは**コンバータ**とも呼ばれます。 CSV（カンマ区切りの表形式）やJSON（キーと値の組み合わせで表す形式）は、その代表例です。

異なるシステムの間をつなぐ

仕事や開発の現場では、データの形式変換が必要になる場面が多々あります。

- CSVからJSONへ: Excelなどの表計算ソフトで管理しているデータ（CSV）を、Webアプリケーションで使いやすい形式（JSON）に変換する。
- 画像の加工: 高画質な写真画像を、Webサイト表示用に小さなサイズにリサイズしたり、形式を（PNGからJPEGへ）変換したりする。
- 文字コードの変換: 古いシステムで作られたShift_JIS形式のテキストファイルを、現代的なUTF-8形式に変換して文字化けを直す。

このような形式の違いを吸収する作業は、CLIツールが最も得意とする分野です。

GUIアプリを開いてファイルから書き出しを行う代わりに、コマンド一発で実行できるようにしておくと効率的です。

【コード例】CSVデータをJSON形式に変換する（JavaScript / Node.js）

カンマ区切りのデータ（CSV）を、プログラムで扱いやすいオブジェクト形式（JSON）に変換して表示するNode.jsのコードです。 Node.js（ノードジェーエス）は、JavaScriptをブラウザの外で実行できるようにする実行環境です。

```
// javascript
// サンプルのCSVデータ（本来はファイルから読み込みますが、ここでは変数に入れます）
// 1行目はヘッダー（項目名）、2行目以降がデータです
const csvData = `id,name,role
1,Alice,Engineer
2,Bob,Designer
3,Charlie,Manager`;

console.log('--- CSVデータ（変換前） ---');
console.log(csvData);
console.log('\n--- JSONデータ（変換後） ---');

// 1. 文字列を行ごとに分割する
// split("\n") は、改行コードで区切って配列にします
```

```
const lines = csvData.split('\n');

// 最初の行はヘッダー（id, name, role）として取得
const headers = lines[0].split(',');

// 2行目以降のデータを処理して、オブジェクトのリストを作る
const jsonData = lines.slice(1).map((line) => {
  const values = line.split(','); // カンマで区切って値を取り出す
  const obj = {};

  // ヘッダーと値を対応付けてオブジェクトを作る
  // 例: { "id": "1", "name": "Alice", "role": "Engineer" }
  headers.forEach((header, index) => {
    obj[header] = values[index];
  });

  return obj;
});

// JSON形式の文字列として整形して表示
// JSON.stringify(データ, null, 2) で見やすくインデント（字下げ）して表示します
console.log(JSON.stringify(jsonData, null, 2));
```

解説です。

入力、処理、出力の3点に分けて整理します。

- 入力: 単なるカンマ区切りの文字列です。人間には読めますが、コンピュータがBobの役割を問われても即答しづらい形式です。
- 処理: 文字列をバラバラに分解（ `split` ）し、意味のある構造（オブジェクト）に組み立て直しています。
- 出力: JSON（ジェイソン）形式です。プログラムからBobのデータとして扱いやすい形になりました。

入力、処理、出力を意識すると、データ変換の目的が明確になります。

API連携ツール

自分のパソコンの中だけでなく、インターネットの向こう側にあるWebサービスと連携するツールです。これが作れるようになると、CLIの世界は大きく広がります。

インターネットの情報をコマンドで操る

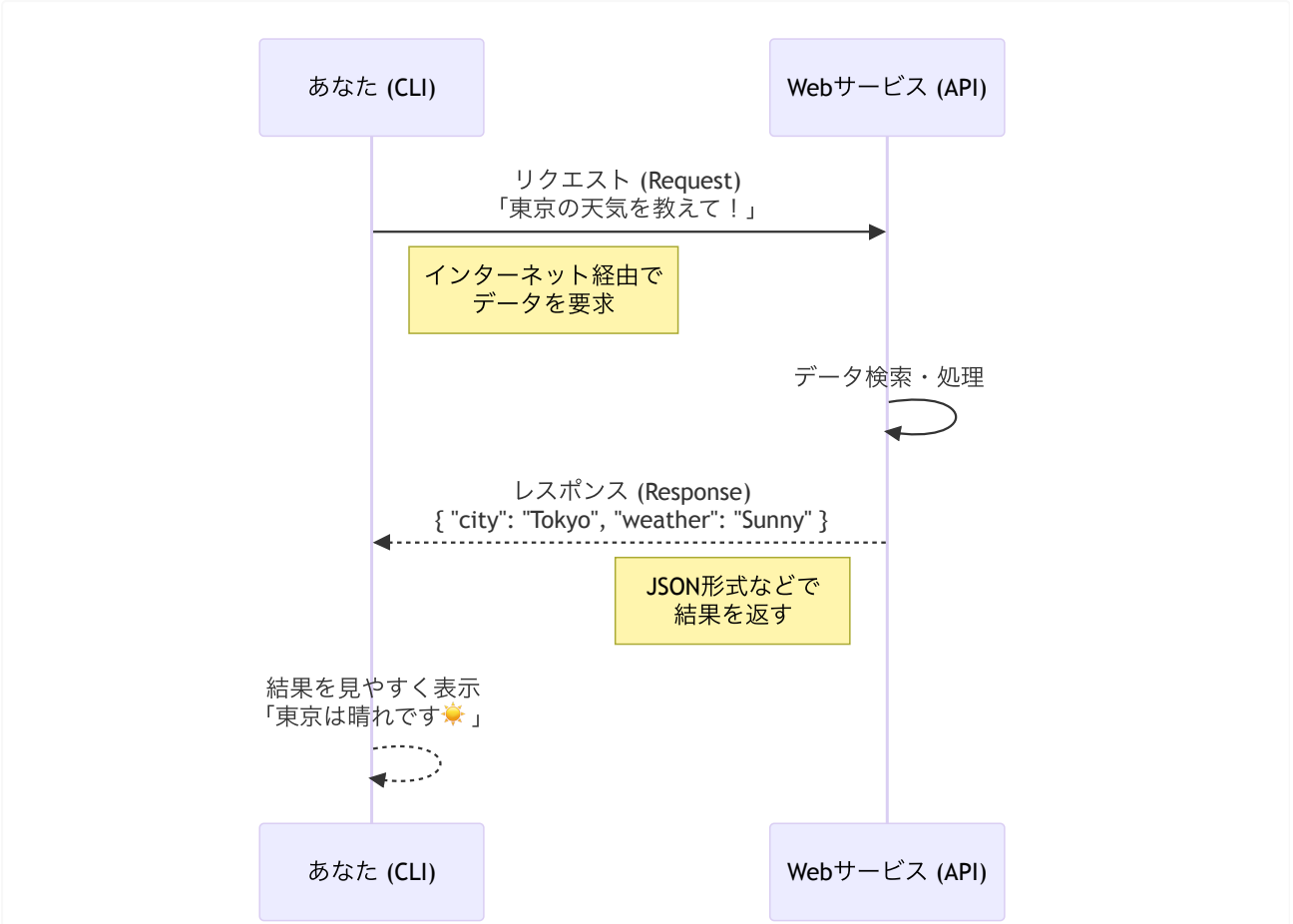
Webサービス（GoogleマップやX、天気予報サイトなど）の多くは、人間が見るための画面だけでなく、プログラムが利用するための窓口であるWeb API（ウェブ・エーピーアイ）を提供しています。

これを利用すると、ブラウザを開かなくても、ターミナルから直接インターネット上の情報を取得したり操作したりできます。

- 天気予報ツール: `weather tokyo` と打つだけで、今日の東京の天気を表示する。
- 翻訳ツール: `trans "Hello"` と打つと、「こんにちは」と返ってくる。
- 通知ツール: 重たい処理が終わったら、自分のスマホやSlackに通知を送る。

外部サービスを利用すれば、CLIから扱える情報の幅が一気に広がります。

Web APIとの通信の仕組み



この図は、リクエストを送り、返ってきたデータを整形して表示するという最小の流れを示しています。実際のAPI連携も、この基本構造の繰り返しです。

【コード例】Web APIからデータを取得する（Python）

実際にインターネット上の無料テスト用APIにアクセスし、データを取得するPythonコードです。

※実行にはインターネット接続が必要です。

```python
# python
import urllib.request # URLにアクセスするための標準ライブラリ
import json           # JSONデータを扱うためのライブラリ

def fetch_todo_data():
    print("--- API連携デモ: ToDoデータの取得 ---")

    # テスト用のWeb APIのURL（JSONPlaceholderという無料サービスを利用）
    # ここでは「IDが1のToDoデータ」を要求しています
    url = "https://jsonplaceholder.typicode.com/todos/1"

    try:
        print(f"[{url}] にアクセス中...")

        # 1. リクエストを送り、レスポンスを受け取る
        # with文を使うことで、通信が終わったら自動的に接続を閉じます
        with urllib.request.urlopen(url) as response:
            # 2. 受け取ったデータ（バイト列）を読み込み、文字列に変換（デコード）する
```

```
            body = response.read().decode("utf-8")

            # 3. 文字列をPythonの辞書（データ構造）に変換する
            data = json.loads(body)

            # 結果を見やすく表示
            print("\n取得成功！データの中身:")
            print(f"タイトル: {data['title']}")
            print(f"完了状態: {'完了' if data['completed'] else '未完了'}")
            print(f"生のデータ: {data}")

    except Exception as e:
        # 通信エラーなどが起きた場合の処理
        print(f"\nエラーが発生しました: {e}")
        print("インターネット接続を確認してください。")

if __name__ == "__main__":
    fetch_todo_data()
```

解説です。

API通信の基本要素を3点で整理します。

- `urllib.request.urlopen(url)`：ブラウザがURLを開くのと同じように、プログラムから指定したアドレスにアクセスします。
- `json.loads(body)`：APIから返ってくるデータは多くの場合JSON形式の文字列です。これをPythonで扱えるデータ型（辞書）に変換（パース）しています。
- この仕組みを応用すれば、天気のAPIや株価のAPIにURLを変えるだけで、様々な情報を取得するツールが作れます。

押さえるべき核心はURLにアクセスし、JSONを読むことです。 これらの例のように、CLIアプリケーションはファイル操作、データ変換、外部通信といった基本要素を組み合わせることで、作業を劇的に効率化してくれます。

バイブコーディングによるCLIアプリケーションの開発

これまでは用意された生成例コードを見て学びましたが、ここからは**自分自身が作り手**になります。 とはいえ、最初からすべてを独力で書く必要はありません。強力なパートナーである生成AIがいます。

AIと対話しながらリズムよくコードを書き進めていくスタイル、それがバイブコーディングです。 実際にCLIツールを開発するプロセスを通じて、**要件を言葉にし、AIに伝わる形に整える力**を身につけます。

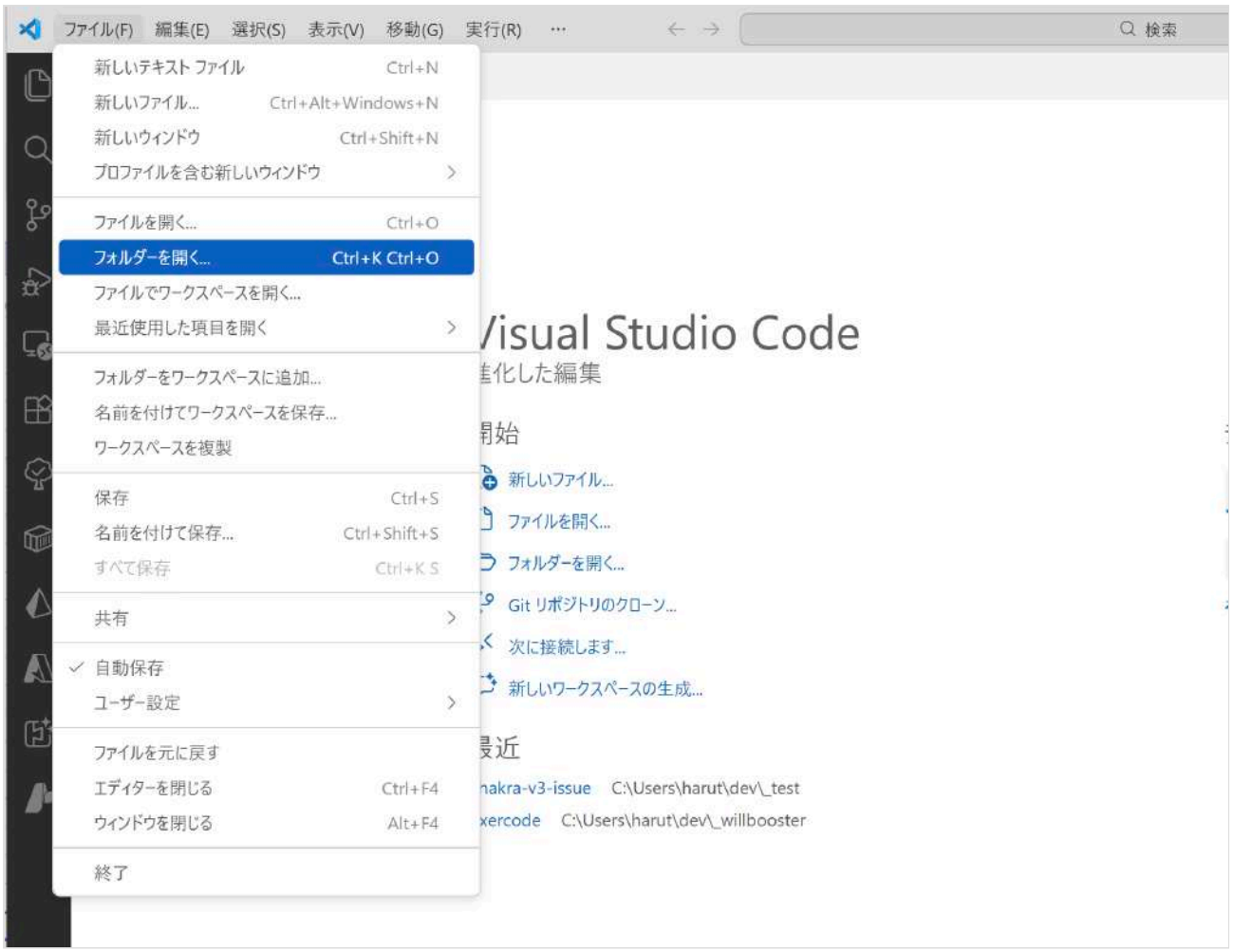
作業場所の準備

まず、CLIツールのファイルを置く作業フォルダーを準備します。作業フォルダーとは、作ったコードや設定ファイルをひとまとめに保管する場所のことです。

作業の流れは次の通りです。

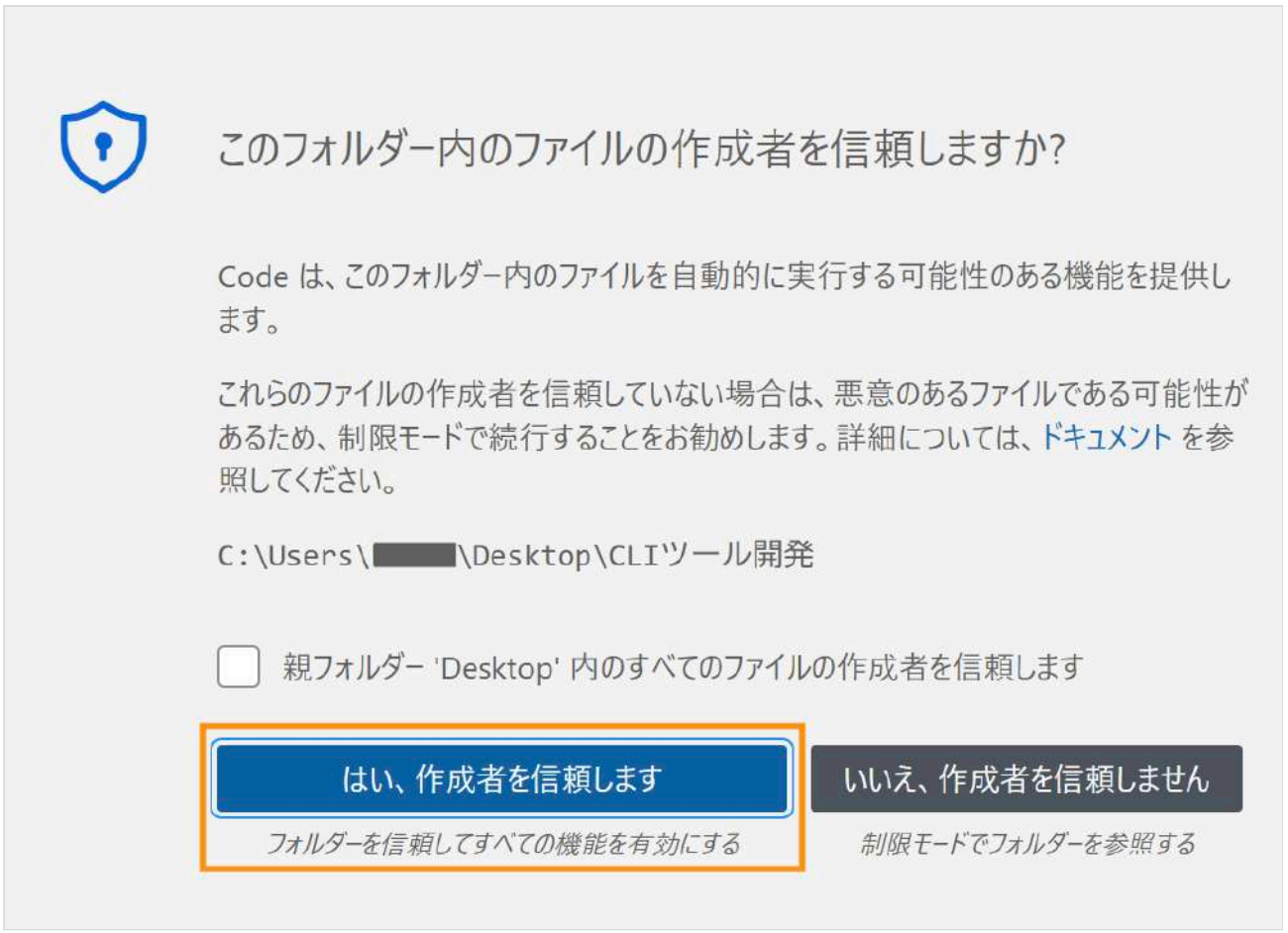
1. デスクトップ上で右クリックをして、「新規作成」から「フォルダー」を選択します。
2. フォルダーの名前を「CLIツール開発」に変更します。
3. VS Codeを起動します。
4. メニューから「ファイル」の「フォルダーを開く」を選択します。

ここまでの操作で、作業フォルダーをVS Codeで開く準備が整います。



次に、先ほど作成したデスクトップ上の「CLIツール開発」フォルダーを選択し、「フォルダーの選択」ボタンをクリックしてください。

この時点で「このフォルダーのファイルを信頼しますか？」という確認が表示されることがあります。VS Codeが安全のために初回のみ確認しているだけです。



表示された場合は、「はい、作成者を信頼します」をクリックして先に進みます。これで準備は完了です。

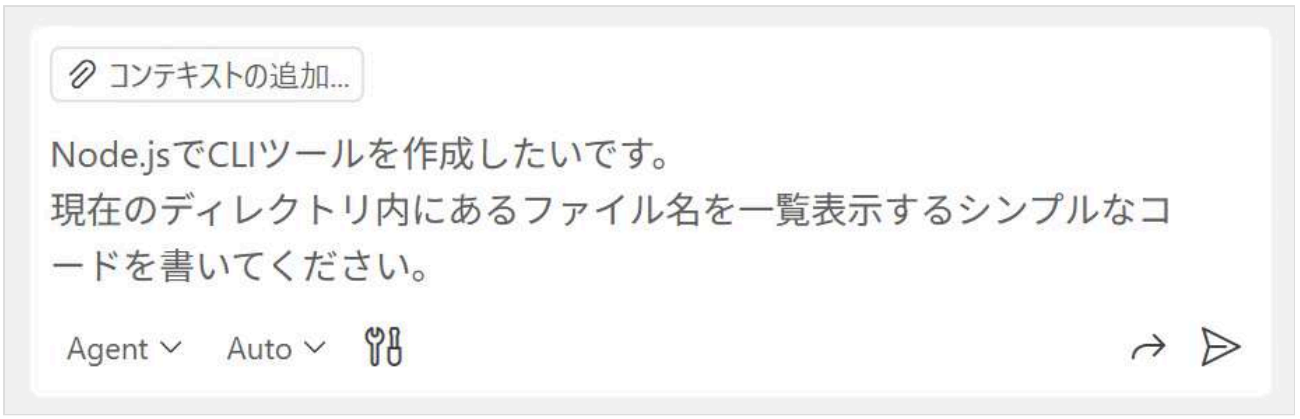
AIチャットの起動と指示の入力

次に、AIチャットに作りたいCLIツールの要件を伝えます。重要なのは、プロンプトの暗記ではなく、何を作りたいかを具体的に言葉にすることです。

まず、画面右側のチャット画面で、 「Agent」と、モデルが「Auto」に設定します。



チャット画面の下部にある入力フォームに、作成したいツールの要件を入力し、紙飛行機のマーク（送信ボタン）を押します。



入力する指示の例は以下の通りです。あくまで一例なので、同じ内容でも表現は自由に変えて構いません。

```
Node.jsでCLIツールを作成したいです。
現在のディレクトリ内にあるファイル名を一覧表示するシンプルなコードを書いてください。
```

言語とやりたいことを明確に伝えると、AIは必要なコードを生成しやすくなります。

途中でチャット欄に [許可] または [Allow] というボタンが表示されることがあります。これは、AIが環境へのアクセス権を求めている合図です。



表示された場合は、内容を確認して許可してください。ここまで進めば、コード生成が行われます。

生成例コードと実行イメージ

AIが出力するコードは人によって異なりますが、次のような形になることが多いです。以下は生成例としての参考コードです。

ファイル名: `index.js`

```
import fs from 'fs';
import path from 'path';

// カレントディレクトリ（現在の作業場所）を取得
const currentDir = process.cwd();

console.log(`📂 ${currentDir} のファイル一覧:\n`);

try {
  // ディレクトリの内容を読み込む
  const files = fs.readdirSync(currentDir, { withFileTypes: true });

  if (files.length === 0) {
    console.log('ファイルがありません');
    process.exit(0);
  }

  // ファイルとディレクトリを分類
  const directories = files.filter((f) => f.isDirectory());
  const normalFiles = files.filter((f) => !f.isDirectory());

  // ディレクトリを表示
  if (directories.length > 0) {
    console.log('📂 ディレクトリ:');
    directories.forEach((dir) => {
      console.log(`   ${dir.name}/`);
    });
    console.log();
  }

  // ファイルを表示
  if (normalFiles.length > 0) {
    console.log('📄 ファイル:');
    normalFiles.forEach((file) => {
      const filePath = path.join(currentDir, file.name);
      const stats = fs.statSync(filePath);
      const sizeKB = (stats.size / 1024).toFixed(2);
      console.log(`   ${file.name} (${sizeKB} KB)`);
    });
  }

  console.log(`\n合計: ${directories.length} ディレクトリ, ${normalFiles.length} ファイル`);
} catch (error) {
  console.error('エラーが発生しました:', error.message);
  process.exit(1);
}
```

上のコードは、現在のディレクトリを調べ、ファイルとフォルダーを分けて一覧表示する仕組みです。ここでいう**ディレクトリ**はフォルダーと同じ意味で、**カレントディレクトリ**は今作業している場所を指します。

実行すると、以下のようにターミナルに結果が表示されます（表示内容は環境によって異なります）。

```
PS C:\Users\user\Desktop\CLIツール開発> node index.js
📂 C:\Users\user\Desktop\CLIツール開発 のファイル一覧:
```

ファイル:
index.js (1.39 KB)
package.json (0.36 KB)

合計: 0 ディレクトリ, 2 ファイル

✓ Read CLIツール開発 file

Node.js CLIツールの基本的なプロジェクト構成を作成します。

✓ {} package.json が作成されました

✓ JS index.js が作成されました

完成しました！Node.js CLIツールを作成しました。以下の2つのファイルを作成しています：

{} package.json

- プロジェクトの設定ファイル
- `npm start` で実行できるようにしています

JS index.js

- メインのCLIスクリプト
- カレントディレクトリのファイル・ディレクトリを一覧表示
- ファイルサイズも表示
- ディレクトリと通常ファイルを分類して表示

使い方:

```
node index.js
# または
npm start
```

機能：

- 📁 ディレクトリと 📄 ファイルをアイコン付きで区別
- ファイルサイズをKB単位で表示
- ファイル数とディレクトリ数を集計
- エラーハンドリング

追加機能の相談と拡張

ツールが動いたら、次は機能を追加します。ここでも、何を追加したいかを具体的に伝えることが重要です。

例えば、ファイルサイズの大きい順に並び替える機能を追加したい場合、次のように依頼します。

先ほどのコードに、ファイルサイズの大きい順に並び替えて表示する機能を追加してください。



追加後のコードも、AIの出力によって細部は変わりますが、以下のような形になることが多いです。

ファイル名: `index.js`（更新後の例）

```
import fs from 'fs';
import path from 'path';

// カレントディレクトリを取得
const currentDir = process.cwd();

console.log(`📁 ${currentDir} のファイル一覧:\n`);

try {
  // ディレクトリの内容を読み込む
  const files = fs.readdirSync(currentDir, { withFileTypes: true });
```

```
  if (files.length === 0) {
    console.log('ファイルがありません');
    process.exit(0);
  }

  // ファイルとディレクトリを分類
  const directories = files.filter((f) => f.isDirectory());
  const normalFiles = files.filter((f) => !f.isDirectory());

  // ディレクトリを表示
  if (directories.length > 0) {
    console.log('📂 ディレクトリ:');
    directories.forEach((dir) => {
      console.log(`   ${dir.name}/`);
    });
    console.log();
  }

  // ファイルを表示
  if (normalFiles.length > 0) {
    console.log('📄 ファイル（サイズ順）:');

    // ファイルサイズを取得して並び替え
    const filesWithSize = normalFiles.map((file) => {
      const filePath = path.join(currentDir, file.name);
      const stats = fs.statSync(filePath);
      return {
        name: file.name,
        size: stats.size,
      };
    });

    // サイズの大きい順にソート
    filesWithSize.sort((a, b) => b.size - a.size);

    // ソート済みのファイルを表示
    filesWithSize.forEach((file) => {
      const sizeKB = (file.size / 1024).toFixed(2);
      console.log(`   ${file.name} (${sizeKB} KB)`);
    });
  }

  console.log(`\n合計: ${directories.length} ディレクトリ, ${normalFiles.length} ファイル`);
} catch (error) {
  console.error('エラーが発生しました:', error.message);
  process.exit(1);
}
```

表示結果は、ファイルがサイズ順に並ぶだけでなく、どの順序で表示されているかが一目で分かるため、作業効率が上がります。

要件の整理と目的の言語化

AIにコードを書かせる前に、人間がやらなければならない最も重要な仕事があります。 それは、何を作りたいのかを明確にすることです。これが曖昧なままAIに指示を出しても、期待通りのものは出てきません。

入力、処理、出力を明確な言葉にする

プログラムの基本構造であるIPO（Input-Process-Output）を思い出してください。作りたいツールについて、この3点を言葉で書き出します。 目的は、プロンプトを暗記することではなく、目的と処理を整理することです。

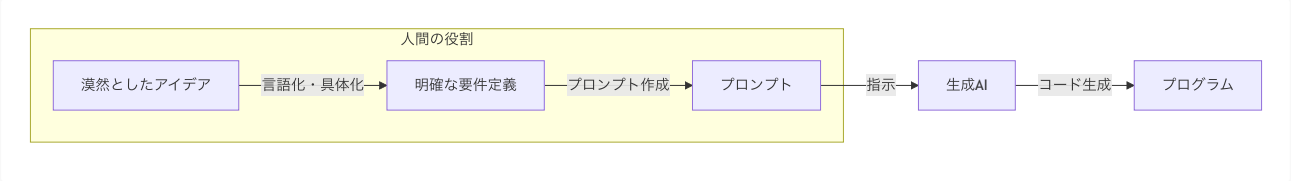
例えば、文字数を数えるツールを作りたいと思ったとします。これだけでは不十分です。以下のように具体化します。

1. **入力 (Input)**:
   - ユーザーはどのようにデータを与えるか？
   - コマンドライン引数としてファイルパス（ファイルの名前）を渡す。
2. **処理 (Process)**:
   - プログラムは何をするか？
   - 渡されたファイルを読み込む。
   - ファイルの中身の文字数と行数を数える。
3. **出力 (Output)**:
   - 結果をどう見せるか？
   - ターミナル画面に「文字数: 〇〇, 行数: △△」のように表示する。

こう分解すると、AIに伝えるべき情報が自然に整理されます。

AIへの指示の質は、要件定義の具体性に依存する

この要件の整理が、そのままAIへの指示の質に直結します。 AIは優秀なプログラマですが、あなたの心の中までは読めません。「いい感じにして」ではなく、設計図を言葉で渡す必要があります。



この図のポイントは、AIに渡す情報の質は人間側の整理に左右されるという点です。要件が明確になるほど、出力のブレは小さくなります。

AIに伝わる言葉の組み立て

要件が固まったら、それをAIに伝わる言葉に変換します。伝えたい内容を整理することを重視します。

具体的かつ段階的に指示する技術

AIに一度に複雑なことを頼むと、混乱したり、一部の機能が抜け落ちたりすることがあります。 コツは、**具体的**に、そして**段階的**に伝えることです。

テキストファイルの情報を表示するツールをNode.jsで作るケースを例に、良い指示の出し方を見ます。

悪い指示の例:

> *ファイルの文字数を数えるプログラムを書いて。*

これだと、言語は何か（Python? JavaScript?）、どうやってファイルを渡すか、エラー処理はどうするかが不明です。

良い指示の例:

以下では、必要な条件を箇条書きで整理して伝えています。

> ***役割**: あなたはプロのNode.jsエンジニアです。*
>
> ***目的**: コマンドライン引数で指定されたテキストファイルを読み込み、その統計情報を表示するCLIツールを作成してください。*
>
> ***要件**:*

> 1. *`process.argv` を使用して、第1引数からファイルパスを取得すること。*
> 2. *ファイルの内容を読み込み、「文字数」と「行数」をカウントすること。*
> 3. *結果をコンソールに見やすく出力すること。*
> 4. *指定されたファイルが存在しない場合など、エラーが発生した際は適切なエラーメッセージを表示すること。*
>
> ***言語**: JavaScript (Node.js)*

箇条書きを使って構造的に伝えると、AIは精度の高いコードを返してくれます。 裏を返せば、要件が曖昧なままだと出力も曖昧になります。

ステップバイステップ手法

さらに複雑なアプリを作る場合は、一度に最終形を目指さず、小さな機能ごとに作らせて組み合わせるステップバイステップの手法が有効です。

1. **段階1**: まずファイル名を受け取って表示するだけの最小限のコードを作らせる。
2. **段階2**: ファイルを読み込んで中身を表示する機能を追加させる。
3. **段階3**: 文字数を数えるロジックを追加させる。

段階的に作ることで、どこで問題が起きたかを特定しやすくなります。

いきなり最終形を作ろうとすると、エラーが出たときにどこが原因か分からなくなります。小さく作って、動くことを確認しながら積み上げていくのが、AI開発の鉄則です。

動作確認と修正

AIがコードを書いてくれても、それで終わりではありません。実際に動かすと、エラーが出たり、思った通りの動きをしなかったりすることは日常茶飯事です。 ここからが、人間とAIの対話の本番です。

エラーメッセージをそのままAIに投げる

コードを実行してエラー（赤い文字など）が出たら、慌てる必要はありません。 そのエラーメッセージをすべてコピーして、そのままAIに貼り付けてください。

> *「実行したら以下のエラーが出ました。修正してください。（エラーメッセージの貼り付け）」*

AIはエラーメッセージを読み解き、原因を特定して修正版を返してくれます。 プログラミング初心者にとってエラーメッセージの解読は難関ですが、AIがいれば心配いりません。

リファクタリング（コードを磨き上げる）

とりあえず動くものができたら、さらにコードを良くするためにAIに相談します。これを**リファクタリング**と呼びます。 機能（動き）は変えずに、内部の構造を整理して、読みやすく保守しやすいコードにする作業です。

- 「もっとコードを短く書けませんか？」
- 「変数名をもっと分かりやすく変更して」
- 「コメントを追記して解説して」

こうした依頼は、動いた後に磨き上げるための相談として効果的です。 対話を繰り返すことで、コードはどんどん洗練されていきます。

【生成例コード】文字数カウントツール

ここまでのプロセス（要件整理、指示出し、修正）を経て得られる、実際のコード例を見ます。 このコードは、コマンドライン引数で指定されたファイルの行数、単語数、文字数をカウントするツールです。

ファイル名: `word_count.js` として保存することを想定しています。

```javascript
// javascript
/**
 * ファイル統計ツール
```

```
 * 使い方: node word_count.js <ファイルパス>
 */

const fs = require('fs'); // ファイルシステムを扱うためのNode.js標準モジュール

// 1. コマンドライン引数の取得
// process.argv は [nodeのパス, スクリプトのパス, 引数1, 引数2...] という配列
const filePath = process.argv[2];

// 引数チェック: ファイルパスが指定されていない場合
if (!filePath) {
  console.error('エラー: ファイルパスを指定してください。');
  console.log('例: node word_count.js my_document.txt');
  process.exit(1); // エラー終了（終了コード1）
}

// 2. ファイルの読み込みと統計処理
try {
  // ファイルを同期的に読み込む（utf-8形式でテキストとして読み込む）
  const content = fs.readFileSync(filePath, 'utf-8');

  // 行数のカウント
  // 改行コード(\n)で分割した配列の長さで計算
  // ファイルが空の場合は0行とするための三項演算子を使用
  const lineCount = content.length === 0 ? 0 : content.split('\n').length;

  // 文字数のカウント
  const charCount = content.length;

  // 単語数のカウント（簡易版）
  // 空白文字（スペース、タブ、改行）で分割して、空文字を除外して数える
  // 正規表現 \s+ は「1つ以上の連続する空白」を意味します
  const wordCount = content.split(/\s+/).filter((word) => word.length > 0).length;

  // 3. 結果の出力
  console.log(`--- ファイル統計: ${filePath} ---`);
  console.log(`行数  : ${lineCount}`);
  console.log(`単語数: ${wordCount}`);
  console.log(`文字数: ${charCount}`);
} catch (error) {
  // 4. エラー処理
  // ファイルが存在しない、読み取り権限がないなどの場合ここに来る
  if (error.code === 'ENOENT') {
    console.error(`エラー: ファイル "${filePath}" が見つかりません。`);
  } else {
    console.error(`エラーが発生しました: ${error.message}`);
  }
  process.exit(1);
}
```

コードの解説です。

重要な4点に絞って見ていきます。

1. **引数の取得**: `process.argv[2]` で、ユーザーがコマンドの後ろに入力したファイル名を受け取っています。入力がなければ、使い方（例）を表示して終了します。

2. **try...catch 文**: ファイル操作はファイルが無いなどのエラーが起きやすいため、`try`（試す）ブロックの中に処理を書き、エラーが起きたら`catch`（捕捉する）ブロックに移動するようにしています。堅牢なツールを作るための重要なパターンです。
3. **統計ロジック**:
   - `split('\n')`: 文章を改行で区切ることで、行ごとのリストにします。
   - `split(/\s+/)`: 正規表現（`\s+`）を使って、スペースや改行などの空白で文章を区切り、単語のリストにします。
4. **エラーハンドリング**: `ENOENT`は"Error NO ENTry"（そのような項目はない＝ファイルが見つからない）という意味のエラーコードです。これを検知して、分かりやすい日本語メッセージを表示しています。

この4点を押さえると、CLIツールに必要な入力、処理、例外対応が一通り理解できます。

最初はエラー処理がなかったり、単語数のカウントが不正確だったりしたかもしれません。しかしAIとの対話を通じて、このような実用的な形に仕上げていくのです。

演習（プログラミング問題）（ファイル操作ツールの作成）

ここからは実際に手を動かしてCLIツールを作成する演習です。 ただし、すべてを自力で書く必要はありません。先ほど学んだバイブコーディングの手法を使い、AIと協力して（ペアプログラミングをして）作り上げてください。

今回のお題は、指定したディレクトリ（フォルダ）の中身を分析するツールです。 いきなり最終形を目指さず、2つの段階に分けて進めます。

ファイル一覧を表示するツールの作成

まずは基本機能として、指定された場所にあるファイルの名前を一覧表示するシンプルなツールを作ります。

狙い: 入力したディレクトリの中身を把握できるようにする。

AIへの指示（プロンプト）の実践

AIに送るべき指示を考えます。 何語で、何を、どのようにするかを明確にします。今回はJavaScript (Node.js) を使います。

以下のようなプロンプトをAI（ChatGPTやClaudeなど）に入力してみてください。

> ***プロンプト例:*** *Node.jsでCLIツールを作成したいです。 コマンドライン引数で「ディレクトリのパス」を受け取り、そのディレクトリ内に存在するファイルやフォルダの名前を一覧表示するスクリプトを書いてください。 エラー処理（ディレクトリが存在しない場合など）も入れてください。*

生成例コード

AIの出力は人によって異なりますが、以下はその一例です。 このコードを`ls_tool.js`という名前で保存して動作を確認します。

```
// javascript
/**
 * ファイル一覧表示ツール
 * ファイル名: ls_tool.js
 */
const fs = require('fs');
const path = require('path');

// 1. 引数からディレクトリのパスを取得
// 引数がなければ現在のディレクトリ（'.'）を対象にする
const targetDir = process.argv[2] || '.';

console.log(`--- ディレクトリ: ${targetDir} の中身 ---`);

try {
```

```
  // 2. ディレクトリの中身を読み込む
  // fs.readdirSync は、ディレクトリ内のファイル名のリスト（配列）を返します
  const items = fs.readdirSync(targetDir);

  // 3. 一つずつ表示する
  items.forEach((item) => {
    // パスを結合して完全なパスを作る
    const fullPath = path.join(targetDir, item);

    // ファイルかディレクトリかを判別するための情報を取得
    const stats = fs.statSync(fullPath);

    // ディレクトリなら [DIR]、ファイルなら [FILE] と先頭につける
    const prefix = stats.isDirectory() ? '[DIR ]' : '[FILE]';

    console.log(`${prefix} ${item}`);
  });
} catch (error) {
  console.error(`エラーが発生しました: ${error.message}`);
}
```

コードのポイント解説です。

特に重要な2点を取り上げます。

- `process.argv[2] || '.'`：ユーザーがパスを指定しなかった場合、自動的に現在の場所（`.`）を対象にする、よく使われるテクニック（デフォルト値の設定）です。
- `fs.statSync`：ファイルの詳細情報（ステータス）を取得する命令です。これを使うことで、それがフォルダなのかファイルなのかを区別できます。

この2点は入力の省略に対応するという機能と、種類を判定するという機能に直結します。

実行方法です。

ターミナルで以下のコマンドを実行し、現在のディレクトリの中身が表示されるか確認してください。

```
node ls_tool.js
```

表示結果は、あなたのフォルダーの中身によって変わるため、例と完全に一致しなくても問題ありません。

機能の追加（拡張子ごとの集計）

基本機能ができたら、もっと便利な機能を追加します。 ファイルが多すぎて整理できない、どの種類のファイルが多いのか知りたいというニーズに応えるため、拡張子（.txt や .jpg など）ごとの数を数える機能を追加します。

狙い: ファイルの種類の偏りを一目で把握できるようにする。

AIへの追加指示（プロンプト）

ここがバイブコーディングの真骨頂です。 AIとのチャットは続いている前提で、以下のように追加のお願いをします。

> ***追加プロンプト例:*** *ありがとうございます。今のコードに追加機能をお願いします。*
>
> *ファイル一覧を表示した後に、「拡張子ごとのファイル数」を集計して表示する機能を追加してください。（例: .txt: 5件, .js: 3件） ディレクトリ（フォルダ）は集計から除外してください。*

生成例コード（改良版）

AIは、既存のコードをベースに、集計ロジックを追加した新しいコードを提示してくれます。 以下はその一例です。

```javascript
// javascript
/**
 * ファイル分析ツール（集計機能付き）
 * ファイル名: ls_tool.js（更新版）
 */
const fs = require('fs');
const path = require('path');

const targetDir = process.argv[2] || '.';

console.log(`--- ディレクトリ: ${targetDir} の分析結果 ---`);

try {
  const items = fs.readdirSync(targetDir);

  // 集計用の「箱」を用意する（キー: 拡張子, 値: 個数）
  // 例: { '.txt': 2, '.js': 5 }
  const extensionCounts = {};
  let fileCount = 0;
  let dirCount = 0;

  items.forEach((item) => {
    const fullPath = path.join(targetDir, item);

    try {
      const stats = fs.statSync(fullPath);

      if (stats.isDirectory()) {
        console.log(`[DIR ] ${item}`);
        dirCount++;
      } else {
        console.log(`[FILE] ${item}`);
        fileCount++;

        // --- ここが集計ロジックの核心部分 ---

        // 拡張子を取得（例: "image.png" -> ".png"）
        // 拡張子がない場合は "(なし)" とする
        const ext = path.extname(item).toLowerCase() || '(なし)';

        // もしその拡張子がまだ箱になければ、0で初期化
        if (!extensionCounts[ext]) {
          extensionCounts[ext] = 0;
        }

        // カウントを +1 する
        extensionCounts[ext]++;
      }
    } catch (err) {
      // アクセス権限などで個別のファイルが見られない場合は無視して進む
      console.log(`[SKIP] ${item} (アクセス不可)`);
    }
  });

  console.log('\n--- 集計レポート ---');
```

```
  console.log(`合計: ディレクトリ ${dirCount}個 / ファイル ${fileCount}個`);
  console.log('拡張子別の内訳:');

  // 集計結果を表示
  // Object.keys で拡張子のリストを取り出し、forEachで回す
  Object.keys(extensionCounts).forEach((ext) => {
    console.log(`  ${ext}: ${extensionCounts[ext]} 件`);
  });
} catch (error) {
  console.error(`全体エラー: ${error.message}`);
}
```

このコードで学べる重要なテクニックです。

実用的な2つのテクニックに絞って整理します。

1. **連想配列（オブジェクト）を使った集計**:
   - `extensionCounts` という空のオブジェクトを用意し、ファイルが見つかるたびに `extensionCounts['.txt']` の数字を増やしていく手法です。
   - 投票箱を用意して、票が入るたびに正の字を書いていくイメージと同じです。プログラミングでは頻出のアルゴリズム（計算手順）です。
2. `path.extname`:
   - ファイル名から拡張子だけを切り出す便利な命令です。自分で最後のドットの位置を探して...といった処理を書く必要はありません。

この2つを理解すると、ファイル分析やレポート作成の応用がしやすくなります。

実行方法です。

再び実行します。今度は一覧の下にレポートが表示されるはずです。

```
node ls_tool.js
```

色々な種類のファイルが入っているフォルダがあれば、そのパスを指定して実行してみてください。

```
# 例: ダウンロードフォルダを分析してみる
node ls_tool.js ~/Downloads
```

表示される拡張子の種類や件数は環境によって異なるため、違いが出ても正常です。

演習で身につく開発サイクル

この演習を通じて、以下の開発フローを体験しました。

1. **ベースを作る**: まず最小限の機能（一覧表示）を作る。
2. **動作確認**: 想定通り動くか確かめる。
3. **拡張する**: 「もっとこうしたい」という要望をAIに伝えて、コードを進化させる。

この流れを繰り返すことで、無理なく機能を積み上げられます。 最初から完璧なコードを書こうとするのではなく、**雪だるま式に機能を大きくしていく**のがAI時代の賢い開発スタイルです。 この `ls_tool.js` は、あなたの手元にある専用のファイル分析ツールです。今後もファイルサイズの表示や並び順の変更といった改造に挑戦してみてください。

問題1 ファイル統計ツール

チャットで要件を聞き取り、それを満たすプログラムを作成してください。

1. 概要

テキストの統計情報（行数・文字数・単語数）を計算するCLIツールを作成してください。

2. 入力形式

標準入力から以下の形式でデータを受け取ります。

```
ファイル名
テキスト1行目
テキスト2行目
...
```

- 1行目はファイル名です（情報として与えられるだけで、処理には使用しません）。
- 2行目以降がテキストの内容です。
- ファイル名のみで、テキスト内容がない場合もあります。

3. 出力形式

以下の3行を出力してください。

```
Lines: X
Characters: Y
Words: Z
```

- `X` はテキスト内容の行数（ファイル名の行を除く）。
- `Y` はテキスト内容の総文字数（改行文字を除く）。
- `Z` はテキスト内容の総単語数（空白文字で区切られた単語の数）。

4. 処理の詳細

- ファイル名（1行目）はカウント対象外です。
- 行数は、テキスト内容の行数をカウントします。
- 文字数は、テキスト内容の各行の文字数を合計します（改行は含みません）。
- 単語数は、テキスト内容を空白文字で分割したときの要素数を合計します。

5. エッジケース

- テキスト内容が空（ファイル名のみ）の場合は、Lines: 0、Characters: 0、Words: 0 を出力します。

6. 入出力例

例1

入力:

```
test.txt
Hello World
This is a test
```

出力:

```
Lines: 2
Characters: 25
Words: 6
```

例2

入力:

```
empty.txt
```

出力:

```
Lines: 0
Characters: 0
Words: 0
```

問題2 CSVからJSONへの変換

チャットで要件を聞き取り、それを満たすプログラムを作成してください。

1. 概要

CSV形式のデータをJSON形式に変換するCLIツールを作成してください。

2. 入力形式

標準入力からCSVデータを受け取ります。

```
ヘッダー行（カンマ区切りのカラム名）
データ行1
データ行2
...
```

- 1行目はヘッダー行で、カンマ区切りのカラム名が含まれます。
- 2行目以降はデータ行で、ヘッダーに対応する値がカンマ区切りで含まれます。
- データ行は1行以上あります。

3. 出力形式

JSON配列を1行で出力してください（コンパクト形式、余分な空白なし）。

```
[{"key1":"value1","key2":"value2"},{"key1":"value3","key2":"value4"}]
```

- 各データ行をJSONオブジェクトに変換します。
- キーはヘッダー行のカラム名です。
- すべての値は文字列として扱います（数値に見える値も文字列のまま出力します）。
- JSON配列全体を1行で出力してください。

4. 処理の詳細

- ヘッダー行をカンマで分割してキーの配列を取得します。
- 各データ行をカンマで分割して値の配列を取得します。
- キーと値を対応させてJSONオブジェクトを構築します。
- すべてのオブジェクトを配列にまとめ、コンパクトなJSON文字列として出力します。

5. 入出力例

例1

入力:

```
name,age
Alice,30
Bob,25
```

出力:

```
[{"name":"Alice","age":"30"},{"name":"Bob","age":"25"}]
```

例2

入力:

```
id,value
1,hello
```

出力:

```
[{"id":"1","value":"hello"}]
```

問題3 テキストフィルタ

チャットで要件を聞き取り、それを満たすプログラムを作成してください。

1. 概要

キーワードを使ってテキストの行をフィルタリングするCLIツールを作成してください。

2. 入力形式

標準入力から以下の形式でデータを受け取ります。

```
キーワード
テキスト1行目
テキスト2行目
...
```

- 1行目はフィルタリングに使用するキーワードです。
- 2行目以降はフィルタリング対象のテキストです。

3. 出力形式

- キーワードを含む行を、元の順序で出力してください。
- マッチする行が1つもない場合は `NO_MATCH` と出力してください。

4. 処理の詳細

- キーワードの検索は**大文字・小文字を区別**します（case-sensitive）。
- 各行にキーワードが部分文字列として含まれているかどうかを判定します。
- マッチした行はそのまま（変更せずに）出力します。

5. 入出力例

例1

入力:

```
ERROR
[INFO] Start
[ERROR] Disk full
[INFO] Done
[ERROR] Timeout
```

出力:

```
[ERROR] Disk full
[ERROR] Timeout
```

例2

入力:

```
xyz
abc
def
ghi
```

出力:

```
NO_MATCH
```

第11章 CLIアプリケーションの自由制作

前節では、AIと協力してファイル分析ツールを作成しました。 作りたい機能を言葉にしてAIに伝えれば、実際に動くプログラムが出来上がる体験を通じて、開発の感覚が掴めたはずです。

本節はいよいよ自由制作です。 教科書通りの課題をこなすのではなく、あなた自身が欲しいと思うツールを、ゼロから企画して作り上げます。 単なるプログラミングの練習ではありません。身の回りにある問題を自分の力で解決する、エンジニアリングの第一歩です。

自由制作

自由に何でも作ってよいと言われると、逆に何を作ればいいか迷うこともあります。 素晴らしいソフトウェアを生み出すためのプロセスは、コードを書くずっと手前から始まっています。 プロのエンジニアも実践している課題発見から完成までのロードマップを体験していきます。

課題の発見

優れたツールの多くは、開発者自身の個人的な悩みから生まれています。 まずは日常生活や仕事、趣味の作業を振り返り、ネタを探してみます。

面倒くさいは発明の母

プログラミングの世界には、怠惰（Laziness）が美徳であるという言葉があります。 これは怠けるという意味ではなく、同じような作業を繰り返すのが嫌でたまらないから、それを自動化する工夫に全力を注ぐというポジティブな姿勢のことです。

日常の中で面倒だ、退屈だ、またこれかと感じた瞬間こそが、開発のチャンスです。

- 日々のルーチン: 毎日同じフォーマットで日報メールを書いている。
  - そこで、今日のやったことを箇条書きするだけで、挨拶文付きのメール本文を作ってくれるツールを作る。
- 情報のチェック: 毎朝、3つのニュースサイトと天気予報を見に行くのが手間だ。
  - そこで、まとめて取得して、重要な見出しだけをターミナルに表示するツールを作る。
- 趣味の管理: 集めた画像のファイル名がバラバラで整理がつかない。
  - そこで、撮影日やカメラの機種名ごとにフォルダを作って自動振り分けするツールを作る。

世界を変えるような大発明である必要はありません。自分の時間を1日5分だけ節約するツールで十分です。

身近な問題を解決する意義

自分自身が最初のユーザーになること（ドッグフーディングといいます）で、改善点がすぐに見つかります。 自分が使うために作るというモチベーションは、学習を継続する上で最強の燃料です。

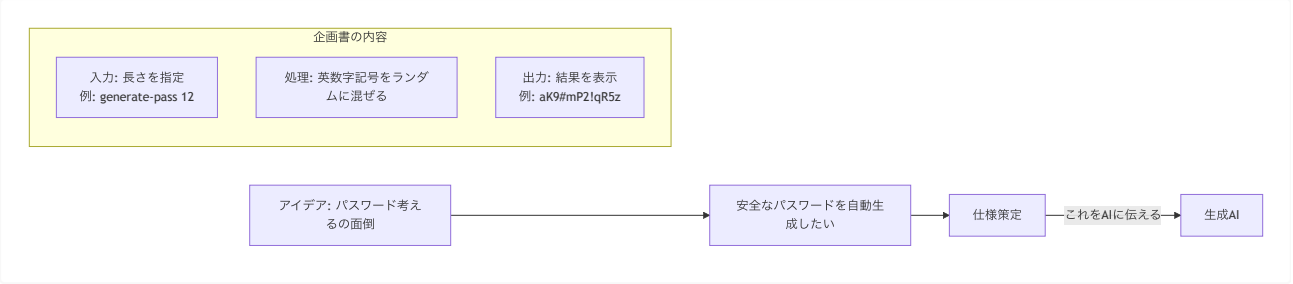
企画と設計

作りたいものが決まったら、すぐにコードを書き始めるのではなく、まず企画書を作ります。 堅苦しい書類ではありません。AIに指示を出すためのプロンプトの元ネタとなるメモ書きです。

アイデアを言葉にする（言語化）

以下の3点を明確にして書き出します。

1. ツール名（仮）: 何をするツールか一言で。
2. 概要: 誰の、どんな問題を解決するか。
3. 動作イメージ:
   - どんなコマンドを打つのか。
   - 何を入力するのか。
   - どんな結果が出るのか。



AIに相談してロードマップを描く

ここでもAIを活用します。いきなりコード書いてと頼む前に、相談から始めます。

> ***相談プロンプト例:*** *私はプログラミング初心者です。Node.jsを使って、コマンドラインで動く「強力なパスワード生成ツール」を作りたいと考えています。 長さを指定したり、記号を入れるか選べたりするようにしたいです。 初心者でも作れるでしょうか？ どのような手順で開発を進めればよいか、ロードマップを提案してください。*

AIは、スキルレベルに合わせて、まずはランダムな数字を出すところから始める、次に文字を混ぜる、最後に引数を受け取れるようにするといった段階的な計画を立ててくれます。 これにより、ゴールまでの道のりが見え、挫折しにくくなります。

実装と反復

計画ができたら、いよいよ実装（コーディング）です。 ここでも一度に完成させようとしないことが鉄則です。

小さなサイクルを回す

前節で学んだステップバイステップの手法で、AIと一緒に少しずつ機能を作っていきます。

1. Step 1: とりあえず動く骨組みを作る。
2. Step 2: 一つ機能を追加する。
3. Step 3: 動作確認して、バグがあれば修正する。
4. Step 2に戻る。

壁にぶつかったら仕様変更もアリ

開発途中で、この機能の実装は難しすぎてエラーが直らないという壁にぶつかることがあります。 そんなときは、思い切って仕様を変更したり、機能を削ったりするのも賢い判断です。

- 記号を含めるかどうかの選択機能が難しい場合
  - まずは記号なし版だけで完成させる。
- 結果をファイルに保存する機能がうまくいかない場合
  - 画面に表示するだけで十分とする。

プログラミング学習において最も重要なのは完璧なものを作ることではなく、完成させて動かす体験をすることです。 AIに「今の私のレベルでは難しいので、もっと簡単な方法に変えて」と正直に伝えれば、代替案を出してくれます。

【実践】パスワード生成ツールの開発例

実際にパスワード生成ツールを自由制作する過程をシミュレーションしたコードを見てみます。 これは、長さとオプション（数字を含めるか）を指定できるツールの完成例です。

ファイル名: `passgen.js`

```
// javascript
/**
 * シンプルなパスワード生成ツール
 * 使い方: node passgen.js <長さ> [オプション]
 * 例: node passgen.js 12 --no-number
 */
```

```
// パスワードに使用する文字の定義
const CHARS = {
    LOWER: 'abcdefghijklmnopqrstuvwxyz',
    UPPER: 'ABCDEFGHIJKLMNOPQRSTUVWXYZ',
    NUMBER: '0123456789',
    SYMBOL: '!@#$%^&*()_+-=[]{}|;:,.<>?`
};

// メイン処理
function main() {
    // 1. 引数の処理
    // デフォルトの長さは 8
    let length = 8;
    // 数字を含めるかどうか（デフォルトは true）
    let useNumber = true;

    const args = process.argv.slice(2);

    // 引数を解析する簡易ロジック
    args.forEach(arg => {
        if (!isNaN(arg)) {
            // 数字なら長さとして扱う
            length = parseInt(arg, 10);
        } else if (arg === '--no-number') {
            // オプション指定があれば数字を除外
            useNumber = false;
        }
    });

    // 長さの制限（安全のため）
    if (length < 4) {
        console.error("エラー: パスワードの長さは4文字以上にしてください。");
        process.exit(1);
    }

    // 2. 使用する文字セットの構築
    let availableChars = CHARS.LOWER + CHARS.UPPER;
    if (useNumber) {
        availableChars += CHARS.NUMBER;
    }
    // ここではシンプルにするため、記号は常に含めない設定にしていますが、
    // 改造して含められるようにするのも良い練習になります。

    // 3. パスワード生成
    let password = '';
    for (let i = 0; i < length; i++) {
        // ランダムな位置の文字を選ぶ
        const randomIndex = Math.floor(Math.random() * availableChars.length);
        password += availableChars[randomIndex];
    }

    // 4. 結果の出力
    console.log("--- パスワード生成完了 ---");
    console.log(`設定: 長さ=${length}, 数字=${useNumber ? 'あり' : 'なし'}`);
```

```
    console.log(`\n生成されたパスワード:\n${password}\n`);
}

// 実行
main();
```

このコードのポイント:

- `process.argv.slice(2)`：先頭の2つ（nodeとファイルパス）を除いた、純粋な引数部分だけを取り出しています。
- `Math.random()`：0以上1未満のランダムな小数を生成します。これを文字数と掛け合わせることで、ランダムな文字を選び出しています。
- 柔軟性: ユーザーが `node passgen.js 16` と打てば16文字になりますし、何も打たなければデフォルトの8文字になります。

このコードも、最初はランダムな文字を出すだけからスタートし、AIに「引数で長さを変えたい」「数字ありなしを選びたい」と追加注文して今の形になったものです。

ドキュメント作成

ツールが完成したら、最後にドキュメント（説明書）を作成します。 自分しか使わないから不要と思うかもしれませんが、1ヶ月後の自分は今の自分が書いたコードの使い方を忘れています。未来の自分のために書くものです。

READMEとは

ソフトウェアの世界では、プロジェクトのトップフォルダに `README.md` という名前の説明書ファイルを置くのが慣習です。 ここには以下の内容を書きます。

1. これは何か: ツールの名前と概要。
2. インストール方法: どうやって準備するか。
3. 使い方: どんなコマンドで動くか。
4. 仕様: どんな機能があるか。

AIにドキュメントを書かせる

ここでもAIが活躍します。完成したコードをAIに貼り付けて、次のように頼みます。

> ***プロンプト例:*** *このコードの `README.md` を作成してください。 初心者でも使い方がわかるように、実行例を含めてMarkdown形式で書いてください。*

AIはコードを解析し、適切な説明書を生成してくれます。 Markdownという記法で出力されるので、それをそのままファイルとして保存すれば完成です。

Markdownの基本

Markdownは、簡単な記号で文章の構造を表現する方法です。

- `#`：見出し（数が増えると小さくなる `##`, `###`）
- `-` または `*`：箇条書き
- ```` ``` ````：コードブロック

これを使って、読みやすいドキュメントを残す癖をつけると、開発者としての力が一段上がります。

こうして、課題発見、企画、AIとの実装、ドキュメント化という一連の流れを経験することで、単なる作業者から、自らの手で道具を生み出すクリエイターへと成長できます。 作ったツールがどんなに小さくても、それは世界に一つだけの、あなたの作品です。使い倒し、改良を続けてください。

制作例

自由制作の例を紹介します。 あくまで参考ですので、あなた自身の興味やニーズに合わせて、オリジナルのアイデアを考えてみてください。

おみくじ

日々の運勢を手軽に確認できるCLIおみくじツールです。コマンドを実行すると、今日の運勢がランダムに表示されます。

最初のプロンプト

```
Node.jsで動くCLIおみくじツールを作りたいです。
コマンドを実行すると、今日の運勢がランダムに表示されるようにしてください。
```

```
PS C:\Users\user\Desktop\おみくじ> node index.js

✨ 2026年1月30日 の運勢は…

🎉 【大吉】 🎉

🚀 最高の一日！新しい挑戦が吉。

PS C:\Users\user\Desktop\おみくじ> node index.js

✨ 2026年1月30日 の運勢は…

🌟 【小吉】 🌟

💝 小さな幸せに気づける日。

PS C:\Users\user\Desktop\おみくじ> node index.js

✨ 2026年1月30日 の運勢は…

✨ 【吉】 ✨

🌈 安定運。いつも通りが吉。
```

完成時のコード例

```
const fortunes = [
  { name: '大吉', emoji: '🎉' },
  { name: '中吉', emoji: '😊' },
  { name: '小吉', emoji: '🌟' },
  { name: '吉', emoji: '✨' },
  { name: '末吉', emoji: '👍' },
  { name: '凶', emoji: '⚠️' },
  { name: '大凶', emoji: '😢' },
];

const messages = {
  大吉: '🚀 最高の一日！新しい挑戦が吉。',
  中吉: '🌸 良い流れ。焦らず進めばOK。',
  小吉: '💝 小さな幸せに気づける日。',
  吉: '🌈 安定運。いつも通りが吉。',
  末吉: '📈 後半に運気上昇。',
  凶: '🛡️ 慎重に行動すると難を避けられる。',
  大凶: '🌙 無理は禁物。休息を優先して。',
};
```

```
const fortune = fortunes[Math.floor(Math.random() * fortunes.length)];
const message = messages[fortune.name] ?? '🙏 良い一日になりますように。';

const today = new Date();
const dateLabel = today.toLocaleDateString('ja-JP', {
  year: 'numeric',
  month: 'long',
  day: 'numeric',
});

console.log(`\n✨ ${dateLabel} の運勢は…\n`);
console.log(`${fortune.emoji} 【${fortune.name}】 ${fortune.emoji}`);
console.log(`\n${message}\n`);
```

成果発表と相互評価

自由制作で自分だけのCLIアプリケーションを完成させた、あるいは動く形にまで持っていけたことは、大きな一歩です。 ただし、開発は作って終わりではありません。自分の作品を他人に紹介し、フィードバック（意見や感想）をもらうことで、作品はより磨かれ、スキルもさらに向上します。

ここでは、成果を共有するためのデモンストレーションの方法と、お互いの作品を評価し合う相互評価のポイントを扱います。

デモンストレーション

デモンストレーションとは、自分の作ったソフトウェアが実際に動いている様子を実演して見せることです。 CLIアプリケーションの場合、派手な画面はありませんが、その分、何ができるのか、どれだけ便利なのかを言葉と実行結果で明確に伝える必要があります。

実際にツールが動いている様子を見せる

まずは、最も重要な動作を披露します。ターミナルを操作して、実際にコマンドを打ち込み、期待通りの結果が出る様子を見せます。

- 準備: 発表前に、正しく動作する入力データをいくつか用意しておきます。
- 実演: ユーザーがどのようにそのツールを使うのか、基本パターンの実行から始めます。
- エッジケース: あえてエラーになりそうな入力をしたときに、ツールが適切なエラーメッセージを出す様子を見せると、完成度の高さをアピールできます。

こだわった点や苦労した点を発表する

コードを見ただけでは伝わらない開発の裏側を語ることで、聞き手はより深くあなたの作品を理解できます。

1. こだわった点（推しポイント）:
    - エラーメッセージを分かりやすく日本語にした。
    - 実行中に進捗状況が表示されるように工夫した。
    - AIに何度も相談して、使いやすい引数の名前を考えた。
2. 苦労した点（学びのポイント）:
    - 最初はうまく動かなかったが、AIと相談して別の方法に変えて解決した。
    - 複雑な計算ロジックをAIに書いてもらったが、理解するのに時間がかかった。

これらの試行錯誤のプロセスこそが、同じ学習者にとって最も価値のある情報になります。

発表をスムーズに進めるための発表補助ツールを自作する例を見てみます。 発表時間（タイマー）を管理し、終わった時にメッセージを表示するシンプルなツールです。

ファイル名: `talk-timer.js`

```javascript
// javascript
/**
 * 発表用シンプルタイマー
 * 使い方: node talk-timer.js <分>
 * 例: node talk-timer.js 3 (3分間のタイマーを開始)
 */

function main() {
  // 1. 発表時間の取得（デフォルトは 3分）
  // process.argv[2] は、実行時にコマンドラインから渡された最初の引数を受け取ります。
  const minutes = parseInt(process.argv[2], 10) || 3;
  let secondsRemaining = minutes * 60;

  console.log(`\n🚀 ${minutes}分間のプレゼンテーションを開始します！`);
  console.log('------------------------------------------');

  // 2. カウントダウンの実行
  // setInterval は、指定したミリ秒ごとに処理を繰り返す関数です。
  const timer = setInterval(() => {
    secondsRemaining--;

    // 残り時間を 分:秒 の形式で計算します。
    const m = Math.floor(secondsRemaining / 60);
    const s = secondsRemaining % 60;

    // 1秒ごとに同じ行を上書きして表示するための特殊な文字（\r）を使用します。
    // これにより、画面が下に流れず、時計のように数字が更新されます。
    process.stdout.write(`\r残り時間: ${m}分${s < 10 ? '0' : ''}${s}秒 ... `);

    // 時間が来たら終了します。
    if (secondsRemaining <= 0) {
      clearInterval(timer); // 繰り返しを止める命令です。
      console.log('\n\n✅ 時間終了です！お疲れ様でした。');
    }
  }, 1000); // 1000ミリ秒（1秒）ごとに実行します。
}

main();
```

このコードの動作と意味:

- `setInterval`：JavaScriptに標準で備わっている機能で、一定時間ごとに特定の処理を繰り返し実行します。ここでは1秒（1000ミリ秒）ごとに残り時間を1減らし、画面の表示を更新し続けています。
- `process.stdout.write`：`console.log` と似ていますが、末尾に自動で改行を入れないという特徴があります。
- `\r`（エスケープシーケンス）：カーソルを現在の行の先頭に戻すという命令です。これを使って同じ行を何度も書き換えることで、文字が動いているようなアニメーション（カウントダウン）を実現しています。
- `clearInterval`：`setInterval` で設定したタイマーを解除・停止させるための命令です。これを実行しないと、プログラムは時間が終わっても終了せず、メモリを消費し続けてしまいます。

相互評価（フィードバック）

発表が終わったら、参加者同士で感想を伝え合います。これが相互評価です。 点数をつけるのが目的ではなく、お互いの気づきを共有して成長につなげることが目的です。

他人の作品を見て、良い点や改善点を伝え合う

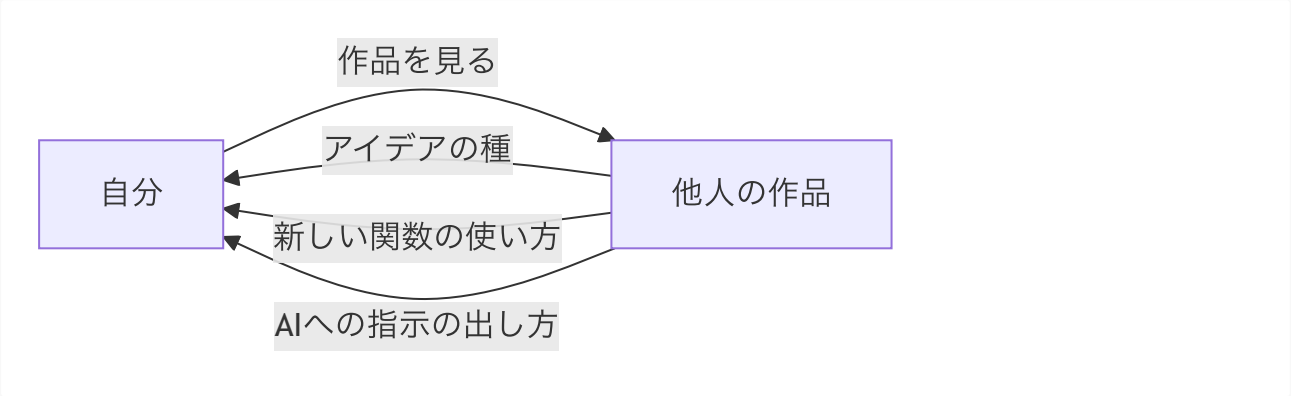
フィードバックを送る際は、以下の3つの視点を意識すると効果的です。

1. 共感（Good）：その悩み、私も持っていたのでこのツールはすごく便利だと思いました。
2. 発見（Interesting）：引数の指定方法が直感的で、使いやすいと感じました。
3. 提案（Advice）：もし結果をファイルに保存する機能があれば、もっと活用できそうだと思いました。

批判するのではなく、相手の努力を尊重した上でもっと良くなるためのアイデアを出すように心がけます。

自分では思いつかなかったアイデアや実装方法から学ぶ

他の人のコードや発表を見ることは、最高の学習教材です。 AIは多くの知識を持っていますが、何を作りたいかという人間の欲求や、どんな不便を感じているかという生身の感覚は、人間にしかありません。



この図が示すように、自分以外の視点を取り入れることで、開発の引き出しが増えていきます。 左側の自分が、右側の他人の作品に触れることで、自分一人では得られなかった知識や発想が矢印のように自分へと戻ってくる流れを表しています。

- 自分と同じAIツールを使っていても、まったく違うアプローチで解決している人がいるはずです。
- 自分が書いたコードと比べて、変数名の付け方やコメントの書き方が綺麗な作品を見つけたら、積極的に真似してください。

プログラミングの世界では、良いコードを読み、その手法を自分のものにすることを「盗む」と表現します。多くの作品に触れ、良い部分をどんどん吸収していきます。

CLIアプリケーション開発のセクションを通じて、AIを相棒にして自分の不便を自分で解決するというエンジニアの本質を体験しました。この経験を持って、次のステップへ進みます。

問題1 CLIアプリケーション自由制作

CLIアプリケーションを自由に制作してください。 ソースコードと`README.md`（使い方の説明）を提出してください。

制約

- Node.js（JavaScript）またはPythonで実装すること。
- `README.md`にツールの概要と使い方を記載すること。

第12章 ウェブアプリ（クライアントサイド）の開発方法

これまでは、黒い画面(ターミナル)で文字だけを使って操作するCLIアプリケーションを作ってきました。 ここからは、普段スマホやPCで見ているWebアプリ、つまりマウスでクリックしたり画像が表示されたりするグラフィカルなアプリケーションを扱います。 本章で扱うクライアントサイドとは、ユーザーの端末(ブラウザ)で動く処理のことです。サーバー側の処理とは区別して、画面表示やボタン操作といった手元で動く部分に焦点を当てます。

Webアプリ(クライアントサイド)を作るために必要な技術は、大きく分けて3つあります。 それが **HTML**、**CSS**、**JavaScript** です。

これらは家づくりに例えると分かりやすくなります。

- HTML: 家の骨組み(柱、床、壁)
- CSS: 家の内装(壁紙の色、家具の配置、装飾)
- JavaScript: 家の設備(スイッチを押すと電気がつく、蛇口をひねると水が出る仕掛け)

この3つが組み合わさって初めて、便利で快適なWebページが出来上がります。 骨組みがなければ家が立たず、内装がなければ居心地が悪く、設備がなければ便利に使えません。同じように、Webアプリも3つの役割が揃って初めて使える画面になります。 それぞれの役割と関係性を見ていきます。

事前準備（作業フォルダーと実行方法）

本章のコードを実行するための作業場所を準備します。

1. デスクトップに `ウェブアプリ` という名前の作業用フォルダーを作成します。
2. VS Codeでそのフォルダーを開いておきます。

詳細な手順は10章にありますので、そちらを参照してください。 この2ステップで、以降のコードを迷わず実行できる準備が整います。

以降のコード例は、保存してからブラウザで開き、リロードする流れで確認できます。 変更したらブラウザをリロードして反映します。

HTML、CSS、JavaScript

Webブラウザ(ChromeやSafariなど)は、サーバーから送られてきたこの3種類のコードを読み取り、瞬時に画面として組み立てて表示しています。

それぞれの役割を、実際のコードと図解で確認します。

HTML（構造）の役割

HTML(HyperText Markup Language)は、Webページの骨格を作るための言語です。 ここは見出し、ここは段落、ここにボタンを置く、というように文書の各部分が何であるかを定義します。

タグと要素

HTMLでは、 `<` と `>` で囲まれたタグを使って文章を囲みます。 たとえば、見出しを作りたいときは `<h1>` というタグを使います。

- `<h1>こんにちは</h1>` : こんにちはという文字を、一番大きな見出しとして扱います。
- `<p>これは文章です。</p>` : これは文章です。を一つの段落(Paragraph)として扱います。
- `<button>送信</button>` : 送信と書かれたボタンを作ります。

このように、タグで囲んで意味づけされたひとかたまりのことを要素(Element)と呼びます。 タグは見た目ではなく意味を伝えるための印であり、後からCSSやJavaScriptが正しく扱うための目印にもなります。

CSS（見た目）の役割

CSS(Cascading Style Sheets)は、HTMLで作られた骨組みに対して、色や大きさ、配置などのデザインを指定するための言語です。

HTMLだけでも文字やボタンは表示されますが、全てが左寄せの白黒で、味気ないものになります。CSSを使うことで、以下のようなことができます。

- 色を変える: 文字の色を赤にしたり、背景を青にしたりする。
- レイアウトを整える: 真ん中に寄せたり、横並びにしたりする。
- フォントを変える: 文字の大きさや種類(ゴシック体や明朝体)を変える。

これらはすべて見た目を整えるための指示であり、HTMLの構造を変えずに印象や使いやすさを大きく改善できます。

レスポンシブデザイン

CSSはレスポンシブデザインを実現するためにも重要です。 レスポンシブデザインとは、PC、タブレット、スマートフォンなど、画面の大きさが異なるデバイスでも見やすく表示されるように調整する仕組みのことです。 スマホで見るときだけ文字を小さくする、PCのときは2列でスマホのときは1列にする、といった指定もCSSで行います。

JavaScript（動き）の役割

JavaScriptは、Webページに動きや機能を与えるプログラミング言語です。 HTMLとCSSで作られたページは、美しいですが静止画のようなものです。そこにインタラクティブ(対話的)な動作を加えるのがJavaScriptです。

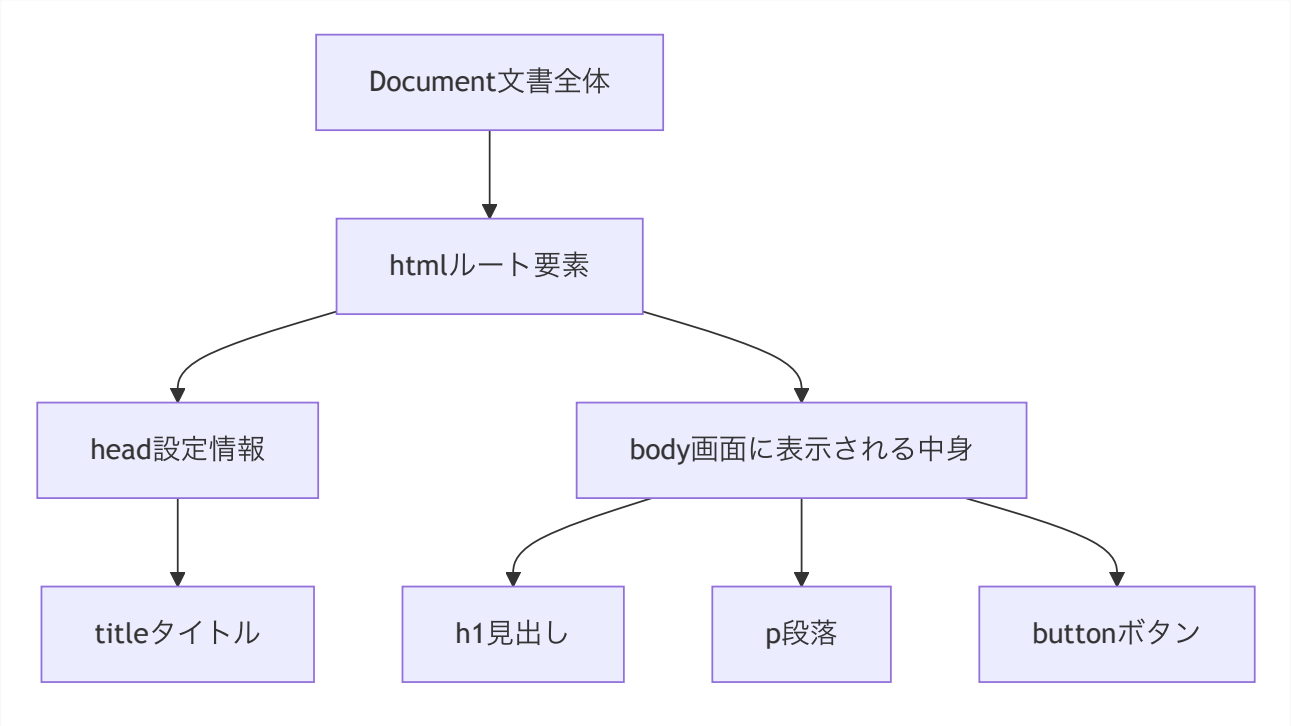
- 反応する: ボタンをクリックしたら、メッセージを表示する。
- 変更する: 条件によって、画面に表示されている画像や文字を書き換える。
- 通信する: サーバーから最新のニュースを取得して表示する。

JavaScriptはユーザーの操作や時間の経過をきっかけに、画面の内容を変える役割を担います。

DOM（ドキュメント・オブジェクト・モデル）

JavaScriptがHTMLを操作するために使う仕組みをDOM(Document Object Model)と呼びます。 これは、HTMLの文書構造をツリー状(木構造)のデータとして扱い、プログラムからアクセスできるようにしたものです。

以下の図を見てください。ブラウザはHTMLを読み込むと、内部でこのようなツリー構造を作ります。



図の解説です。 図に登場する主要な要素の役割を、3点に整理します。

- Document: Webページ全体を表す入り口です。
- Body: 普段画面で見ている部分です。
- ツリー構造: `body` の中に `h1` や `button` がぶら下がっています。JavaScriptはこのツリーを辿って、`body` の中にある `button` を探してクリックされたら `h1` の文字を変える、といった操作を行います。

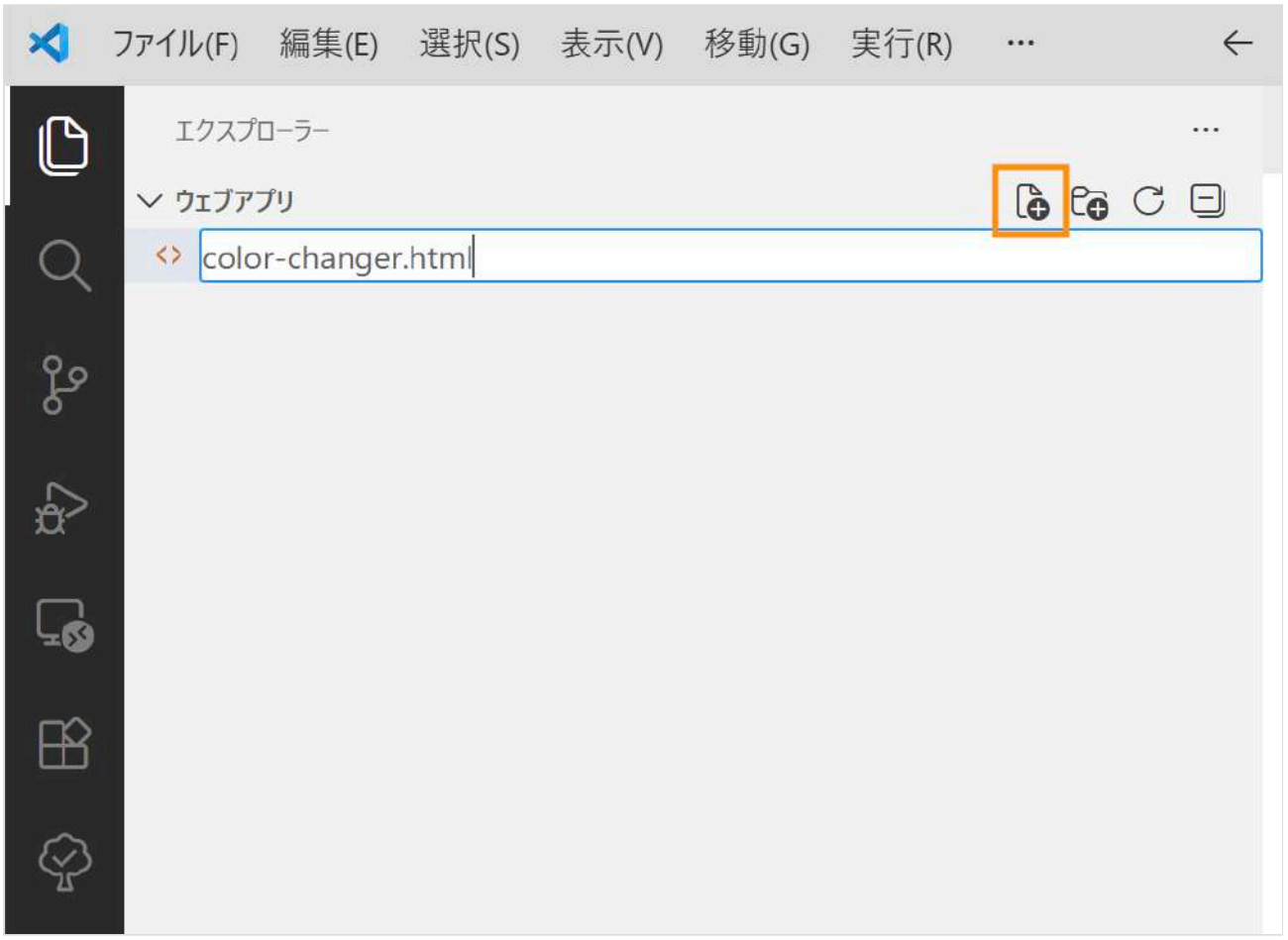
この図のように親子関係がはっきりした構造を理解すると、どの要素をどう操作すればよいかが見えてきます。

3要素の連携例：色が変わるミニアプリ

HTML、CSS、JavaScriptがどのように連携して動くのか、実際のコードで確認します。 以下は、ボタンをクリックすると背景色が変わり、メッセージも変化するシンプルなWebアプリのコードです。 通常はファイルを分けますが、ここでは分かりやすくするために1つのファイル(HTMLファイル)に全て記述しています。

デスクトップ上に作業場所を準備します。 作業場所の準備は、10章を参照してください。 フォルダー名は、ウェブアプリという名前にしておくと分かりやすいです。

VS Codeでウェブアプリフォルダーを開きます。 新規ファイル作成で `color-changer.html` という名前のファイルを作成し、以下のコードをコピー&ペーストしてください。



ファイル名: `color-changer.html`

```
<!DOCTYPE html>
<!-- html: 文書の言語を日本語に設定 -->
<html lang="ja">
  <head>
    <meta charset="UTF-8" />
    <title>カラーチェンジャー</title>
```

```
  <!-- CSS: 見た目の設定 -->
  <style>
    /* body全体の設定: 文字を中央揃えにし、フォントを見やすくする */
    body {
      font-family: sans-serif;
      text-align: center;
      padding-top: 50px;
      transition: background-color 0.5s; /* 背景色を0.5秒かけてじわっと変える */
    }

    /* h1見出しの設定 */
    h1 {
      color: #333;
      font-size: 24px;
    }

    /* ボタンのデザイン */
    button {
      padding: 10px 20px;
      font-size: 18px;
      background-color: #007bff; /* 青色 */
      color: white;
      border: none;
      border-radius: 5px; /* 角を丸くする */
      cursor: pointer; /* マウスを乗せたときに指の形にする */
    }

    /* ボタンにマウスが乗ったときの変化 */
    button:hover {
      background-color: #0056b3; /* 少し濃い青にする */
    }
  </style>
</head>
<body>
  <!-- HTML: 画面の構造 -->
  <h1 id="message">ボタンを押して色を変えよう！</h1>
  <p>今の気分に合わせた色に変更します。</p>
  <button id="change-btn">色を変える</button>

  <!-- JavaScript: 動きの設定 -->
  <script>
    // 1. HTMLの要素を取得する（DOM操作）
    // id="message" がついたh1タグを探してきて、変数 messageElement に入れる
    const messageElement = document.getElementById('message');
    // id="change-btn" がついたbuttonタグを探してきて、変数 buttonElement に入れる
    const buttonElement = document.getElementById('change-btn');

    // ランダムな色を作るためのリスト
    const colors = ['#ffcccc', '#ccffcc', '#ccccff', '#ffffcc', '#e0e0e0'];
    const messages = ['情熱の赤！', '癒やしの緑！', '冷静な青！', '元気な黄色！', 'シックなグレー！'];

    // 2. ボタンがクリックされたときの動作を定義する（イベントリスナー）
    buttonElement.addEventListener('click', () => {
      // 0から4までのランダムな整数を作る
      const randomIndex = Math.floor(Math.random() * colors.length);
```

```
        // 選ばれた色を背景色に設定する
        document.body.style.backgroundColor = colors[randomIndex];

        // 選ばれたメッセージに書き換える
        messageElement.textContent = messages[randomIndex];
      });
    </script>
  </body>
</html>
```

エクスプローラーで `color-changer.html` ファイルをダブルクリックして、Webブラウザで開いてみてください。

このコードのポイント: 構造・見た目・動きがどう分担されているかを、次の観点で確認します。

1. HTML部分 (`<body>` 内):
    - `h1` や `button` に `id="name"` という名札をつけています。これは後でJavaScriptから操作するときに、どの要素かを特定するために使います。
2. CSS部分 (`<style>` 内):
    - `body { ... }` や `button { ... }` のように、タグ名を指定してデザインを適用しています。
    - `transition` を使うことで、色がパッと切り替わるのではなく、ふわっと変わるアニメーション効果をつけています。
3. JavaScript部分 (`<script>` 内):
    - `document.getElementById(...)` : HTMLの中から、指定したIDを持つ要素(部品)を見つけてくる命令です。これがDOM操作の基本です。
    - `addEventListener('click', ...)` : クリックされたら実行する処理を待ち受ける設定です。
    - `document.body.style.backgroundColor = ...` : ページの背景色を直接プログラムで書き換えています。
    - `messageElement.textContent = ...` : 見出しの文字の中身を書き換えています。

この3つの役割が同じファイル内で協力している点に注目してください。構造・見た目・動きが役割分担されていると、変更が簡単になります。

このように、HTMLで構造を作り、CSSで飾り付け、JavaScriptで命を吹き込むことで、Webアプリは動いています。 次節からは、AIと協力してこれらのコードを実際に書いていきます。

バイブコーディングによるウェブアプリ（クライアントサイド）の開発

前節では、WebアプリがHTML、CSS、JavaScriptの3つの要素から成り立っていることを確認しました。これらをすべてゼロから手書きするのは大変な作業ですが、生成AIを活用するバイブコーディングなら、自然言語で指示を出すだけで素早く形にできます。

AIと協力してWebアプリを作る具体的なステップを解説します。

画面設計（モックアップ作成）

Webアプリ開発の第一歩は、どんな画面にしたいかを明確にして、AIにそのイメージを伝えることです。これをモックアップ(模型)作成と呼びます。

AIへの伝え方

AIは頭の中を見ることはできません。いい感じのサイトを作ってと伝えるだけでは、期待通りのものは出てきません。言葉で画面の構成要素を具体的に伝える必要があります。

たとえば、簡単なTo-Doリスト(やることリスト)を作りたい場合、以下のように指示(プロンプト)を出します。

> ***プロンプト例:*** *シンプルなTo-Doリストアプリの*HTML*と*CSS*を作ってください。 画面の中央に、以下の要素を縦に並べて配置してください。*

> 1. *タイトル：「My To-Do List」*
> 2. *テキスト入力欄：新しいタスクを入力する場所*
> 3. *「追加」ボタン：入力したタスクをリストに加えるボタン*
> 4. *タスク一覧：追加されたタスクが表示されるエリア 全体的に白と青を基調とした、清潔感のあるデザインにしてください。*

構成要素を順番に並べて伝えると、AIは配置と見た目を同時に理解しやすくなります。

手書きのラフスケッチ(線画)を描いて写真を撮り、それをAIにアップロードして、このラフ画通りのHTML/CSSコードを書いてと指示するのも有効な方法です。

UIフレームワークの活用提案

デザインを一から指定するのは大変ですが、世の中にはUIフレームワークと呼ばれる、プロが作ったデザイン部品セットが存在します。これを使うと、簡単に整ったデザインを作れます。

- Bootstrap(ブートストラップ): 歴史があり、標準的なデザインがすぐに作れるフレームワーク。
- Tailwind CSS(テイルウィンド・シーエスエス): 柔軟性が高く、現代的なサイト制作で人気のあるフレームワーク。

どちらも見た目を整えるための部品集なので、AIに指定するとデザイン品質を一段引き上げられます。

AIに対してBootstrapを使ってきれいにデザインしてと付け加えるだけで、プロ品質の画面コードが生成されます。AIはこれらのフレームワークの使い方を熟知しているので、積極的に活用してください。

イベント処理の実装

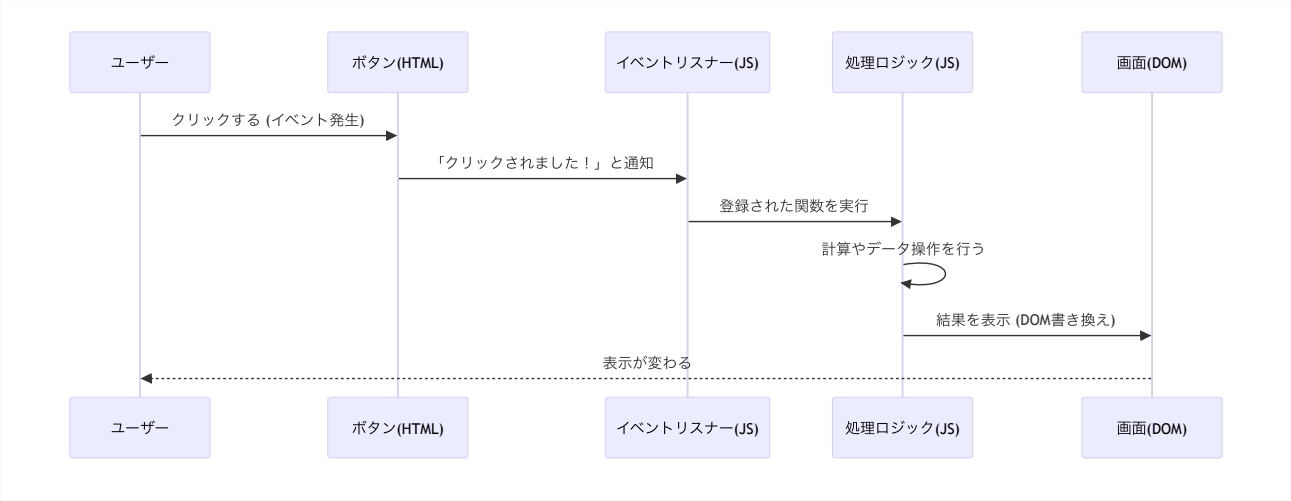
画面ができたら、次はJavaScriptを使って動きをつけます。これをイベント処理の実装と呼びます。

イベントリスナーとは

Webアプリにおけるイベントとは、ユーザーが行う操作のことです。ボタンをクリックした、キーボードを押した、マウスを動かした、などがイベントにあたります。

プログラム側で、このイベントが発生したらこの処理を実行すると待ち構えておく仕組みをイベントリスナー(Event Listener)と呼びます。聞き耳を立てている係を配置するイメージです。

以下の図は、ユーザーがボタンを押してから画面が変わるまでの流れを示しています。



図の解説です。 クリックから画面更新までの流れを、4つの段階に分けて確認します。

1. ユーザーがボタンをクリックします。
2. あらかじめ設定しておいたイベントリスナーがそれを感知します。
3. リスナーは、紐付けられたロジック(関数)を呼び出します。
4. ロジックが実行され、最終的に画面(DOM)の内容を書き換えることで、ユーザーに結果が伝わります。

この流れを理解しておくと、どのタイミングで何が動いているかが明確になり、バグ修正や機能追加が楽になります。

実践：いいねボタンの実装

AIに指示を出して、シンプルないいねカウンターを作ります。

> ***プロンプト例:*** *JavaScriptを追加して機能するようにしてください。 「いいね！」ボタンを押すと、画面上のカウント数が1ずつ増えるようにしてください。 また、10回以上押されたら「すごい人気！」というメッセージを表示してください。*

AIが生成するであろうコードの例を見てみます。 先ほどと同じ手順で `like-app.html` として保存して実行してみてください。

ファイル名: `like-app.html`

```
<!DOCTYPE html>
<html lang="ja">
  <head>
    <meta charset="UTF-8" />
    <title>いいねカウンター</title>
    <style>
      body {
        font-family: sans-serif;
        text-align: center;
        margin-top: 50px;
      }
      button {
        padding: 10px 20px;
        font-size: 20px;
        cursor: pointer;
        background-color: #ff4757;
        color: white;
        border: none;
        border-radius: 50px;
      }
      #count-display {
        font-size: 40px;
        font-weight: bold;
        margin: 20px;
      }
      #message {
        color: orange;
        font-size: 24px;
        font-weight: bold;
        height: 30px; /* メッセージがない時も高さを確保してレイアウト崩れを防ぐ */
      }
    </style>
  </head>
  <body>
    <h1>記事はいかがでしたか？</h1>

    <!-- カウントを表示する場所 -->
    <div id="count-display">0</div>

    <!-- いいねボタン -->
    <button id="like-button">❤ いいね！</button>

    <!-- メッセージを表示する場所 -->
```

```
    <div id="message"></div>

    <script>
      // --- 変数の準備 ---
      let count = 0; // 現在のいいね数を記憶する変数

      // --- HTML要素の取得 ---
      const countDisplay = document.getElementById('count-display');
      const likeButton = document.getElementById('like-button');
      const messageDisplay = document.getElementById('message');

      // --- イベント処理の実装 ---
      // ボタンに「クリックされたら実行する関数」を登録（イベントリスナー）
      likeButton.addEventListener('click', () => {
        // 1. カウントを増やす（ロジック）
        count = count + 1;

        // 2. 画面の数字を更新する（DOM操作）
        countDisplay.textContent = count;

        // 3. 条件分岐：10回以上ならメッセージを出す
        if (count >= 10) {
          messageDisplay.textContent = 'すごい人気！';
        }
      });
    </script>
  </body>
</html>
```

コードの解説です。 どの行が何をしているかを、特に重要な3点に絞って整理します。

- `let count = 0;`：データの状態を保持するための変数です。画面上だけでなく、プログラム内部でも数字を記憶しておく必要があります。
- `addEventListener('click', ...)`：これがイベントリスナーです。クリックされるたびに `{ ... }` の中の処理が実行されます。
- `if (count >= 10)`：条件分岐を使って、特定の状況(10回押された)でのみ動作を変えています。

この3点がそろうと、状態を持ち、クリックで更新し、条件で分岐するという基本構造が一通り理解できます。

動作確認とブラウザの活用

コードを書いたら(AIに生成させたら)、必ず動作確認を行います。一発で完璧に動くとは限りません。ここで重要なのがブラウザの開発者ツールです。

実際の動作確認

作成したHTMLファイルをブラウザで開き、実際にボタンを押してみます。

- ボタンのデザインは崩れていないか
- クリックして数字が増えるか
- 10回押したときにメッセージが出るか

この3点を押さえると、見た目・動作・条件分岐の最低限を確認でき、基本的な品質チェックになります。

もしボタンを押しても何も起きない場合、プログラムにエラーがある可能性があります。Webページはエラーがあっても画面上には何も表示せず、ただ黙って動かなくなることがよくあります。

開発者ツールの活用

そんな時に使うのが、ブラウザに標準搭載されている開発者ツール(Developer Tools)です。 画面上の何もないところで右クリックし、検証(Inspect)を選ぶか、キーボードの `F12` キーを押すと開きます。

特に重要なのがConsole(コンソール)タブです。

1. Consoleタブを開く: ここには、JavaScriptのエラーメッセージが赤文字で表示されます。
2. エラーを確認する: たとえば `Uncaught ReferenceError: cont is not defined` のようなメッセージが出ていれば、`cont` という変数は定義されていない(スペルミスかもしれない)と教えてくれています。
3. AIに修正を依頼する: エラーの内容がわからなくても問題ありません。その赤いエラーメッセージをコピーして、そのままAIに貼り付けます。

この3ステップで原因の特定から修正の依頼まで一気に進められるため、初心者でも確実に問題解決ができます。

> ***AIへの修正依頼プロンプト例:*** *コードを実行したらブラウザのコンソールに以下のエラーが出ました。修正してください。 エラー内容:* `Uncaught TypeError: Cannot set properties of null (setting 'textContent')`

このように、実装し、動作確認し、開発者ツールでエラーを発見し、AIで修正するというサイクルを回すことで、詳しくない言語でもアプリを完成させることができます。これが現代的な開発スタイルです。

演習（プログラミング問題）

ここまでの学習で、WebアプリがHTML(構造)、CSS(見た目)、JavaScript(動き)の組み合わせでできていること、そしてそれらをAIに生成させる方法を確認しました。 この演習では、実際に手を動かして(といってもコードを書くのは主にAIですが)、2つのアプリケーションを作成します。

AIへの指示(プロンプト)を工夫して、自分の思い通りのアプリを完成させてください。

カウンターアプリの作成

前節のいいねボタンを発展させて、数を数えるためのカウンターアプリを作ります。 野鳥の数を数えたり、交通量調査に使ったりする数取器(カチカチする機械)をイメージしてください。

目標となる機能

単に数を増やすだけでなく、以下の機能を持たせます。

1. 増やすボタン(+): カウントを1増やす。
2. 減らすボタン(-): カウントを1減らす(ただし0より小さくならないようにする)。
3. リセットボタン: カウントを0に戻す。

この3つが揃うと、日常の数を数える作業をそのままWebアプリとして再現できます。

AIへのプロンプト例

この要件をAIに伝えてコードを生成させます。以下のようなプロンプトを使ってみてください。

> ***プロンプト:*** *HTML、CSS、JavaScriptを使って、シンプルなカウンターアプリを作ってください。 画面中央に大きく数字を表示し、その下に以下の3つのボタンを横並びで配置してください。*
>
> 1. *「+」ボタン：数字を1増やす*
> 2. *「-」ボタン：数字を1減らす（ただし、マイナスにはならないように制御する）*
> 3. *「Reset」ボタン：数字を0に戻す*
>
> *デザインは、スマホで操作しやすいようにボタンを大きめに、カラフルにしてください。 コードは1つのHTMLファイルにまとめてください。*

操作条件を具体的に書くと、AIが機能とUIを同時に満たすコードを出しやすくなります。

コード例と解説

AIが生成するコードの一例です。 これを `counter.html` として保存し、ブラウザで動作を確認してください。

ファイル名: `counter.html`

```
<!DOCTYPE html>
<html lang="ja">
  <head>
    <meta charset="UTF-8" />
    <title>カウンターアプリ</title>
    <style>
      /* 全体のレイアウト設定 */
      body {
        font-family: sans-serif;
        text-align: center; /* 中央揃え */
        background-color: #f0f4f8;
        margin-top: 50px;
      }

      /* 数字の表示エリア */
      #count-display {
        font-size: 80px; /* とても大きく */
        font-weight: bold;
        color: #333;
        margin-bottom: 30px;
      }

      /* ボタン共通のデザイン */
      button {
        font-size: 24px;
        padding: 15px 30px;
        margin: 10px;
        border: none;
        border-radius: 10px;
        cursor: pointer;
        color: white;
        transition: transform 0.1s; /* 押したときに少し動くアニメーション用 */
      }

      /* ボタンを押したときのへこむ演出 */
      button:active {
        transform: scale(0.95);
      }

      /* 各ボタンの色分け */
      #btn-plus {
        background-color: #2ecc71;
      } /* 緑 */
      #btn-minus {
        background-color: #e74c3c;
      } /* 赤 */
      #btn-reset {
        background-color: #95a5a6;
      } /* グレー */
    </style>
  </head>
  <body>
    <h1>カウンター</h1>
```

```
    <!-- ここに数字が表示されます -->
    <div id="count-display">0</div>

    <!-- 操作ボタン -->
    <div>
      <button id="btn-minus">-</button>
      <button id="btn-reset">Reset</button>
      <button id="btn-plus">+</button>
    </div>

    <script>
      // --- 変数の準備 ---
      let count = 0; // 現在の数値を記憶する変数

      // --- HTML要素の取得 ---
      const display = document.getElementById('count-display');
      const btnPlus = document.getElementById('btn-plus');
      const btnMinus = document.getElementById('btn-minus');
      const btnReset = document.getElementById('btn-reset');

      // --- 画面の数字を更新する関数 ---
      // 何度も使う処理なので、関数（部品）にしておくと便利です
      function updateDisplay() {
        display.textContent = count;
      }

      // --- イベントリスナーの設定 ---

      // 1. プラスボタンが押されたとき
      btnPlus.addEventListener('click', () => {
        count = count + 1; // 1増やす
        updateDisplay(); // 画面更新
      });

      // 2. マイナスボタンが押されたとき
      btnMinus.addEventListener('click', () => {
        if (count > 0) {
          // 現在が0より大きいときだけ実行（0以下にしない）
          count = count - 1; // 1減らす
          updateDisplay(); // 画面更新
        }
      });

      // 3. リセットボタンが押されたとき
      btnReset.addEventListener('click', () => {
        count = 0; // 0に戻す
        updateDisplay(); // 画面更新
      });
    </script>
  </body>
</html>
```

学習のポイント: カウンターアプリで特に重要な設計の考え方を、次の2点にまとめます。

- 条件分岐 (`if`): マイナスボタンの処理で `if (count > 0)` とすることで、0のときは減らさない(何もしない)という制御を実現しています。これがバグのないアプリを作るための第一歩です。
- 関数の再利用: `updateDisplay()` という関数を作ることで、数字を表示する処理を3回書かずに済ませています。

この2点を意識すると、簡単なアプリでもミスが起きにくい設計を学べます。

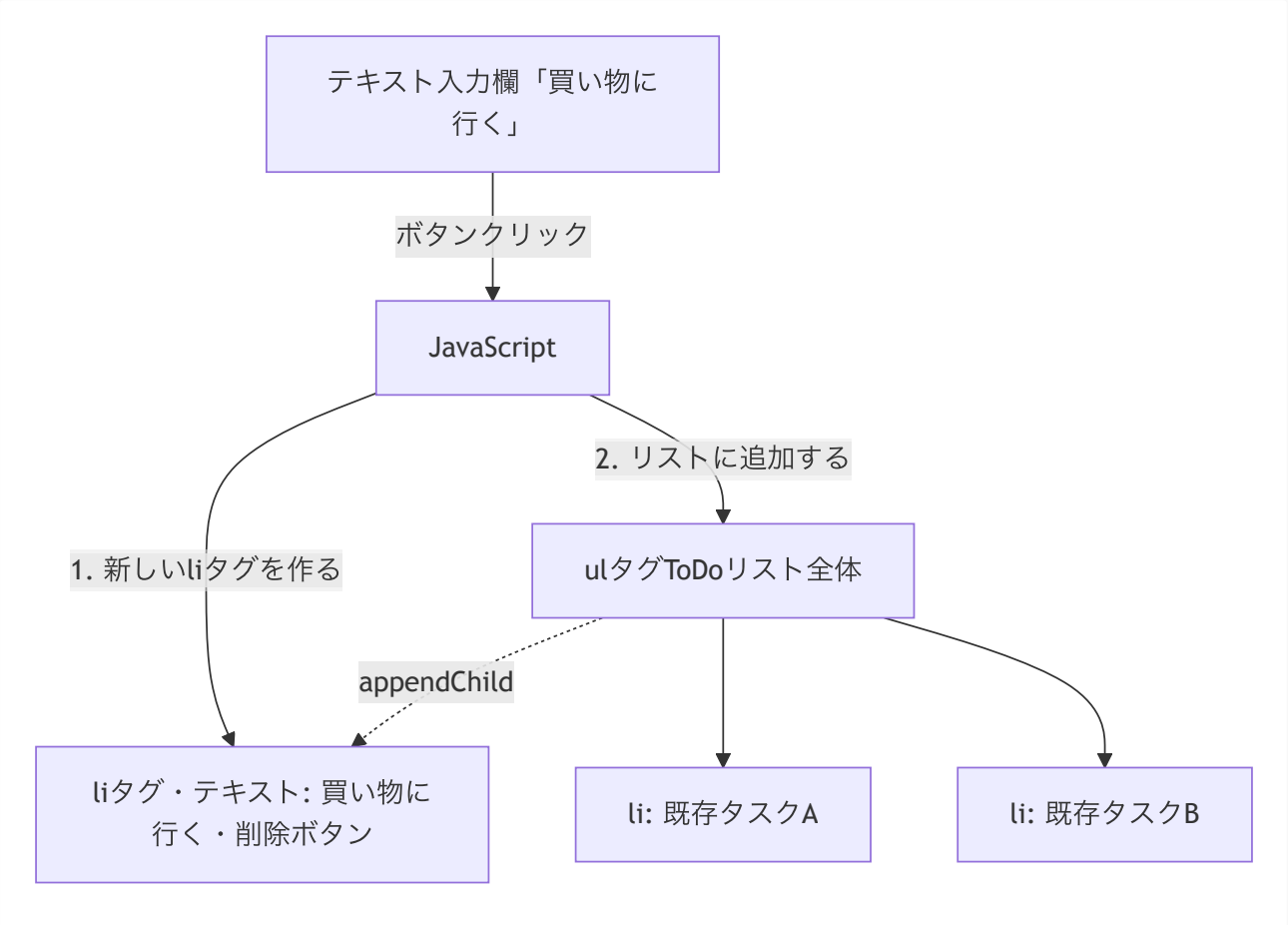
ToDoリストの作成

次は少し難易度を上げて、ToDoリスト(やることリスト)を作成します。 カウンターアプリとの最大の違いは、画面上の要素(HTMLタグ)が増えたり減ったりすることです。これをDOMの動的な操作と呼びます。

Webアプリの基本機能であるデータの追加(Create)と削除(Delete)を扱います。

動作のイメージ

仕組みを図で理解します。JavaScriptがHTMLをどのように書き換えるかを見てください。



処理の流れ: 追加と削除の処理は、次の4段階で進みます。

1. ユーザーが文字を入力して追加ボタンを押す。
2. JavaScriptが、入力された文字を使って新しい `<li>` タグ(リストの項目)を作成する。
3. 作成したタグを、親となる `<ul>` タグ(リスト全体)の最後尾にくっつける(appendChild)。
4. 削除ボタンが押されたら、その `<li>` タグを取り除く(remove)。

この4段階の流れは、掲示板やチャットなど項目が増減する画面に共通する基本動作です。

AIへのプロンプト例

複雑な動きに見えますが、AIに正しく伝えればすぐに作ってくれます。

> ***プロンプト:*** *ToDoリストアプリを作ってください。 1つのHTMLファイルで、以下の機能を持たせてください。*

> 1. *上部に「タスク入力欄」と「追加ボタン」がある。*
> 2. *タスクを入力して追加ボタンを押すと、下部のリストに新しいタスクが追加される。*
> 3. *追加された各タスクの右側には「削除」ボタンがあり、押すとそのタスクが消える。*
> 4. *入力欄が空の状態で追加ボタンを押しても、何も起きないようにする。*
>
> *デザインはシンプルで見やすく、リストが見やすいように枠線などで区切ってください。*

機能要件とデザイン要件を分けて書くと、AIが必要な処理と見た目を取り違えにくくなります。

コード例と解説

生成されたコード(例)を `todo.html` として保存して実行してみてください。

ファイル名: `todo.html`

```
<!DOCTYPE html>
<html lang="ja">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>My To-Do List</title>
    <style>
      * {
        margin: 0;
        padding: 0;
        box-sizing: border-box;
      }

      body {
        font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
        background: linear-gradient(135deg, #e3f2fd 0%, #bbdefb 100%);
        min-height: 100vh;
        display: flex;
        justify-content: center;
        align-items: center;
        padding: 20px;
      }

      .container {
        background-color: white;
        border-radius: 10px;
        box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
        width: 100%;
        max-width: 500px;
        padding: 40px;
      }

      h1 {
        color: #1976d2;
        text-align: center;
        margin-bottom: 30px;
        font-size: 32px;
        font-weight: 600;
      }

      .input-section {
        display: flex;
```

```
  gap: 10px;
  margin-bottom: 30px;
}

input[type='text'] {
  flex: 1;
  padding: 12px 15px;
  border: 2px solid #e0e0e0;
  border-radius: 5px;
  font-size: 16px;
  transition: border-color 0.3s;
}

input[type='text']:focus {
  outline: none;
  border-color: #1976d2;
}

button {
  padding: 12px 30px;
  background-color: #1976d2;
  color: white;
  border: none;
  border-radius: 5px;
  font-size: 16px;
  font-weight: 600;
  cursor: pointer;
  transition: background-color 0.3s;
}

button:hover {
  background-color: #1565c0;
}

button:active {
  background-color: #0d47a1;
}

.task-list {
  list-style: none;
}

.task-item {
  background-color: #f5f5f5;
  padding: 15px;
  margin-bottom: 10px;
  border-radius: 5px;
  border-left: 4px solid #1976d2;
  display: flex;
  justify-content: space-between;
  align-items: center;
  transition: background-color 0.3s;
}

.task-item:hover {
```

```
      background-color: #eeeeee;
    }

    .task-text {
      color: #333333;
      font-size: 16px;
      flex: 1;
    }

    .delete-btn {
      padding: 6px 12px;
      background-color: #e53935;
      color: white;
      border: none;
      border-radius: 3px;
      font-size: 14px;
      cursor: pointer;
      transition: background-color 0.3s;
    }

    .delete-btn:hover {
      background-color: #c62828;
    }

    .empty-message {
      text-align: center;
      color: #999999;
      padding: 20px;
      font-size: 14px;
    }
  </style>
</head>
<body>
  <div class="container">
    <h1>My To-Do List</h1>
    <div class="input-section">
      <input
        type="text"
        id="taskInput"
        placeholder="新しいタスクを入力してください..."
        onkeypress="handleKeyPress(event)"
      />
      <button onclick="addTask()">追加</button>
    </div>
    <ul id="taskList" class="task-list">
      <li class="empty-message">タスクはまだありません</li>
    </ul>
  </div>
  <script>
    const taskInput = document.getElementById('taskInput');
    const taskList = document.getElementById('taskList');

    function addTask() {
      const taskText = taskInput.value.trim();
```

```
        if (taskText === '') {
          alert('タスクを入力してください');
          return;
        }

        // 空のメッセージを削除
        const emptyMessage = taskList.querySelector('.empty-message');
        if (emptyMessage) {
          emptyMessage.remove();
        }

        // 新しいタスク要素を作成
        const taskItem = document.createElement('li');
        taskItem.className = 'task-item';
        taskItem.innerHTML = `
                <span class="task-text">${escapeHtml(taskText)}</span>
                <button class="delete-btn" onclick="deleteTask(this)">削除</button>
            `;

        taskList.appendChild(taskItem);
        taskInput.value = '';
        taskInput.focus();
      }

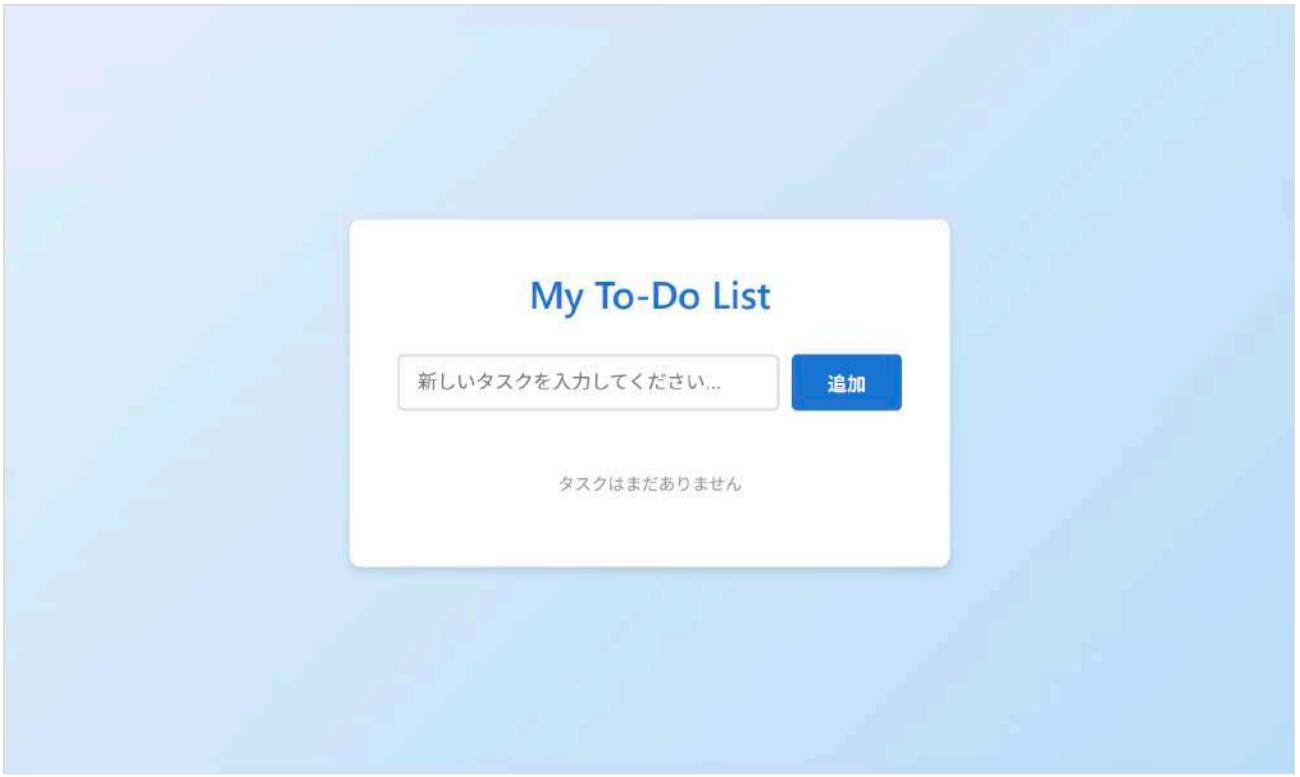
      function deleteTask(button) {
        button.parentElement.remove();

        // リストが空になったら、メッセージを表示
        if (taskList.children.length === 0) {
          const emptyMessage = document.createElement('li');
          emptyMessage.className = 'empty-message';
          emptyMessage.textContent = 'タスクはまだありません';
          taskList.appendChild(emptyMessage);
        }
      }

      function handleKeyPress(event) {
        if (event.key === 'Enter') {
          addTask();
        }
      }

      function escapeHtml(text) {
        const map = {
          '&': '&amp;',
          '<': '&lt;',
          '>': '&gt;',
          '"': '&quot;',
          "'": '&#039;',
        };
        return text.replace(/[&<>"']/g, (m) => map[m]);
      }
    </script>
  </body>
</html>
```

動作例



バイブコーディングの実践（対話で改善する）

ここからは、AIとの会話を2ターンで進めてToDoリストを改善していきます。

ターン1：削除時の確認を追加する

狙い: 誤操作を防ぎ、アプリの安全性を高める。

削除時に確認ダイアログを表示して、誤って削除しないようにする機能を追加します。

> ***追加プロンプト:*** *削除ボタンが押されたときに、確認ダイアログを表示して、「本当に削除しますか？」と尋ねるようにしてください。*

AIが作成を完了したら、修正内容を反映するためにブラウザをリロードします。

確認ダイアログ追加後の生成例コード

以下は、確認ダイアログを追加した場合の一例です。AIの出力は人によって異なるため、あくまで参考として読み、必要に応じてコピーして動作確認してください。

```
<!DOCTYPE html>
<html lang="ja">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>My To-Do List</title>
    <style>
      * {
        margin: 0;
        padding: 0;
        box-sizing: border-box;
      }
```

```css
body {
  font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
  background: linear-gradient(135deg, #e3f2fd 0%, #bbdefb 100%);
  min-height: 100vh;
  display: flex;
  justify-content: center;
  align-items: center;
  padding: 20px;
}

.container {
  background-color: white;
  border-radius: 10px;
  box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
  width: 100%;
  max-width: 500px;
  padding: 40px;
}

h1 {
  color: #1976d2;
  text-align: center;
  margin-bottom: 30px;
  font-size: 32px;
  font-weight: 600;
}

.input-section {
  display: flex;
  gap: 10px;
  margin-bottom: 30px;
}

input[type='text'] {
  flex: 1;
  padding: 12px 15px;
  border: 2px solid #e0e0e0;
  border-radius: 5px;
  font-size: 16px;
  transition: border-color 0.3s;
}

input[type='text']:focus {
  outline: none;
  border-color: #1976d2;
}

button {
  padding: 12px 30px;
  background-color: #1976d2;
  color: white;
  border: none;
  border-radius: 5px;
  font-size: 16px;
```

```
  font-weight: 600;
  cursor: pointer;
  transition: background-color 0.3s;
}

button:hover {
  background-color: #1565c0;
}

button:active {
  background-color: #0d47a1;
}

.task-list {
  list-style: none;
}

.task-item {
  background-color: #f5f5f5;
  padding: 15px;
  margin-bottom: 10px;
  border-radius: 5px;
  border-left: 4px solid #1976d2;
  display: flex;
  justify-content: space-between;
  align-items: center;
  transition: background-color 0.3s;
}

.task-item:hover {
  background-color: #eeeeee;
}

.task-text {
  color: #333333;
  font-size: 16px;
  flex: 1;
}

.delete-btn {
  padding: 6px 12px;
  background-color: #e53935;
  color: white;
  border: none;
  border-radius: 3px;
  font-size: 14px;
  cursor: pointer;
  transition: background-color 0.3s;
}

.delete-btn:hover {
  background-color: #c62828;
}

.empty-message {
```

```
      text-align: center;
      color: #999999;
      padding: 20px;
      font-size: 14px;
    }
  </style>
</head>
<body>
  <div class="container">
    <h1>My To-Do List</h1>
    <div class="input-section">
      <input
        type="text"
        id="taskInput"
        placeholder="新しいタスクを入力してください..."
        onkeypress="handleKeyPress(event)"
      />
      <button onclick="addTask()">追加</button>
    </div>
    <ul id="taskList" class="task-list">
      <li class="empty-message">タスクはまだありません</li>
    </ul>
  </div>
  <script>
    const taskInput = document.getElementById('taskInput');
    const taskList = document.getElementById('taskList');

    function addTask() {
      const taskText = taskInput.value.trim();

      if (taskText === '') {
        alert('タスクを入力してください');
        return;
      }

      // 空のメッセージを削除
      const emptyMessage = taskList.querySelector('.empty-message');
      if (emptyMessage) {
        emptyMessage.remove();
      }

      // 新しいタスク要素を作成
      const taskItem = document.createElement('li');
      taskItem.className = 'task-item';
      taskItem.innerHTML = `
              <span class="task-text">${escapeHtml(taskText)}</span>
              <button class="delete-btn" onclick="deleteTask(this)">削除</button>
          `;

      taskList.appendChild(taskItem);
      taskInput.value = '';
      taskInput.focus();
    }

    function deleteTask(button) {
```

```
        if (!confirm('本当に削除しますか？')) {
          return;
        }

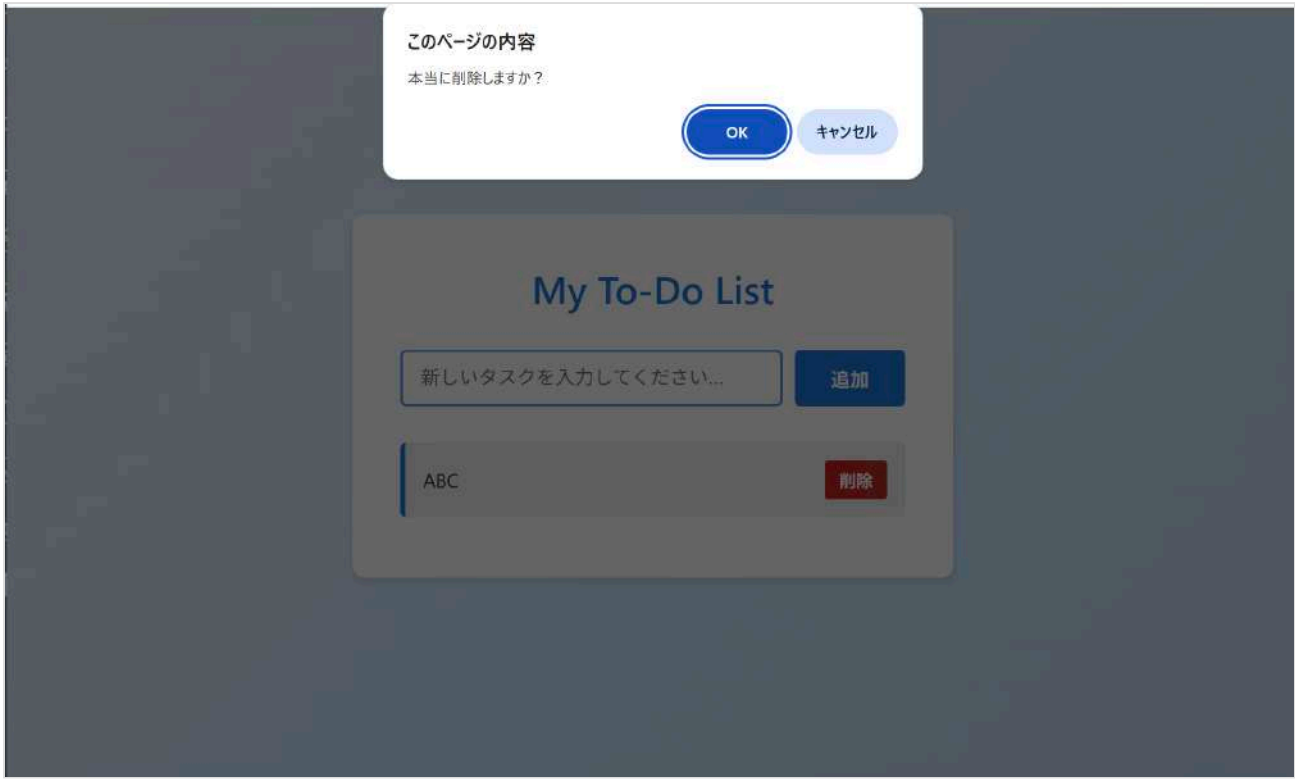
        button.parentElement.remove();

        // リストが空になったら、メッセージを表示
        if (taskList.children.length === 0) {
          const emptyMessage = document.createElement('li');
          emptyMessage.className = 'empty-message';
          emptyMessage.textContent = 'タスクはまだありません';
          taskList.appendChild(emptyMessage);
        }
      }

      function handleKeyPress(event) {
        if (event.key === 'Enter') {
          addTask();
        }
      }

      function escapeHtml(text) {
        const map = {
          '&': '&amp;',
          '<': '&lt;',
          '>': '&gt;',
          '"': '&quot;',
          "'": '&#039;',
        };
        return text.replace(/[&<>"']/g, (m) => map[m]);
      }
    </script>
  </body>
</html>
```

確認ダイアログの動作例



※確認ダイアログの見た目はブラウザやOSによって異なります。

ターン2：リロード後もタスクが残るようにする

狙い: ページを閉じてもデータが消えないアプリに進化させる。

現在はリロードするとタスクが消えてしまいますが、データをローカルストレージに保存することで、ブラウザを閉じてもデータを保持できます。 ローカルストレージとは、ブラウザ内に文字列としてデータを保存できる仕組みで、簡単な永続化に向いています。 AIにローカルストレージに保存するコードを追加してと指示するだけで実装できます。

> ***追加プロンプト:*** *タスクを追加・削除したときに、ブラウザのローカルストレージに保存するようにしてください。 ページをリロードしても、タスクが消えないようにしてください。*

AIが作成を完了したら、修正内容を反映するためにブラウザをリロードします。 タスクを追加・削除した後、ページをリロードしてもタスクが残っていることを確認してください。

これで追加・削除・復元の3機能が連動し、リロードしても状態が維持されます。

保存対応後の生成例コード

以下は、ローカルストレージ対応まで反映した場合の一例です。出力は毎回同じになるとは限らないため、参考例として確認し、必要ならコピーして動作確認してください。

```
<!DOCTYPE html>
<html lang="ja">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>My To-Do List</title>
    <style>
      * {
        margin: 0;
        padding: 0;
        box-sizing: border-box;
```

```
}

body {
  font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
  background: linear-gradient(135deg, #e3f2fd 0%, #bbdefb 100%);
  min-height: 100vh;
  display: flex;
  justify-content: center;
  align-items: center;
  padding: 20px;
}

.container {
  background-color: white;
  border-radius: 10px;
  box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
  width: 100%;
  max-width: 500px;
  padding: 40px;
}

h1 {
  color: #1976d2;
  text-align: center;
  margin-bottom: 30px;
  font-size: 32px;
  font-weight: 600;
}

.input-section {
  display: flex;
  gap: 10px;
  margin-bottom: 30px;
}

input[type='text'] {
  flex: 1;
  padding: 12px 15px;
  border: 2px solid #e0e0e0;
  border-radius: 5px;
  font-size: 16px;
  transition: border-color 0.3s;
}

input[type='text']:focus {
  outline: none;
  border-color: #1976d2;
}

button {
  padding: 12px 30px;
  background-color: #1976d2;
  color: white;
  border: none;
  border-radius: 5px;
```

```
  font-size: 16px;
  font-weight: 600;
  cursor: pointer;
  transition: background-color 0.3s;
}

button:hover {
  background-color: #1565c0;
}

button:active {
  background-color: #0d47a1;
}

.task-list {
  list-style: none;
}

.task-item {
  background-color: #f5f5f5;
  padding: 15px;
  margin-bottom: 10px;
  border-radius: 5px;
  border-left: 4px solid #1976d2;
  display: flex;
  justify-content: space-between;
  align-items: center;
  transition: background-color 0.3s;
}

.task-item:hover {
  background-color: #eeeeee;
}

.task-text {
  color: #333333;
  font-size: 16px;
  flex: 1;
}

.delete-btn {
  padding: 6px 12px;
  background-color: #e53935;
  color: white;
  border: none;
  border-radius: 3px;
  font-size: 14px;
  cursor: pointer;
  transition: background-color 0.3s;
}

.delete-btn:hover {
  background-color: #c62828;
}
```

```
    .empty-message {
      text-align: center;
      color: #999999;
      padding: 20px;
      font-size: 14px;
    }
  </style>
</head>
<body>
  <div class="container">
    <h1>My To-Do List</h1>
    <div class="input-section">
      <input
        type="text"
        id="taskInput"
        placeholder="新しいタスクを入力してください..."
        onkeypress="handleKeyPress(event)"
      />
      <button onclick="addTask()">追加</button>
    </div>
    <ul id="taskList" class="task-list">
      <li class="empty-message">タスクはまだありません</li>
    </ul>
  </div>
  <script>
    const taskInput = document.getElementById('taskInput');
    const taskList = document.getElementById('taskList');
    const STORAGE_KEY = 'todoListTasks';

    // ページ読み込み時にローカルストレージからタスクを復元
    function loadTasksFromStorage() {
      const savedTasks = localStorage.getItem(STORAGE_KEY);
      if (savedTasks) {
        const tasks = JSON.parse(savedTasks);
        const emptyMessage = taskList.querySelector('.empty-message');
        if (emptyMessage) {
          emptyMessage.remove();
        }

        tasks.forEach((taskText) => {
          createTaskElement(taskText);
        });
      }
    }

    // ローカルストレージにタスクを保存
    function saveTasksToStorage() {
      const tasks = [];
      const taskItems = taskList.querySelectorAll('.task-item');
      taskItems.forEach((item) => {
        const taskText = item.querySelector('.task-text').textContent;
        tasks.push(taskText);
      });
      localStorage.setItem(STORAGE_KEY, JSON.stringify(tasks));
    }
```

```
// タスク要素を作成
function createTaskElement(taskText) {
  const taskItem = document.createElement('li');
  taskItem.className = 'task-item';
  taskItem.innerHTML = `
          <span class="task-text">${escapeHtml(taskText)}</span>
          <button class="delete-btn" onclick="deleteTask(this)">削除</button>
      `;
  taskList.appendChild(taskItem);
}

function addTask() {
  const taskText = taskInput.value.trim();

  if (taskText === '') {
    alert('タスクを入力してください');
    return;
  }

  // 空のメッセージを削除
  const emptyMessage = taskList.querySelector('.empty-message');
  if (emptyMessage) {
    emptyMessage.remove();
  }

  createTaskElement(taskText);
  taskInput.value = '';
  taskInput.focus();

  // ローカルストレージに保存
  saveTasksToStorage();
}

function deleteTask(button) {
  if (!confirm('本当に削除しますか？')) {
    return;
  }

  button.parentElement.remove();

  // リストが空になったら、メッセージを表示
  if (taskList.children.length === 0) {
    const emptyMessage = document.createElement('li');
    emptyMessage.className = 'empty-message';
    emptyMessage.textContent = 'タスクはまだありません';
    taskList.appendChild(emptyMessage);
  }

  // ローカルストレージに保存
  saveTasksToStorage();
}

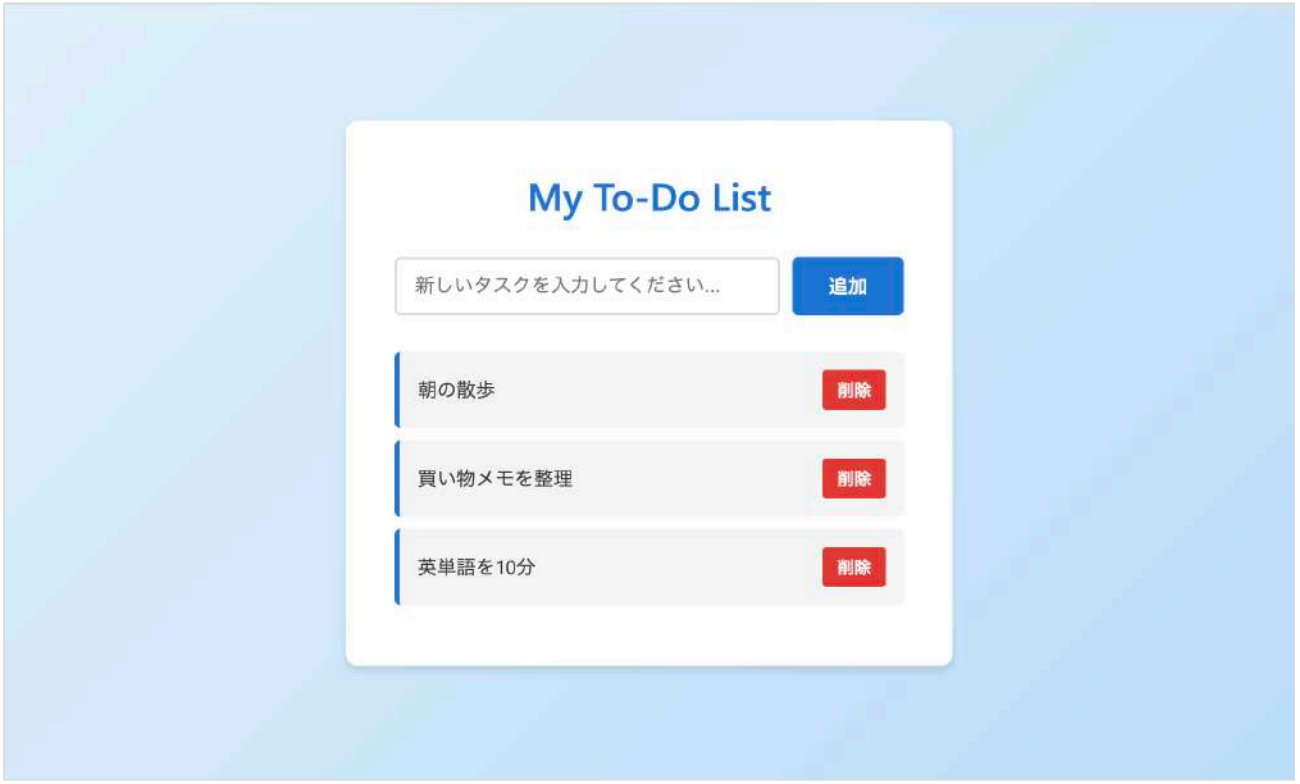
function handleKeyPress(event) {
  if (event.key === 'Enter') {
```

```
            addTask();
          }
        }

        function escapeHtml(text) {
          const map = {
            '&': '&amp;',
            '<': '&lt;',
            '>': '&gt;',
            '"': '&quot;',
            "'": '&#039;',
          };
          return text.replace(/[&<>"']/g, (m) => map[m]);
        }

        // ページ読み込み時に実行
        window.addEventListener('DOMContentLoaded', loadTasksFromStorage);
      </script>
    </body>
  </html>
```

動作例(リロード後もタスクが残る)



AIとの協力ポイント

AIに依頼するときは、次のポイントを意識すると精度が上がります。

- 変更点はどの関数のどこを置き換えるかを必ずセットで伝える。
- 追加後は必ずリロードして、動作を確認する。
- 動かなかったら、エラー文をそのまま貼って相談する。

この3つを守るだけで、AIとの対話は格段にスムーズになり、修正の手戻りも減ります。

学習のポイント: DOM操作と保存の基本命令を整理します。

- `document.createElement('タグ名')`：HTMLに最初から書いておくのではなく、JavaScriptの力で何もないところから新しいタグを生み出す命令です。
- `parent.appendChild(child)`：生み出したタグを、親となる要素の中に子要素として組み込みます。これを行って初めて画面に表示されます。
- `element.remove()`：指定した要素を画面から消去します。削除ボタンの中にこの命令を仕込んでおくことで、削除ボタンが含まれている行(li)を消す機能を実現しています。
- `window.localStorage`：ブラウザにデータを保存するための仕組みです。これを使うことで、ページを閉じてもデータが消えないアプリを作れます。
- `JSON.stringify()` と `JSON.parse()`：配列やオブジェクトを文字列に変換したり、文字列から元の形に戻したりするための命令です。ローカルストレージには文字列しか保存できないため、これらを使ってデータをやり取りします。

これらの命令をセットで理解すると、画面に追加する・削除する・保存するという基本操作を自分の手で組み立てられるようになります。

この2つの演習を通じて、画面にあるものを操作する(カウンター)だけでなく、画面の構造そのものを作り変える(ToDoリスト)こともできるようになりました。 これができれば、掲示板やチャットアプリなど、より複雑なアプリケーションを作る基礎が固まったことになります。

問題1 料理レシピページ（HTML構造）

チャットで要件を聞き取り、それを満たすWebページを作成してください。

1. 概要

料理レシピを表示する静的HTMLページを作成してください。セマンティックなHTML要素を適切に使用します。

2. 要件

1. `<h1>` タグで「カレーライスの作り方」を表示
2. `<h2>` タグで「材料」を表示
3. `<ul>` タグ内に以下の4つの `<li>` を表示:
    - 豚肉 200g
    - じゃがいも 2個
    - にんじん 1本
    - カレールー 1箱
4. `<h2>` タグで「手順」を表示
5. `<ol>` タグ内に以下の3つの `<li>` を表示:
    - 野菜と肉を一口大に切る
    - 鍋で炒めてから水を加えて煮込む
    - カレールーを入れて溶かす

3. 提出

`index.html` ファイルを提出してください。

問題2 プロフィールカード（CSSスタイリング）

チャットで要件を聞き取り、それを満たすWebページを作成してください。

1. 概要

CSSスタイリングを含むプロフィールカードのページを作成してください。

2. 要件

1. `<div>` 要素に `class="card"` を指定
2. `.card` のCSSスタイル:
   - `background-color: #f0f0f0`
   - `border-radius: 8px`
   - `padding: 16px`
3. `.card` 内に `<h2>` タグで「田中太郎」を表示
4. `.card` 内に `<p>` タグ（ `class="role"` ）で「エンジニア」を表示

3. 提出

`index.html` ファイルを提出してください。

問題3 カウンターアプリ（JavaScriptインタラクション）

チャットで要件を聞き取り、それを満たすWebページを作成してください。

1. 概要

JavaScriptでボタンクリック時にDOM操作を行うカウンターアプリのページを作成してください。

2. 要件

1. `id="count"` の要素にカウント値を表示（初期値は `0` ）
2. `id="increment"` のボタン: クリックするとカウントが1増加
3. `id="decrement"` のボタン: クリックするとカウントが1減少

3. 提出

`index.html` ファイルを提出してください。

第13章 ウェブアプリ（クライアントサイド）の自由制作

これまでの講義で、HTML、CSS、JavaScriptというWebアプリを作るための3つの道具と、それをAIに書いてもらう方法を扱いました。 知識のインプットはここまでで十分です。ここからは、オリジナリティを発揮する自由制作の時間に入ります。

何を作ればいいか迷っている場合でも大丈夫です。この節では、アイデアの出し方から、AIを使って形にし、さらにブラッシュアップするまでの工程をガイドします。

自由制作

プログラミング学習で最も力がつくのは、教科書の例題を写すときではなく、自分が作りたいものを作ろうとして壁にぶつかり、それを乗り越えたときです。 AIという強力な相棒がいる今、アイデアさえあればどんなアプリでも作り始めることができます。

アイデア出し

まず何を作るかを決めます。 ルールはただ一つ、ブラウザだけで完結するアプリであることです。データベースやユーザー登録が必要な複雑なシステムは一旦脇に置きます。

アイデアのヒント

初心者が取り組みやすく、かつ作っていて楽しいテーマの例を挙げます。

- 実用ツール系:
    - 単位変換機: インチをセンチに直したり、ドルを円に換算したりするツール。
    - 割り勘電卓: 合計金額と人数を入れると、一人あたりの支払額を計算してくれるアプリ。
    - ポモドーロタイマー: 25分の作業と5分の休憩を管理するタイマー。
    - パスワード生成機: 強力なパスワードをランダムに作ってくれるツール。
- エンタメ系:
    - おみくじアプリ: ボタンを押すと大吉や凶がランダムに出る。
    - 3択クイズ: 自分の好きなジャンル（アニメ、歴史、料理など）のクイズを出題する。
    - 性格診断: 質問に答えていくと、性格タイプを判定してくれる。

モチベーション維持のコツ

最も重要なのは、自分の趣味や興味に関連したテーマを選ぶことです。 例えば、料理が好きなら計量スプーンの換算表、釣りが好きなら釣った魚のサイズ記録メモなど、自分自身が「これがあったらちょっと便利かも」「楽しいかも」と思えるものを選びます。誰かのためにではなく自分のために作ることが、完成までモチベーションを保つ秘訣です。

プロトタイピング

作りたいものが決まったら、すぐに完璧な完成品を目指すのではなく、まずプロトタイプ（試作品）を作ります。 プロトタイプとは、デザインは二の次にして、核となる機能だけが動く状態のものです。

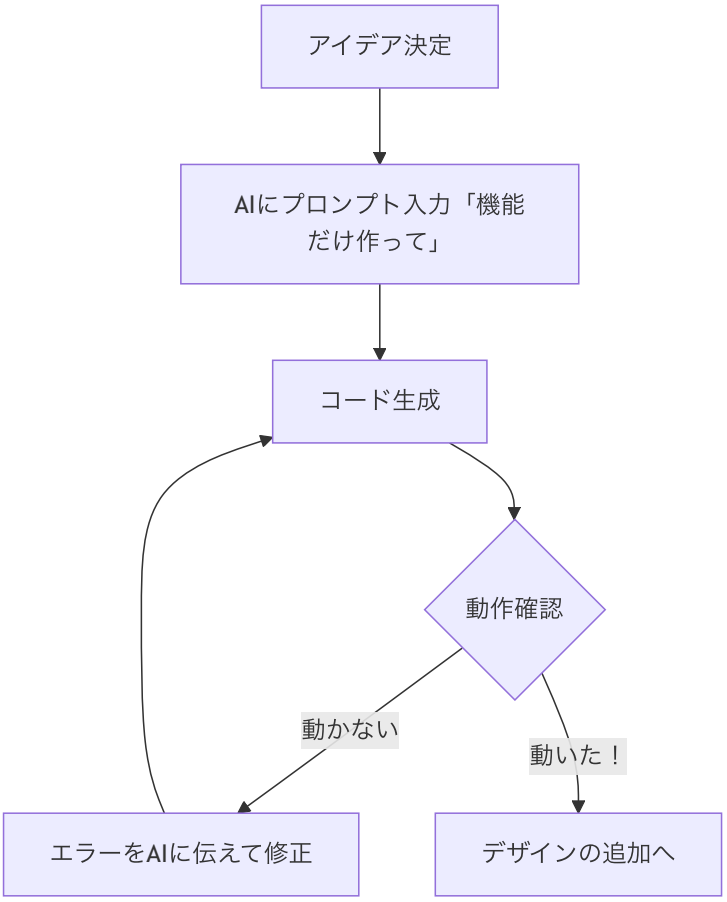
AIを活用した高速プロトタイピング

AIへの指示も、まずは機能だけに絞るのがポイントです。いきなり「おしゃれでアニメーションがあって...」と複雑な要求をすると、AIも混乱してエラーが出やすくなります。

開発のステップ:

1. 機能の実装: まず動くものを作る（見た目は真っ白でOK）。
2. 動作確認: 本当に計算できているか、ボタンは反応するか確認する。
3. デザイン追加: 後からCSSで見栄えを良くする。

この流れを図にすると以下のようになります。



実践：デジタル時計のプロトタイプ

例として、現在時刻を表示するデジタル時計を作ってみます。 まずは機能だけをAIに作らせます。

> ***プロンプト例:*** *現在時刻（時：分：秒）を表示するだけのシンプルなWebページを作ってください。 デザインは不要です。 JavaScriptを使って、1秒ごとに時間が更新されるようにしてください。 HTMLファイル1つにまとめてください。*

AIが生成するコード（プロトタイプ）は以下のようになります。

```
<!DOCTYPE html>
<html lang="ja">
  <head>
    <meta charset="UTF-8" />
    <title>デジタル時計（プロトタイプ）</title>
  </head>
  <body>
    <!-- ここに時間が表示される -->
    <div id="clock">Loading...</div>

    <script>
      function updateTime() {
        // 現在の日時データを取得
        const now = new Date();

        // 時間、分、秒を取り出す
        const hours = now.getHours();
        const minutes = now.getMinutes();
```

```
        const seconds = now.getSeconds();

        // 画面に表示する文字列を作る
        // 見やすくするために ": " でつなぐ
        const timeString = hours + ':' + minutes + ':' + seconds;

        // 画面を更新する
        document.getElementById('clock').textContent = timeString;
      }

      // 1000ミリ秒（＝1秒）ごとに updateTime 関数を実行するタイマー
      setInterval(updateTime, 1000);

      // ページを開いた瞬間に一度実行しておく（待たずに表示するため）
      updateTime();
    </script>
  </body>
</html>
```

このコードを `clock-proto.html` として保存して開いてみてください。 左上に小さく数字が出るだけの素っ気ない画面ですが、1秒ごとに時間が進むというコア機能は完璧に動作しています。これでプロトタイプは成功です。

ブラッシュアップ

コア機能が動くことを確認したら、次はブラッシュアップ（磨き上げ）の工程です。 以下の2点に注目して改善を行います。

1. デザイン（CSS）：色、フォント、配置を整えて、見た目を魅力的にする。
2. ユーザー体験（UX）：数字が読みやすい大きさか、目に優しい色使いかなど、使い心地を良くする。

AIへの追加指示

プロトタイプのコードを元に、AIに追加の指示を出します。

> ***追加プロンプト例:*** *さきほどの時計のコードをブラッシュアップしてください。*
>
> 1. *画面の中央に大きく時間を表示してください。*
> 2. *背景は黒、文字はネオンのような明るい緑色にして、サイバーパンク風のデザインにしてください。*
> 3. *「時」「分」「秒」が常に2桁になるように（例: 9時 → 09時）調整してください。*
> 4. *Google Fontsからデジタル時計っぽいフォント（'Orbitron'など）を読み込んで使ってください。*

実践：完成版デジタル時計

これらの要望を取り入れると、コードは以下のように進化します。

```
<!DOCTYPE html>
<html lang="ja">
  <head>
    <meta charset="UTF-8" />
    <title>ネオン時計</title>
    <!-- Google Fontsからデジタル風フォントを読み込む -->
    <link href="https://fonts.googleapis.com/css2?family=Orbitron:wght@500&display=swap" rel="stylesheet" />
    <style>
      /* 全体の設定：画面いっぱいを使って中央揃えにする */
      body {
        background-color: #0d0d0d; /* ほぼ真っ黒な背景 */
        color: #0f0; /* 基本の文字色は緑 */
        height: 100vh; /* 画面の高さ100% */
```

```
        margin: 0;
        display: flex; /* フレックスボックス（配置用ツール） */
        justify-content: center; /* 横方向の中央 */
        align-items: center; /* 縦方向の中央 */
        font-family: 'Orbitron', sans-serif; /* デジタル風フォント適用 */
      }

      /* 時計の枠組み */
      .clock-container {
        border: 4px solid #0f0; /* 緑色の枠線 */
        padding: 40px 80px;
        border-radius: 20px; /* 角を丸く */
        box-shadow:
          0 0 20px #0f0,
          inset 0 0 20px #0f0; /* 外側と内側に光る効果 */
        background-color: #000;
      }

      /* 時間表示の文字 */
      #clock {
        font-size: 80px;
        text-shadow: 0 0 20px #0f0; /* 文字自体も光らせる */
        letter-spacing: 5px; /* 文字の間隔を少し空ける */
      }

      /* 秒の部分だけ少し小さくするなどの工夫も可能 */
    </style>
  </head>
  <body>
    <div class="clock-container">
      <div id="clock">00:00:00</div>
    </div>

    <script>
      function updateTime() {
        const now = new Date();

        // padStart(2, '0') は「2桁になるように左側を0で埋める」という命令
        // 例: 9 -> "09", 12 -> "12"
        const hours = String(now.getHours()).padStart(2, '0');
        const minutes = String(now.getMinutes()).padStart(2, '0');
        const seconds = String(now.getSeconds()).padStart(2, '0');

        const timeString = `${hours}:${minutes}:${seconds}`;

        document.getElementById('clock').textContent = timeString;
      }

      setInterval(updateTime, 1000);
      updateTime();
    </script>
  </body>
</html>
```

解説:

- デザインの変化: CSSの `box-shadow` や `text-shadow` を使うことで、発光しているようなネオン管の表現が可能になりました。
- UXの向上: `padStart(2, '0')` というテクニックを使うことで、時刻が「9:5:1」ではなく「09:05:01」と表示され、桁がズレずに見やすくなりました。

このように、まず機能を作り（プロトタイプ）、後から見た目と使い勝手を良くする（ブラッシュアップ）という手順を踏むことで、挫折せずに完成度の高いアプリを作ることができます。

制作例

自由制作の例を紹介します。 あくまで参考ですので、自分自身の興味やニーズに合わせてオリジナルのアイデアを考えてみてください。

文字数チェッカー

テキストを入力すると、その文字数をリアルタイムでカウントして表示するアプリです。

プロンプト例:

> **テキストエリアに入力された文字数をリアルタイムでカウントして表示するWebアプリを1つのHTMLファイルで作ってください。**

完成例:




```
<!DOCTYPE html>
<html lang="ja">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>文字数カウンター</title>
    <style>
      * {
        margin: 0;
        padding: 0;
        box-sizing: border-box;
      }
```

```css
body {
  font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
  background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
  min-height: 100vh;
  display: flex;
  justify-content: center;
  align-items: center;
  padding: 20px;
}

.container {
  background: white;
  border-radius: 10px;
  box-shadow: 0 10px 40px rgba(0, 0, 0, 0.3);
  padding: 40px;
  max-width: 600px;
  width: 100%;
}

h1 {
  color: #333;
  margin-bottom: 30px;
  text-align: center;
  font-size: 28px;
}

.input-group {
  position: relative;
  margin-bottom: 20px;
}

label {
  display: block;
  margin-bottom: 10px;
  color: #555;
  font-weight: 600;
}

textarea {
  width: 100%;
  padding: 15px;
  border: 2px solid #ddd;
  border-radius: 8px;
  font-size: 16px;
  font-family: inherit;
  resize: vertical;
  min-height: 200px;
  transition: border-color 0.3s ease;
}

textarea:focus {
  outline: none;
  border-color: #667eea;
}
```

```css
.counter-display {
  display: flex;
  gap: 20px;
  margin-top: 20px;
  flex-wrap: wrap;
}

.counter-box {
  flex: 1;
  min-width: 150px;
  background: #f5f5f5;
  padding: 20px;
  border-radius: 8px;
  text-align: center;
  border: 2px solid #e0e0e0;
}

.counter-box h2 {
  font-size: 14px;
  color: #999;
  margin-bottom: 10px;
  text-transform: uppercase;
  letter-spacing: 1px;
}

.counter-value {
  font-size: 36px;
  font-weight: bold;
  color: #667eea;
}

.counter-box.warning .counter-value {
  color: #ff9800;
}

.counter-box.danger .counter-value {
  color: #f44336;
}

.info-text {
  text-align: center;
  color: #999;
  font-size: 12px;
  margin-top: 15px;
}

.settings-box {
  margin-top: 20px;
  padding: 15px;
  background: #f9f9f9;
  border-radius: 8px;
  border: 1px solid #e0e0e0;
}
```

```css
.toggle-container {
  display: flex;
  align-items: center;
  justify-content: space-between;
  gap: 15px;
}

.toggle-label {
  color: #555;
  font-size: 14px;
  font-weight: 500;
}

.toggle-switch {
  position: relative;
  display: inline-block;
  width: 50px;
  height: 26px;
}

.toggle-switch input {
  opacity: 0;
  width: 0;
  height: 0;
}

.slider {
  position: absolute;
  cursor: pointer;
  top: 0;
  left: 0;
  right: 0;
  bottom: 0;
  background-color: #ccc;
  transition: 0.3s;
  border-radius: 26px;
}

.slider:before {
  position: absolute;
  content: '';
  height: 20px;
  width: 20px;
  left: 3px;
  bottom: 3px;
  background-color: white;
  transition: 0.3s;
  border-radius: 50%;
}

input:checked + .slider {
  background-color: #667eea;
}

input:checked + .slider:before {
```

```
        transform: translateX(24px);
      }
    </style>
  </head>
  <body>
    <div class="container">
      <h1>📝 文字数カウンター</h1>
      <div class="input-group">
        <label for="textInput">テキストを入力してください：</label>
        <textarea id="textInput" placeholder="ここに文字を入力すると、文字数がリアルタイムで表示されます...">
</textarea>
      </div>
      <div class="settings-box">
        <div class="toggle-container">
          <span class="toggle-label">改行を文字数にカウントする</span>
          <label class="toggle-switch">
            <input type="checkbox" id="includeNewlines" checked />
            <span class="slider"></span>
          </label>
        </div>
      </div>
      <div class="counter-display">
        <div class="counter-box">
          <h2>文字数</h2>
          <div class="counter-value" id="charCount">0</div>
        </div>
        <div class="counter-box">
          <h2>日本語文字数</h2>
          <div class="counter-value" id="japaneseCharCount">0</div>
        </div>
        <div class="counter-box">
          <h2>単語数</h2>
          <div class="counter-value" id="wordCount">0</div>
        </div>
        <div class="counter-box">
          <h2>行数</h2>
          <div class="counter-value" id="lineCount">1</div>
        </div>
      </div>
      <div class="info-text">リアルタイムでカウントされます</div>
    </div>
    <script>
      const textInput = document.getElementById('textInput');
      const includeNewlinesToggle = document.getElementById('includeNewlines');
      const charCount = document.getElementById('charCount');
      const japaneseCharCount = document.getElementById('japaneseCharCount');
      const wordCount = document.getElementById('wordCount');
      const lineCount = document.getElementById('lineCount');

      function updateCounts() {
        let text = textInput.value;

        // 改行をカウントするかどうか
        const includeNewlines = includeNewlinesToggle.checked;
```

```
        // 改行を除外する場合
        let textForCounting = text;
        if (!includeNewlines) {
          textForCounting = text.replace(/\r?\n/g, '');
        }

        // 文字数をカウント
        const chars = textForCounting.length;
        charCount.textContent = chars;

        // 日本語文字数をカウント（ひらがな、カタカナ、漢字のみ）
        const japaneseRegex = /[\u3040-\u309F\u30A0-\u30FF\u4E00-\u9FFF]/g;
        const japaneseChars = textForCounting.match(japaneseRegex) || [];
        japaneseCharCount.textContent = japaneseChars.length;

        // 単語数をカウント（空白で区切られた連続した文字を単語として数える）
        const words = text.trim().length === 0 ? 0 : text.trim().split(/\s+/).length;
        wordCount.textContent = words;

        // 行数をカウント（改行で区切られた行を数える）
        const lines = text.length === 0 ? 1 : text.split('\n').length;
        lineCount.textContent = lines;
      }

      // イベントリスナー設定
      textInput.addEventListener('input', updateCounts);
      textInput.addEventListener('change', updateCounts);
      includeNewlinesToggle.addEventListener('change', updateCounts);

      // 初期化
      updateCounts();
    </script>
  </body>
</html>
```

成果発表と相互評価

一生懸命作ったアプリケーションも、自分のパソコンの中に閉じ込めておくだけではもったいないです。 プログラミングの学習において、作ったものを誰かに見てもらいフィードバックをもらうことは、コードを書くことと同じくらい重要な成長のステップです。

この節では、完成した作品を他者に披露する方法と、お互いのコードから学び合うコードレビューについて解説します。

作品の公開

Webアプリにおける公開（デプロイ）とは、自分のパソコンにあるファイルをインターネット上のサーバーにアップロードして、世界中の誰もがアクセスできるようにすることです。 ただし、本格的なサーバーへの公開は手順がやや複雑です。まずは、もっと手軽な方法で作品を操作してもらえる状態にして発表してみます。

1. ファイルを直接共有する

今回作成したのは、HTMLファイル（とCSS/JS）だけで動くクライアントサイドのアプリです。 つまり、そのHTMLファイルをチャットツールやメールで友人に送るだけで、相手のパソコンのブラウザでも全く同じように動かすことができます。

- メリット: 特別な登録や設定が不要で、一番簡単です。
- 方法: 作成した `.html` ファイルを相手に送り、ブラウザで開いてもらいます。

2. デモンストレーション（実演）を行う

ファイルを送る前に、まずは自分が操作している画面を見せて、作品の魅力をプレゼンテーションします。

- アピールポイント:
  - なぜこれを作ったのか: アイデアのきっかけや、解決したかった課題。
  - どこにこだわったか: デザインのテーマや、使いやすさの工夫（UX）。
  - 苦労した点: AIになかなか意図が伝わらなかった部分や、エラーをどう乗り越えたか。

画面共有などを使いながら実演することで、静的なファイルだけでは伝わらない作り手の熱量を伝えることができます。

コードレビュー

作品を見せ合ったら、次は中身（プログラム）を見せ合います。これをコードレビューと呼びます。 プロの現場ではバグを見つけるために行われますが、ここではAIが書いたコードから面白い発見を共有する目的で行います。

AIにコードを書かせていると、動くけど中身はどうなっているのかよくわからないという状態になりがちです。発表を機に、AIが生成したロジックを読み解いてみてください。

何を共有すればいい？

コードレビューでは、以下のようなポイントに注目して話をしてみてください。

1. AIへの指示（プロンプト）と結果
   - 「サイバーパンク風にして」と頼んだら、AIが `box-shadow` （影をつける命令）を使ってネオンのような光を表現してくれた
   - 「計算ロジックを書いて」と頼んだら、こんな数式が生成された
2. AIが提案した知らない書き方
   - 前の節のデジタル時計の例でいえば、桁を揃えるために `padStart` という便利な機能があることをAIに教わった、といった発見です。

実際のレビュー例

例えば、以下のようなコードがあったとします。

```
// AIが生成した、ランダムな色を作るコード
function getRandomColor() {
  const letters = '0123456789ABCDEF';
  let color = '#';
  for (let i = 0; i < 6; i++) {
    color += letters[Math.floor(Math.random() * 16)];
  }
  return color;
}
```

このコードについて、以下のように発表・共有します。

> *背景色をランダムに変えたくてAIに頼んだら、この `getRandomColor` という関数を作ってくれました。 見てみると、16進数の文字（0-F）をランダムに6回選んで、 `#` の後ろにくっつけることで色コードを作っているようです。 最初は意味がわかりませんでしたが、AIに解説してもらって仕組みが理解できました。*

相互評価のメリット

- 自分では思いつかない発想を知る: 他の人がAIからどのようなコードを引き出したかを見ることで、新しいプロンプトのヒントが得られます。

- コードを読む力がつく: 他人のコード（AIが書いたコード）を読んで意味を理解しようとするプロセスは、プログラミング能力を大きく高めます。

互いの作品に刺激を受けることが、次の制作への一番のエネルギーになります。自信を持って自分の作品を発表してください。

問題1 ウェブアプリ自由制作

ブラウザで動作するウェブアプリを自由に制作してください。 `index.html` とその表示に必要なファイル一式を提出してください。

制約

- ページは `index.html` のみとする。
- HTMLファイルを直接ブラウザで開いたときに正しく表示・動作するように制作すること。

第14章 ブラウザゲームの開発方法

これまでの講義では、便利なツールとしてのWebアプリを作る方法を扱いました。今回は、多くの人がプログラミングに興味を持つきっかけとなるゲーム開発に挑戦します。

ゲーム開発は、単なるツールの作成とは異なり、アニメーションやリアルタイムな反応が求められるため、少し特別な仕組みが必要です。AIという相棒がいれば、かつては高度な技術が必要だったアクションゲームも、驚くほど手軽に作れます。

AIと協力して、バイブコーディングでブラウザゲームを開発するための基礎知識と、具体的な進め方を解説します。

まず、本章のコードを実行するための作業場所を準備します。

1. デスクトップに `ブラウザゲーム` という名前の作業用フォルダーを作成します。
2. VS Codeでそのフォルダーを開いておきます。

詳細な手順は10章にありますので、そちらを参照してください。

以降のコード例は、保存してブラウザで開き、リロードする流れで確認できます。 変更したらブラウザをリロードして反映します。

バイブコーディングによるブラウザゲームの開発

ゲーム開発は、ルールを決める、絵を動かす、操作を受け付けるという3つの要素で成り立っています。これらをAIにどう指示し、組み立てていくかを見ていきます。

ゲームのルール設計

コードを書き始める前に、どんなゲームにするかを言葉で明確にする必要があります。これを仕様策定やゲームデザインと呼びます。 AIは優秀なプログラマですが、面白いゲームのアイデアを自ら考えてくれるわけではありません。明確な設計図、つまりプロンプトを渡す必要があります。

言語化すべき4つの要素

AIにゲームを作らせる際は、最低限以下の4点を決めて伝えます。

1. 目的: プレイヤーは何を目指すのか。例えばスコアを稼ぐ、ゴールに辿り着くなど。
2. 操作方法: どうやって操作するのか。例えば矢印キーで移動、スペースキーでジャンプなど。
3. 勝利条件: どうなればクリアか。例えば敵を全滅させる、制限時間を生き残るなど。
4. 敗北条件: どうなればゲームオーバーか。例えば敵に当たる、穴に落ちるなど。

既存ゲームを参考にする（メタファーの活用）

ゼロから全く新しいルールを考えるのは大変ですし、AIに伝えるのも難しいです。 そこで有効なのが、既存の有名なゲームを例に出すことです。

- インベーダーゲーム風の、上から敵が攻めてくるシューティングゲーム
- ブロック崩しのように、ボールを跳ね返すゲーム
- Chromeの恐竜ゲームのような、横スクロールのランアクション

このように伝えるだけで、AIはその仕組みを理解し、弾の発射や重力の計算など一般的な挙動を補完してコードを書いてくれます。

画面描画（Canvas APIなど）

Webサイトのボタンやテキストを表示するにはHTMLのタグを使いましたが、ゲームでキャラクターを滑らかに動かすには、Canvasという仕組みを使うのが一般的です。

Canvasとは

`<canvas>` は、ブラウザ上に用意されたお絵描き用の画用紙のようなHTML要素です。 JavaScriptを使って、この画用紙の上に四角形や円、画像などをプログラムで描画します。

アニメーションの原理（パラパラ漫画）

ゲームの中でキャラクターが動いて見えるのは、実は描いては消すを猛スピードで繰り返しているからです。これをパラパラ漫画の原理と言います。

1. 画面を真っ白に消す。
2. キャラクターを少しズレた位置に描く。
3. 一瞬待つ。
4. 1に戻る。

これを1秒間に60回（60FPS）行うことで、人間の目には滑らかに動いているように見えます。

AIへの指示とコード例

まず、Canvasを使って赤い四角形を表示するだけのシンプルなコードをAIに書いてもらいます。 ここで大切なのはプロンプトの暗記ではなく、ゲームの構成要素を言葉で整理できるようになることです。

> *プロンプト例: HTML5のCanvas APIを使って、画面の中央に赤い正方形（50px × 50px）を描画するだけのHTMLファイルを作ってください。 JavaScriptで描画処理を行ってください。*

生成されるコードの一例は以下のようになります。

ファイル名: `canvas-basic.html`

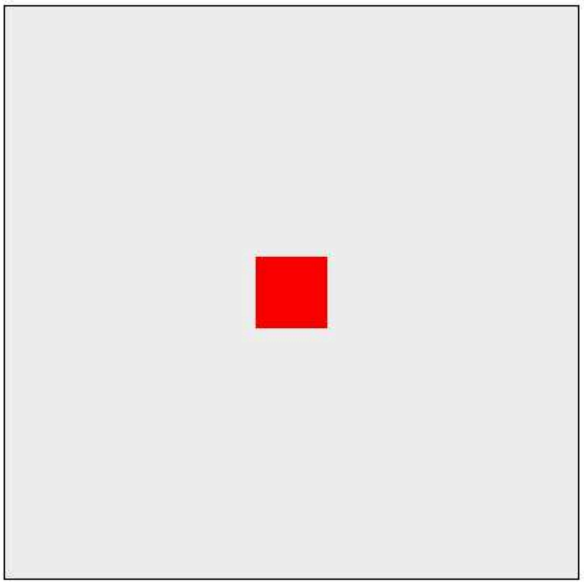
```
<!DOCTYPE html>
<html lang="ja">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Canvasの基本</title>
    <style>
      /* キャンバスを分かりやすく黒枠で囲む */
      canvas {
        border: 1px solid black;
        background-color: #eee; /* 背景は薄いグレー */
      }
      body {
        display: flex;
        justify-content: center;
        align-items: center;
        height: 100vh;
        margin: 0;
      }
    </style>
  </head>
  <body>
    <!-- ここがお絵描き領域 -->
    <canvas id="gameCanvas" width="400" height="400"></canvas>

    <script>
      // 1. キャンバス要素を取得
      const canvas = document.getElementById('gameCanvas');
      // 2. 描画するための「筆（コンテキスト）」を取得
      const ctx = canvas.getContext('2d');

      // 3. 色を赤に設定
      ctx.fillStyle = 'red';
```

```
      // 4. 四角形を描く（x座標，y座標，幅，高さ）
      // キャンバスの中心に描くための座標計算
      const x = (canvas.width - 50) / 2;
      const y = (canvas.height - 50) / 2;
      ctx.fillRect(x, y, 50, 50);
    </script>
  </body>
</html>
```

このコードを実行すると、グレーの領域の中央に赤い四角が表示されます。これがゲーム画面の第一歩です。



> *動作確認の流れ:* `canvas-basic.html` *を保存し、エクスプローラーからダブルクリックでブラウザ表示します。*

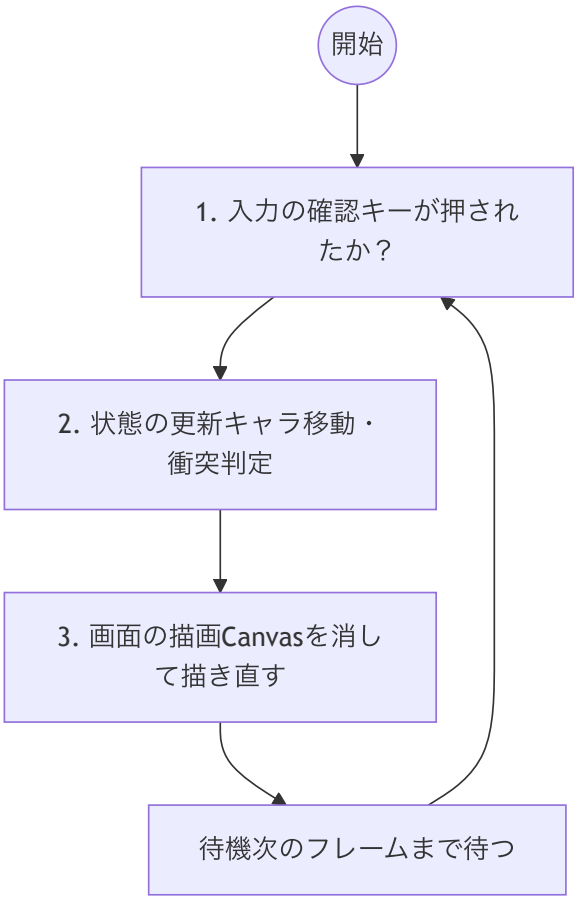
> *追加プロンプト例: 背景色や四角の色を変えて、見た目を分かりやすくしてください。*

操作と判定（ゲームループ）

静止画を表示するだけではゲームになりません。プレイヤーの操作を受け付け、キャラクターを動かし、敵に当たったかどうかを判定する必要があります。 これらを管理するのが、ゲームループと呼ばれる仕組みです。

ゲームループの構造

ゲームループは、ゲームが起動している間ずっと繰り返される無限ループです。1回のループ（フレーム）で以下の処理を順番に行います。



1. 入力の確認: 右キーが押されているかなどをチェックします。
2. 状態の更新:
    - キーが押されていれば、キャラクターの座標（X, Y）を変化させます。
    - 敵との距離を計算し、ぶつかっていたら（衝突判定）ゲームオーバーにします。
3. 画面の描画: 古い絵を消して、新しい座標にキャラクターを描き直します。

実践：動かせる四角形を作る

先ほどのコードをAIに進化させて、矢印キーで四角形を動かせるようにしてみます。

> *プロンプト例: 先ほどのCanvasのコードを改良してください。 矢印キーで赤い四角形を上下左右に操作できるようにしたいです。 `requestAnimationFrame` を使って、滑らかなゲームループを実装してください。*

生成されるコードの一例です。

ファイル名: `move-rect.html`

```
<!DOCTYPE html>
<html lang="ja">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>四角形を動かす</title>
    <style>
      canvas {
        border: 1px solid black;
        background-color: #eee;
```

```
    }
    body {
      display: flex;
      justify-content: center;
      align-items: center;
      height: 100vh;
      margin: 0;
      overflow: hidden;
    }
  </style>
</head>
<body>
  <canvas id="gameCanvas" width="400" height="400"></canvas>

  <script>
    const canvas = document.getElementById('gameCanvas');
    const ctx = canvas.getContext('2d');

    // キャラクターの状態（位置とサイズ）
    const player = {
      x: 175, // 初期X座標
      y: 175, // 初期Y座標
      size: 50,
      speed: 5,
    };

    // キーの入力状態を記録するオブジェクト
    const keys = {
      ArrowUp: false,
      ArrowDown: false,
      ArrowLeft: false,
      ArrowRight: false,
    };

    // キーが押されたら true にする
    document.addEventListener('keydown', (e) => {
      if (keys.hasOwnProperty(e.code)) keys[e.code] = true;
    });

    // キーが離されたら false にする
    document.addEventListener('keyup', (e) => {
      if (keys.hasOwnProperty(e.code)) keys[e.code] = false;
    });

    // ゲームループ関数（繰り返し呼ばれる）
    function gameLoop() {
      // --- 1. 更新処理 (Update) ---
      if (keys.ArrowUp) player.y -= player.speed;
      if (keys.ArrowDown) player.y += player.speed;
      if (keys.ArrowLeft) player.x -= player.speed;
      if (keys.ArrowRight) player.x += player.speed;

      // 画面外に出ないように制限（壁の衝突判定のようなもの）
      if (player.x < 0) player.x = 0;
      if (player.x > canvas.width - player.size) player.x = canvas.width - player.size;
```

```
        if (player.y < 0) player.y = 0;
        if (player.y > canvas.height - player.size) player.y = canvas.height - player.size;

        // --- 2. 描画処理 (Draw) ---

        // 画面をクリア（前のフレームの絵を消す）
        ctx.clearRect(0, 0, canvas.width, canvas.height);

        // プレイヤーを描画
        ctx.fillStyle = 'red';
        ctx.fillRect(player.x, player.y, player.size, player.size);

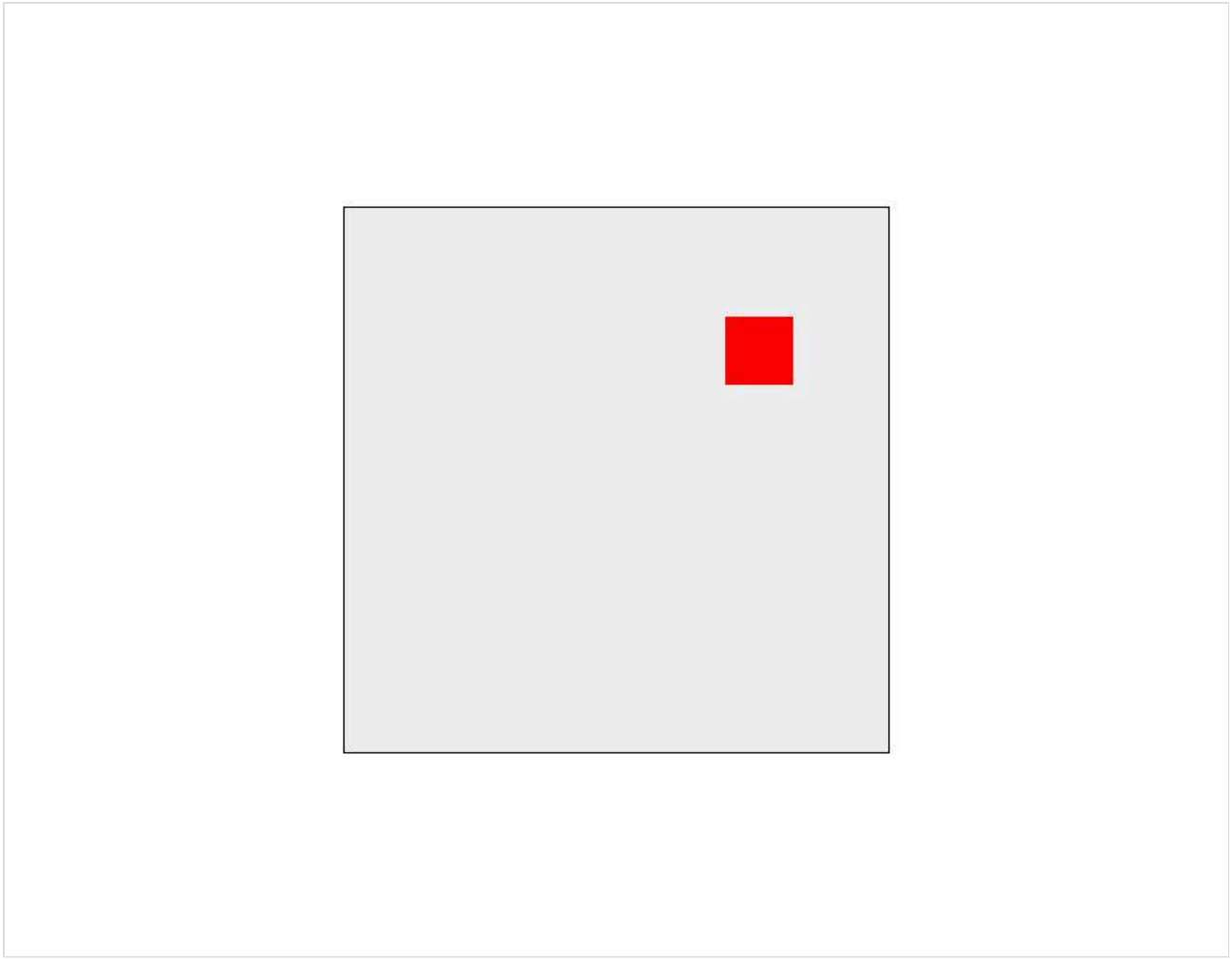
        // --- 3. 次のフレームを予約 ---
        // ブラウザに「次の描画準備ができたら教えて」と依頼する
        requestAnimationFrame(gameLoop);
      }

      // ゲームループを開始
      gameLoop();
    </script>
  </body>
</html>
```

コードの解説です。

- `keys` オブジェクト: どのキーが押されているかを記憶しておきます。
- `gameLoop` 関数: これが心臓部です。
  - 更新: `player.x` や `player.y` の数値を書き換えることで移動を表現します。
  - 描画: `ctx.clearRect` で一度画面を消してから、新しい位置に `fillRect` します。
  - `requestAnimationFrame(gameLoop)`：画面の描画タイミングに合わせて繰り返し実行されるため、滑らかな動きになります。

入力、更新、描画のサイクルさえ理解していれば、あとは敵を表示させたり、スコアを表示させたりと、いくらでも機能を拡張していけます。 衝突判定（当たり判定）の計算などは数学が必要で少し難しいですが、そここそAIにプレイヤーと敵が重なったらゲームオーバーにしてと任せれば良い部分です。



> *動作確認の流れ:* `move-rect.html` *を保存し、ブラウザで開いて矢印キーで動かして確認します。*

> *追加プロンプト例: キャラクターの移動速度を変えられるように、* `speed` *をスライダーで調整できるようにしてください。*

演習（プログラミング問題）

前節で学んだゲームループとCanvasの知識を活かして、実際に動くアクションゲームを形にします。AIを強力なアシスタントとして使いながら、ゲーム開発の核心部分である当たり判定の仕組みを深く理解することを目指します。

シンプルなアクションゲーム

今回作成するのは、画面の下にいるプレイヤーを左右に動かして、上から次々と落ちてくる障害物（敵）を避ける落下物回避ゲームです。

ゲームの仕様

AIに指示を出すための仕様を整理します。

1. プレイヤー: 画面下部に配置。左右の矢印キーで移動する。
2. 障害物: 画面上部のランダムな位置から出現し、下に向かって一定速度で落ちてくる。
3. 勝利条件: 避け続けることでスコアが増えていく。
4. 敗北条件: 障害物に当たったらゲームオーバー。

AIへのプロンプト例

以下のようなプロンプトをAIに投げて、ベースとなるコードを生成させます。 このプロンプトは一例なので、表現や順序が違っても問題ありません。

> *プロンプト: HTMLとJavaScriptで、落ちてくる障害物を避けるゲームを作ってください。*
>
> - *Canvas（400x600）を使用してください。*
> - *プレイヤーは青い四角で、左右キーで動かせます。*
> - *障害物は赤い四角で、上からランダムに落ちてきます。*
> - *障害物が下まで行ったら消して、新しい障害物を上に生成してください。*
> - *プレイヤーと障害物が当たったら、アラートで「Game Over」と表示して停止してください。*

生成例コード

AIが生成するコードの一例は以下のようになります。出力は同一にならないため、あくまで参考として読み、特に当たり判定がどのように行われているかに注目してください。

ファイル名: `falling-game.html`

```
<!DOCTYPE html>
<html lang="ja">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>落下物回避ゲーム</title>
    <style>
      body {
        display: flex;
        flex-direction: column;
        align-items: center;
        background-color: #222;
        color: white;
        margin: 0;
        padding-top: 20px;
      }
      canvas {
        border: 2px solid #555;
        background-color: #000;
      }
      .info {
        margin-bottom: 10px;
        font-family: sans-serif;
      }
    </style>
  </head>
  <body>
    <div class="info">Score: <span id="score">0</span></div>
    <canvas id="gameCanvas" width="400" height="600"></canvas>

    <script>
      const canvas = document.getElementById('gameCanvas');
      const ctx = canvas.getContext('2d');
      const scoreElement = document.getElementById('score');

      // 設定
      const player = { x: 180, y: 540, width: 40, height: 40, speed: 7 };
      const enemies = [];
      const enemySize = 30;
      let score = 0;
      let isGameOver = false;
```

```
// キー入力管理
const keys = {};
window.addEventListener('keydown', (e) => (keys[e.code] = true));
window.addEventListener('keyup', (e) => (keys[e.code] = false));

// 障害物を1つ作成する関数
function createEnemy() {
  const x = Math.random() * (canvas.width - enemySize);
  enemies.push({ x: x, y: -enemySize, width: enemySize, height: enemySize, speed: 3 + Math.random() * 3 });
}

// 定期的に障害物を生成（1秒ごと）
setInterval(() => {
  if (!isGameOver) createEnemy();
}, 1000);

// 当たり判定のロジック
function checkCollision(rect1, rect2) {
  return (
    rect1.x < rect2.x + rect2.width &&
    rect1.x + rect1.width > rect2.x &&
    rect1.y < rect2.y + rect2.height &&
    rect1.y + rect1.height > rect2.y
  );
}

function gameLoop() {
  if (isGameOver) return;

  // --- 1. 更新 (Update) ---
  if (keys.ArrowLeft && player.x > 0) player.x -= player.speed;
  if (keys.ArrowRight && player.x < canvas.width - player.width) player.x += player.speed;

  for (let i = enemies.length - 1; i >= 0; i--) {
    const enemy = enemies[i];
    enemy.y += enemy.speed;

    // 当たり判定のチェック
    if (checkCollision(player, enemy)) {
      isGameOver = true;
      alert('Game Over! Score: ' + score);
      location.reload(); // リロードして最初から
    }

    // 画面外に出た障害物の削除とスコア加算
    if (enemy.y > canvas.height) {
      enemies.splice(i, 1);
      score += 10;
      scoreElement.innerText = score;
    }
  }

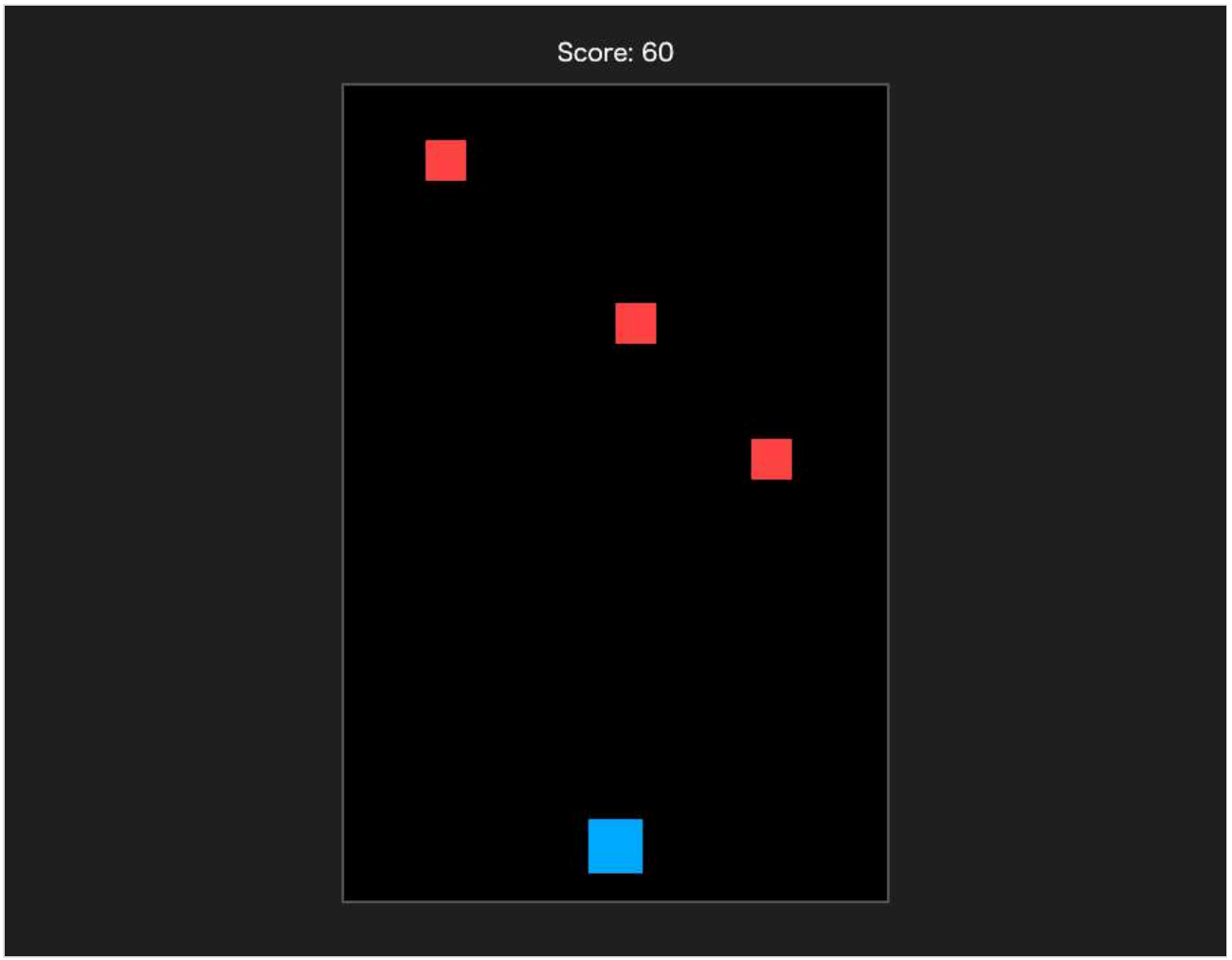
  // --- 2. 描画 (Draw) ---
  ctx.clearRect(0, 0, canvas.width, canvas.height);
```

```
        // プレイヤー描画
        ctx.fillStyle = '#00aaff';
        ctx.fillRect(player.x, player.y, player.width, player.height);

        // 障害物描画
        ctx.fillStyle = '#ff4444';
        enemies.forEach((enemy) => {
          ctx.fillRect(enemy.x, enemy.y, enemy.width, enemy.height);
        });

        requestAnimationFrame(gameLoop);
      }

      gameLoop();
    </script>
  </body>
</html>
```



動作確認の流れ: `falling-game.html` *を保存し、ブラウザで開いて左右キーで動かして確認します。*

追加プロンプト例: ゲーム開始ボタンとリスタートボタンを追加してください。ゲームオーバー時はボタンで再開できるようにしてください。

追加プロンプト例: スコアの最高記録（ハイスコア）をローカルストレージに保存して、次回起動時に表示してください。

補足: このサンプルでは、敵の生成に `setInterval` を使い、描画と移動は `requestAnimationFrame` のゲームループで行っています。 難易度の調整を入れる場合は、 `setTimeout` を使って出現間隔を毎回計算し直す方法が分かりやすいです。

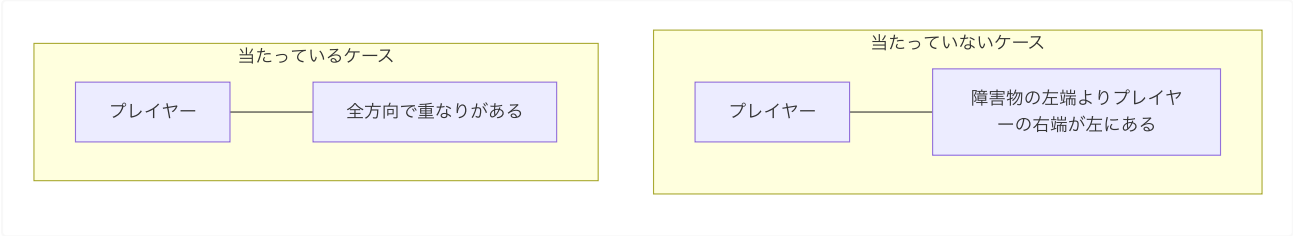
当たり判定のロジックをAIがどう実装するかを確認する

上記のコードの中で最も重要なのが、 `checkCollision` 関数です。AIはこのシンプルな数行のコードで、2つの物体が重なった瞬間を正確に検出しています。

矩形（長方形）同士の当たり判定の原理

コンピュータが当たったと判断するには、図形同士が重なっているかどうかを、座標（数値）の比較で判定する必要があります。

以下の図は、プレイヤー（青）と障害物（赤）が当たっていない状態と、当たっている状態を比較したものです。



この当たっている状態を判定するために、AIは当たっていない条件をすべて否定するという論理を使っています。

具体的には、以下の4つの条件がすべて満たされたときに当たっていると言えます。

1. `rect1.x < rect2.x + rect2.width` : 1の左端が、2の右端より左にある
2. `rect1.x + rect1.width > rect2.x` : 1の右端が、2の左端より右にある
3. `rect1.y < rect2.y + rect2.height` : 1の上端が、2の下端より上にある
4. `rect1.y + rect1.height > rect2.y` : 1の下端が、2の上端より下にある

コードの解説

さきほどのコードをもう一度見てみます。

```
function checkCollision(rect1, rect2) {
  return (
    rect1.x < rect2.x + rect2.width && // 条件1
    rect1.x + rect1.width > rect2.x && // 条件2
    rect1.y < rect2.y + rect2.height && // 条件3
    rect1.y + rect1.height > rect2.y // 条件4
  );
}
```

- `rect1` と `rect2` は、それぞれ `x, y, width, height` という情報を持ったオブジェクトです。
- `&&` は「かつ（AND）」という意味で、すべての式が `true` になったときだけ、この関数全体の結果が `true` （当たっている）になります。

バイブコーディングの実践（対話で改善する）

ここからは、2〜3ターンの会話でゲームを改善していく流れを体験します。 一発で完成させるのではなく、少しずつ要望を足していくのがバイブコーディングのコツです。

ターン1：まずは遊べる状態にする

> *プロンプト: 先ほどの落下物回避ゲームに、ゲームオーバー画面を追加してください。 アラートではなく、キャンバスの中央に「Game Over」と表示し、スコアも表示してください。*

狙い: アラートをなくして、ゲームとしての体験を改善します。

ここでのポイントは、アラートの代わりにキャンバス上に表示することです。AIの出力は人によって異なるため、同じ実装にならなくても問題ありません。表示位置やフォントが多少違っていても、ゲームオーバーが画面内に表示されれば目的は達成されています。

生成例コード（ターン1）

以下は、ゲームオーバー画面を追加した場合の一例です。出力は同一にならないため、参考として確認してください。

ファイル名: `falling-game-turn1.html`

```
<!DOCTYPE html>
<html lang="ja">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>落下物回避ゲーム</title>
    <style>
      body {
        display: flex;
        flex-direction: column;
        align-items: center;
        background-color: #222;
        color: white;
        margin: 0;
        padding-top: 20px;
      }
      canvas {
        border: 2px solid #555;
        background-color: #000;
      }
      .info {
        margin-bottom: 10px;
        font-family: sans-serif;
      }
    </style>
  </head>
  <body>
    <div class="info">Score: <span id="score">0</span></div>
    <canvas id="gameCanvas" width="400" height="600"></canvas>

    <script>
      const canvas = document.getElementById('gameCanvas');
      const ctx = canvas.getContext('2d');
      const scoreElement = document.getElementById('score');

      const player = { x: 180, y: 540, width: 40, height: 40, speed: 7 };
      const enemies = [];
      const enemySize = 30;
      let score = 0;
      let isGameOver = false;

      const keys = {};
      window.addEventListener('keydown', (e) => (keys[e.code] = true));
      window.addEventListener('keyup', (e) => (keys[e.code] = false));

      function createEnemy() {
```

```
  const x = Math.random() * (canvas.width - enemySize);
  enemies.push({ x: x, y: -enemySize, width: enemySize, height: enemySize, speed: 3 + Math.random() * 3 });
}

setInterval(() => {
  if (!isGameOver) createEnemy();
}, 1000);

function checkCollision(rect1, rect2) {
  return (
    rect1.x < rect2.x + rect2.width &&
    rect1.x + rect1.width > rect2.x &&
    rect1.y < rect2.y + rect2.height &&
    rect1.y + rect1.height > rect2.y
  );
}

function gameLoop() {
  // --- 1. 更新 (Update) ---
  if (!isGameOver) {
    if (keys.ArrowLeft && player.x > 0) player.x -= player.speed;
    if (keys.ArrowRight && player.x < canvas.width - player.width) player.x += player.speed;

    for (let i = enemies.length - 1; i >= 0; i--) {
      const enemy = enemies[i];
      enemy.y += enemy.speed;

      if (checkCollision(player, enemy)) {
        isGameOver = true;
      }

      if (enemy.y > canvas.height) {
        enemies.splice(i, 1);
        score += 10;
        scoreElement.innerText = score;
      }
    }
  }

  // --- 2. 描画 (Draw) ---
  ctx.clearRect(0, 0, canvas.width, canvas.height);

  ctx.fillStyle = '#00aaff';
  ctx.fillRect(player.x, player.y, player.width, player.height);

  ctx.fillStyle = '#ff4444';
  enemies.forEach((enemy) => {
    ctx.fillRect(enemy.x, enemy.y, enemy.width, enemy.height);
  });

  if (isGameOver) {
    ctx.fillStyle = 'rgba(0, 0, 0, 0.6)';
    ctx.fillRect(0, 0, canvas.width, canvas.height);
    ctx.fillStyle = '#fff';
    ctx.font = 'bold 36px sans-serif';
```

```
          ctx.textAlign = 'center';
          ctx.fillText('Game Over', canvas.width / 2, canvas.height / 2 - 20);
          ctx.font = '20px sans-serif';
          ctx.fillText(`Score: ${score}`, canvas.width / 2, canvas.height / 2 + 20);
        }

        requestAnimationFrame(gameLoop);
      }

      gameLoop();
    </script>
  </body>
</html>
```

動作例



ターン2：操作性をよくする

> *プロンプト: スペースキーでゲームをリスタートできるようにしてください。 ゲームオーバー時は停止し、リスタート時にスコアや敵の配列も初期化してください。*

狙い: 何度も遊べるようにして、操作のストレスを減らします。

ここでのポイントは、ゲームオーバー時に停止し、スペースキーで初期状態に戻ることです。AIの出力は異なる形でも構いません。スコアと敵の配列が初期化されることが確認できれば達成です。

生成例コード（ターン2）

以下は、スペースキーでリスタートできるようにした場合の一例です。

ファイル名: `falling-game-turn2.html`

```
<!DOCTYPE html>
<html lang="ja">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>落下物回避ゲーム</title>
    <style>
      body {
        display: flex;
        flex-direction: column;
        align-items: center;
        background-color: #222;
        color: white;
        margin: 0;
        padding-top: 20px;
      }
      canvas {
        border: 2px solid #555;
        background-color: #000;
      }
      .info {
        margin-bottom: 10px;
        font-family: sans-serif;
      }
    </style>
  </head>
  <body>
    <div class="info">Score: <span id="score">0</span></div>
    <canvas id="gameCanvas" width="400" height="600"></canvas>

    <script>
      const canvas = document.getElementById('gameCanvas');
      const ctx = canvas.getContext('2d');
      const scoreElement = document.getElementById('score');

      const player = { x: 180, y: 540, width: 40, height: 40, speed: 7 };
      const enemies = [];
      const enemySize = 30;
      let score = 0;
      let isGameOver = false;

      const keys = {};
      window.addEventListener('keydown', (e) => (keys[e.code] = true));
      window.addEventListener('keyup', (e) => (keys[e.code] = false));

      function createEnemy() {
        const x = Math.random() * (canvas.width - enemySize);
        enemies.push({ x: x, y: -enemySize, width: enemySize, height: enemySize, speed: 3 + Math.random() * 3 });
      }
```

```
setInterval(() => {
  if (!isGameOver) createEnemy();
}, 1000);

function checkCollision(rect1, rect2) {
  return (
    rect1.x < rect2.x + rect2.width &&
    rect1.x + rect1.width > rect2.x &&
    rect1.y < rect2.y + rect2.height &&
    rect1.y + rect1.height > rect2.y
  );
}

function resetGame() {
  score = 0;
  enemies.length = 0;
  isGameOver = false;
  scoreElement.innerText = score;
  player.x = 180;
  player.y = 540;
}

window.addEventListener('keydown', (e) => {
  if (e.code === 'Space' && isGameOver) resetGame();
});

function gameLoop() {
  // --- 1. 更新 (Update) ---
  if (!isGameOver) {
    if (keys.ArrowLeft && player.x > 0) player.x -= player.speed;
    if (keys.ArrowRight && player.x < canvas.width - player.width) player.x += player.speed;

    for (let i = enemies.length - 1; i >= 0; i--) {
      const enemy = enemies[i];
      enemy.y += enemy.speed;

      if (checkCollision(player, enemy)) {
        isGameOver = true;
      }

      if (enemy.y > canvas.height) {
        enemies.splice(i, 1);
        score += 10;
        scoreElement.innerText = score;
      }
    }
  }

  // --- 2. 描画 (Draw) ---
  ctx.clearRect(0, 0, canvas.width, canvas.height);

  ctx.fillStyle = '#00aaff';
  ctx.fillRect(player.x, player.y, player.width, player.height);
```

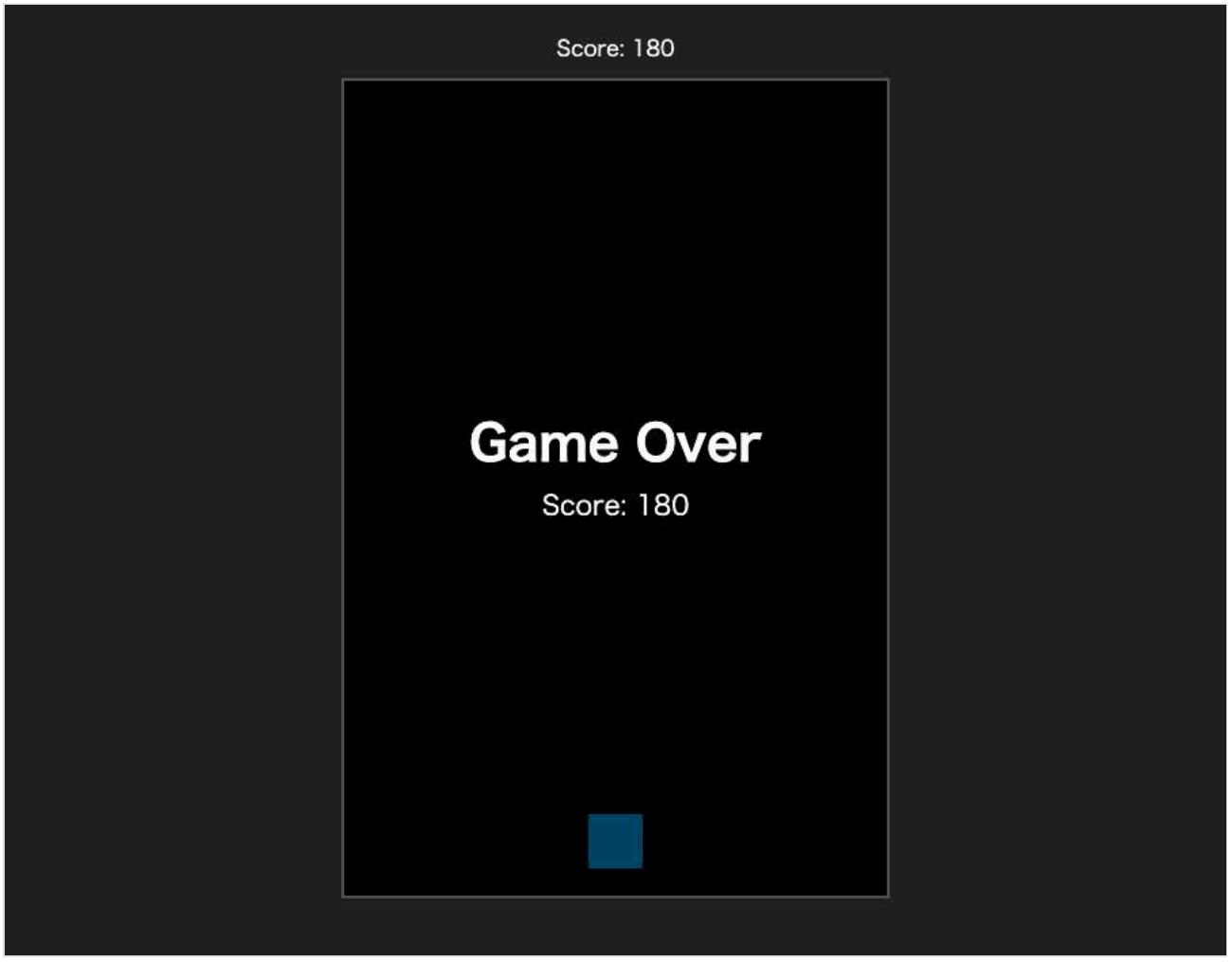
```
        ctx.fillStyle = '#ff4444';
        enemies.forEach((enemy) => {
          ctx.fillRect(enemy.x, enemy.y, enemy.width, enemy.height);
        });

        if (isGameOver) {
          ctx.fillStyle = 'rgba(0, 0, 0, 0.6)';
          ctx.fillRect(0, 0, canvas.width, canvas.height);
          ctx.fillStyle = '#fff';
          ctx.font = 'bold 36px sans-serif';
          ctx.textAlign = 'center';
          ctx.fillText('Game Over', canvas.width / 2, canvas.height / 2 - 20);
          ctx.font = '20px sans-serif';
          ctx.fillText(`Score: ${score}`, canvas.width / 2, canvas.height / 2 + 20);
        }

        requestAnimationFrame(gameLoop);
      }

      gameLoop();
    </script>
  </body>
</html>
```

動作例



ターン3：難易度の上昇を追加する

> *プロンプト: スコアが増えるほど、敵の出現間隔が短くなり、落下速度も少しずつ上がるようにしてください。*

狙い: ゲームらしい緊張感と成長性を加えます。

ここでのポイントは、スコアに応じて難易度が上がることです。出現間隔や速度の計算式は人によって違ってよいので、時間が経つほど難しくなる動きが出ていれば目的は達成されています。

生成例コード（ターン3）

以下は、難易度が上がる仕組みを入れた場合の一例です。

ファイル名: `falling-game-turn3.html`

```
<!DOCTYPE html>
<html lang="ja">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>落下物回避ゲーム</title>
    <style>
      body {
        display: flex;
        flex-direction: column;
        align-items: center;
        background-color: #222;
        color: white;
        margin: 0;
        padding-top: 20px;
      }
      canvas {
        border: 2px solid #555;
        background-color: #000;
      }
      .info {
        margin-bottom: 10px;
        font-family: sans-serif;
      }
    </style>
  </head>
  <body>
    <div class="info">Score: <span id="score">0</span></div>
    <canvas id="gameCanvas" width="400" height="600"></canvas>

    <script>
      const canvas = document.getElementById('gameCanvas');
      const ctx = canvas.getContext('2d');
      const scoreElement = document.getElementById('score');

      const player = { x: 180, y: 540, width: 40, height: 40, speed: 7 };
      const enemies = [];
      const enemySize = 30;
      let score = 0;
      let isGameOver = false;
      let spawnTimerId = null;
```

```
const keys = {};
window.addEventListener('keydown', (e) => (keys[e.code] = true));
window.addEventListener('keyup', (e) => (keys[e.code] = false));

function createEnemy() {
  const x = Math.random() * (canvas.width - enemySize);
  const speed = 3 + score * 0.01;
  enemies.push({ x, y: -enemySize, width: enemySize, height: enemySize, speed });
}

function scheduleEnemySpawn() {
  if (spawnTimerId) clearTimeout(spawnTimerId);
  const interval = Math.max(300, 1000 - score * 2);
  spawnTimerId = setTimeout(() => {
    if (!isGameOver) {
      createEnemy();
      scheduleEnemySpawn();
    }
  }, interval);
}

function checkCollision(rect1, rect2) {
  return (
    rect1.x < rect2.x + rect2.width &&
    rect1.x + rect1.width > rect2.x &&
    rect1.y < rect2.y + rect2.height &&
    rect1.y + rect1.height > rect2.y
  );
}

function resetGame() {
  score = 0;
  enemies.length = 0;
  isGameOver = false;
  scoreElement.innerText = score;
  player.x = 180;
  player.y = 540;
  scheduleEnemySpawn();
}

window.addEventListener('keydown', (e) => {
  if (e.code === 'Space' && isGameOver) resetGame();
});

function gameLoop() {
  // --- 1. 更新 (Update) ---
  if (!isGameOver) {
    if (keys.ArrowLeft && player.x > 0) player.x -= player.speed;
    if (keys.ArrowRight && player.x < canvas.width - player.width) player.x += player.speed;

    for (let i = enemies.length - 1; i >= 0; i--) {
      const enemy = enemies[i];
      enemy.y += enemy.speed;
```

```
            if (checkCollision(player, enemy)) {
              isGameOver = true;
            }

            if (enemy.y > canvas.height) {
              enemies.splice(i, 1);
              score += 10;
              scoreElement.innerText = score;
            }
          }
        }

        // --- 2. 描画 (Draw) ---
        ctx.clearRect(0, 0, canvas.width, canvas.height);

        ctx.fillStyle = '#00aaff';
        ctx.fillRect(player.x, player.y, player.width, player.height);

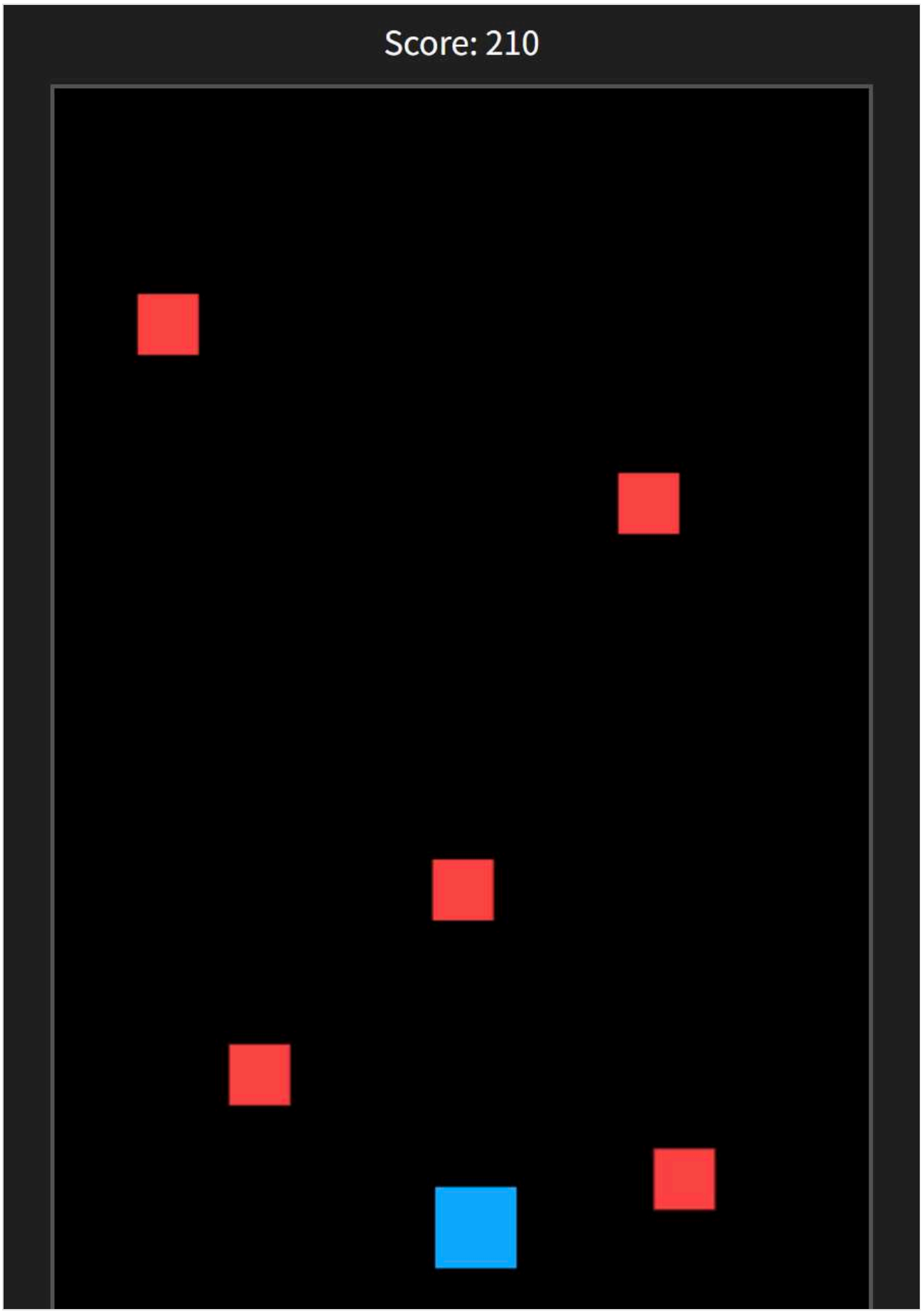
        ctx.fillStyle = '#ff4444';
        enemies.forEach((enemy) => {
          ctx.fillRect(enemy.x, enemy.y, enemy.width, enemy.height);
        });

        if (isGameOver) {
          ctx.fillStyle = 'rgba(0, 0, 0, 0.6)';
          ctx.fillRect(0, 0, canvas.width, canvas.height);
          ctx.fillStyle = '#fff';
          ctx.font = 'bold 36px sans-serif';
          ctx.textAlign = 'center';
          ctx.fillText('Game Over', canvas.width / 2, canvas.height / 2 - 20);
          ctx.font = '20px sans-serif';
          ctx.fillText(`Score: ${score}`, canvas.width / 2, canvas.height / 2 + 20);
        }

        requestAnimationFrame(gameLoop);
      }

      scheduleEnemySpawn();
      gameLoop();
    </script>
  </body>
</html>
```

動作例



> *補足: 追加した要望が反映されたかを確認するために、保存してからブラウザのリロードを忘れずに行ってください。*

AIとの協力ポイント

このように、数学的で少し頭を使うロジックは、AIが最も得意とする分野です。当たり判定を作ってと頼むだけで、AIは最適な数学モデルを選んで実装してくれます。

なぜそのコードで当たりが判定できるのかを知っておくことで、もっとシビアな判定にしたい、判定範囲を少し小さくするといった細かい調整をAIに指示できるようになります。

今回の演習を通じて、AIが書いたコードを一行ずつ読み解き、ゲームが動く裏側にある論理的な仕組みを体感できました。

問題1 ゲームスコアボード（テーブル表示）

チャットで要件を聞き取り、それを満たすWebページを作成してください。

1. 概要

ゲームのスコアボードをテーブル形式で表示するページを作成してください。

2. 要件

1. `<h1>` タグで「スコアボード」を表示
2. `<table>` 内に `<thead>` と `<tbody>` を配置
3. `<thead>` 内の `<th>`：**順位**，**名前**，**スコア**
4. `<tbody>` に以下の3行を表示:
    - 1, Alice, 1500
    - 2, Bob, 1200
    - 3, Charlie, 900

3. 提出

`index.html` ファイルを提出してください。

問題2 クリックカウンターゲーム（インタラクション）

チャットで要件を聞き取り、それを満たすWebページを作成してください。

1. 概要

クリック数を競うゲーム風のページを作成してください。

2. 要件

1. `<h1>` タグで「クリックカウンター」を表示
2. `id="click-btn"` の `<button>`：テキストは「クリック！」
3. `id="score"` の `<span>`：初期値 `0`、ボタンをクリックするたびに1増加

3. 提出

`index.html` ファイルを提出してください。

問題3 HPバー操作（ゲーム状態管理）

チャットで要件を聞き取り、それを満たすWebページを作成してください。

1. 概要

ゲームのHP（ヒットポイント）管理を行うインタラクティブなページを作成してください。

2. 要件

1. `<h1>` タグで「プレイヤーステータス」を表示
2. `id="hp"` の `<span>`：初期値 `100`
3. `id="damage"` の `<button>`：テキスト「ダメージ」、クリックでHP -10（最小0）
4. `id="heal"` の `<button>`：テキスト「回復」、クリックでHP +10（最大100）

3. 提出

`index.html` ファイルを提出してください。

第15章 ブラウザゲームの自由制作

前回の講義では、ゲーム開発の基礎となるゲームループや当たり判定の仕組みを扱いました。四角形が動くだけでも、ゲームの第一歩です。

今回はいよいよ自由制作です。自分のアイデアをAIの力を借りて形にしていきます。プログラミングだけでなく、企画や素材作りまで含めたトータルなクリエイティブ体験に取り組みます。

自由制作

自由制作は、正解のない冒険です。しかし、闇雲に進むと遭難してしまいます。 ゲームを完成させるための3つのステップ、企画、アセット準備、実装と調整に沿って、AIを最大限に活用するコツを解説します。

オリジナルゲームの企画

面白いゲームの定義は一概には言えませんが、作り手として意識すべきポイントはいくつかあります。

既存ジャンルに独自の要素を加える

初心者が陥りがちなのが、壮大すぎるRPGや複雑なシミュレーションゲームを作ろうとして挫折するパターンです。 まずは、パズル、シューティング、アクションといった定番のジャンルを選び、そこにひとつだけ独自の要素を加えるところから始めます。

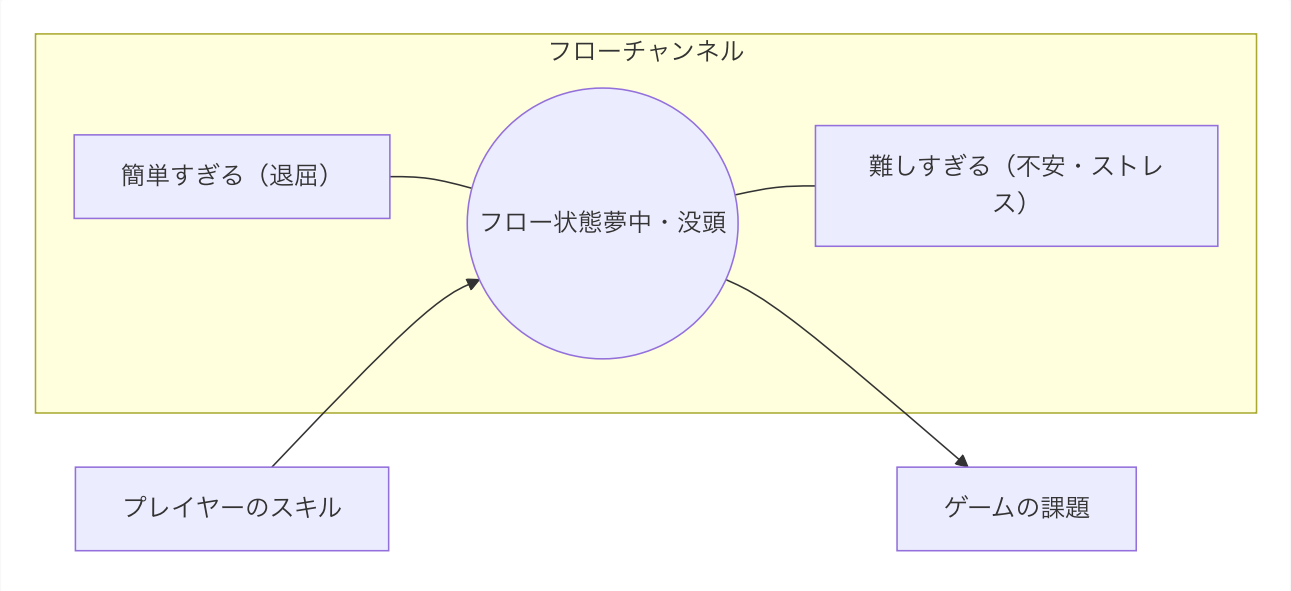
- パズル × 重力: ブロックを消すと、上からではなく横から落ちてくるパズル。
- シューティング × タイピング: 敵に書かれた文字をタイプして攻撃するゲーム。
- アクション × リズム: 音楽のリズムに合わせてジャンプしないと落ちてしまうアクション。

AIにアイデア出しを手伝ってもらうのも有効です。

> *プロンプト例: ブラウザで遊べるシンプルなアクションゲームを作りたいです。 「ブロック崩し」をベースに、何か一つユニークなルールを追加して、斬新なゲームにするアイデアを3つ出してください。*

レベルデザインの重要性

ゲームの面白さを左右するのがレベルデザイン、つまり難易度調整です。 簡単すぎると退屈し、難しすぎるとストレスを感じて投げ出してしまいます。プレイヤーが夢中になるフロー状態を維持するには、このバランスが重要です。



- 序盤: チュートリアルを兼ねて、失敗してもすぐにやり直せる簡単なステージにする。
- 中盤: 敵の数を増やしたり、足場を狭くしたりして、少しずつハードルを上げる。
- 終盤: 習得したテクニックをすべて使わないとクリアできない挑戦的なステージにする。

AIにコードを書かせる際も、「最初は敵のスピードを遅くして、スコアが上がるごとに徐々に速くしてください」と指示することで、このレベルデザインを実装できます。

アセットの準備（AI生成）

前回の講義では、Canvasで描画した四角形をキャラクターとして扱いました。しかし、ゲームの魅力を高めるには、魅力的な絵や音といったアセット、つまり素材が欠かせません。

ここでも生成AIが強力な味方になります。

画像アセットの生成

画像生成AI（DALL-E 3やMidjourneyなど）を使えば、プロのイラストレーターに依頼したような素材が短時間で手に入ります。ゲーム用素材を作るときは、以下のキーワードを含めると効果的です。

- Pixel art: レトロゲーム風のドット絵素材。
- Sprite sheet: キャラクターの歩く、ジャンプするなどの動作を一枚の画像に並べたもの。
- White background: 切り抜きやすくするための白背景指定。

> *プロンプト例: 2Dアクションゲームの主人公キャラクターのドット絵（Pixel art）を描いてください。勇敢な猫の騎士で、正面を向いています。背景は白にしてください。*

画像をゲームに組み込む方法

生成された画像をゲームで使うには、JavaScriptの `Image` オブジェクトを使います。通常は画像ファイルを読み込みますが、コードだけで完結させるために、画像をBase64という文字データとして埋め込む手法を紹介します。

以下のコードは、四角形の代わりにドット絵風のスマイリー画像を表示する例です。

```
<!DOCTYPE html>
<html lang="ja">
  <head>
    <meta charset="UTF-8" />
    <title>画像アセットの描画</title>
    <style>
      canvas {
        border: 1px solid #333;
        background-color: #87ceeb; /* 空のような青 */
      }
      body {
        display: flex;
        justify-content: center;
        align-items: center;
        height: 100vh;
        margin: 0;
      }
    </style>
  </head>
  <body>
    <canvas id="gameCanvas" width="400" height="400"></canvas>

    <script>
      const canvas = document.getElementById('gameCanvas');
      const ctx = canvas.getContext('2d');

      // 1. 画像オブジェクトを作成
      const charaImage = new Image();
```

```
      // 2. 画像のソース（URL）を設定
      // ここでは練習用に、データURIスキームを使って画像を直接コードに埋め込んでいます。
      // 実際には 'player.png' のようなファイルパスを指定します。
      // （この長い文字列は、黄色いスマイリーフェイスの画像データです）
      charaImage.src =

'

      // キャラクターの状態
      const player = { x: 175, y: 175, size: 50 };

      // 3. 画像の読み込みが完了したら描画する
      // ※画像は読み込みに時間がかかるため、onloadイベントを使うのが必須です
      charaImage.onload = function () {
        // ゲームループの開始など
        gameLoop();
      };

      function gameLoop() {
        // 背景クリア
        ctx.clearRect(0, 0, canvas.width, canvas.height);

        // 画像を描画（画像, x, y, 幅, 高さ）
        // fillRectの代わりに drawImage を使う
        ctx.drawImage(charaImage, player.x, player.y, player.size, player.size);

        // シンプルなアニメーション（その場で少し揺れる）
        player.y = 175 + Math.sin(Date.now() / 200) * 10;

        requestAnimationFrame(gameLoop);
      }
    </script>
  </body>
</html>
```

`drawImage` メソッドを使うことで、四角形がキャラクターの絵に変わりました。これだけでゲームの雰囲気が一気に華やかになります。

実装とデバッグ

企画が決まり、素材ができたら、いよいよ実装です。しかし、ゲーム開発には特有の難しさがあります。

ゲーム特有のバグ

Webアプリのバグは「動かない」「エラーが出る」ことが多いですが、ゲームのバグは「動くけれどおかしい」ことがよくあります。

- すり抜け: 移動速度が速すぎて、壁や敵を突き抜けてしまう。
- 埋まり: キャラクターが壁の中にめり込んで動けなくなる。
- フリーズ: 無限ループが発生してブラウザが固まる。

こうしたバグに遭遇したら、現象を詳細にAIへ伝えます。「時々すり抜ける」ではなく、「プレイヤーが最高速度で壁に向かった時だけ突き抜ける」のように伝えると、AIは速度に合わせて判定回数を増やすなどの解決策を提案してくれます。

操作感（ゲームフィール）の調整

バグではないけれど、「なんとなく操作しづらい」「気持ちよくない」という感覚的な問題も重要です。これをゲームフィールやJuicinessと呼びます。

- 慣性: キーを離してもすぐに止まらず、少し滑るようにすると物理的な重みを感じられる。
- ヒットストップ: 敵に当たった瞬間、ほんの一瞬だけ画面を止めることで打撃感を演出する。
- 画面揺れ: 爆発などの衝撃に合わせて画面全体を揺らす。

これらもAIに指示できます。「敵を倒したときに、画面を少し揺らす演出を追加して」と頼んでみてください。

実践：完成形コードの例（クリックゲーム）

最後に、これまで学んだ要素を組み合わせた、小さなゲームの例を見てみます。 逃げるスマイリーをクリックして捕まえるシンプルなゲームです。

```
<!DOCTYPE html>
<html lang="ja">
  <head>
    <meta charset="UTF-8" />
    <title>Catch the Smiley!</title>
    <style>
      body {
        text-align: center;
        background-color: #222;
        color: #fff;
        font-family: sans-serif;
        overflow: hidden;
      }
      canvas {
        border: 4px solid #fff;
        background-color: #333;
        cursor: crosshair;
      }
      h1 {
        margin: 10px 0;
        font-size: 20px;
      }
    </style>
  </head>
  <body>
    <h1>逃げるスマイリーをクリックせよ！ Score: <span id="score">0</span></h1>
    <canvas id="gameCanvas" width="600" height="400"></canvas>

    <script>
      const canvas = document.getElementById('gameCanvas');
      const ctx = canvas.getContext('2d');
      const scoreSpan = document.getElementById('score');

      // 画像アセット（Base64）
      const smileySrc =

'
      const img = new Image();
      img.src = smileySrc;

      // ゲームの状態
      let score = 0;
```

```
const target = {
  x: 300,
  y: 200,
  size: 50,
  dx: 2,
  dy: 2, // 移動速度（Velocity）
  speedMultiplier: 1.0, // 難易度調整用
};

// クリックイベント（当たり判定）
canvas.addEventListener('mousedown', (e) => {
  const rect = canvas.getBoundingClientRect();
  const clickX = e.clientX - rect.left;
  const clickY = e.clientY - rect.top;

  // クリックした位置がターゲットの画像の範囲内か判定
  if (
    clickX >= target.x &&
    clickX <= target.x + target.size &&
    clickY >= target.y &&
    clickY <= target.y + target.size
  ) {
    // 捕まえた時の処理
    score++;
    scoreSpan.textContent = score;

    // 難易度アップ（スピード上昇）
    target.speedMultiplier += 0.2;

    // 位置をランダムにリセット
    target.x = Math.random() * (canvas.width - target.size);
    target.y = Math.random() * (canvas.height - target.size);

    // 方向もランダムに変更
    target.dx = (Math.random() > 0.5 ? 2 : -2) * target.speedMultiplier;
    target.dy = (Math.random() > 0.5 ? 2 : -2) * target.speedMultiplier;

    // 演出（背景を一瞬光らせる）
    canvas.style.backgroundColor = '#555';
    setTimeout(() => (canvas.style.backgroundColor = '#333'), 100);
  }
});

// ゲームループ
function loop() {
  // 背景クリア
  ctx.clearRect(0, 0, canvas.width, canvas.height);

  // ターゲットの移動
  target.x += target.dx;
  target.y += target.dy;

  // 壁での跳ね返り処理
  if (target.x < 0 || target.x > canvas.width - target.size) target.dx *= -1;
  if (target.y < 0 || target.y > canvas.height - target.size) target.dy *= -1;
```

```
        // 画像の描画
        if (img.complete) {
          // 画像の読み込み完了を確認
          ctx.drawImage(img, target.x, target.y, target.size, target.size);
        } else {
          // 画像がまだなら四角を描く（フォールバック）
          ctx.fillStyle = 'yellow';
          ctx.fillRect(target.x, target.y, target.size, target.size);
        }

        requestAnimationFrame(loop);
      }

      loop();
    </script>
  </body>
</html>
```

このコードには、自由制作に必要な要素が詰まっています。

1. 独自のルール: クリックで捕まえるというシンプルなアクション。
2. アセット活用: `Image` オブジェクトによる画像の表示。
3. レベルデザイン: クリックするたびに `speedMultiplier` が増え、敵が速くなって難易度が上がる。
4. ゲームフィール: 捕まえた瞬間に背景色を一瞬変えて、フィードバックを与える。

まずはこのコードをコピーして動かし、「クリックしたら爆発音を鳴らしたい」「敵の種類を増やしたい」など、自分のアイデアを追加して世界に一つだけのゲームに改造してみてください。 AIは「こうしたい」という意図を伝えれば応えてくれる、強力な開発パートナーです。

制作例

自由制作の参考例を紹介します。 あくまで参考ですので、自分自身の興味やニーズに合わせて、オリジナルのアイデアを考えてみてください。

オセロ

オセロ（リバーシ）は、2人で交互に石を置き、相手の石を挟んで自分の色に変えていくボードゲームです。シンプルなルールながら戦略性が高く、AI対戦も楽しめます。

プロンプト例:

```
ブラウザで遊べるオセロゲームを1つのHTMLファイルで作ってください。
- 8x8のボードを表示してください。
- プレイヤーは黒、機械は白の石を使います。
- クリックで石を置き、ルールに従って相手の石を挟んでひっくり返してください。
- ゲーム終了時に勝者を表示してください。
```

完成例:




```
<!doctype html>
<html lang="ja">
  <head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <title>オセロ（8x8）</title>
    <style>
      :root {
        color-scheme: light dark;
        --bg: #0b1020;
        --panel: #121a33;
        --board: #0f6a3b;
        --cell: #0d5f35;
        --line: #0a4e2c;
        --text: #e8edf7;
        --muted: #b8c1d9;
        --accent: #7aa2ff;
      }
      * {
        box-sizing: border-box;
      }
      body {
        margin: 0;
        font-family:
```

```
    'Segoe UI',
    system-ui,
    -apple-system,
    sans-serif;
  background: radial-gradient(1200px 800px at 20% 10%, #162045, var(--bg));
  color: var(--text);
  min-height: 100vh;
  display: grid;
  place-items: center;
  padding: 24px;
}
.app {
  width: min(920px, 100%);
  display: grid;
  gap: 16px;
  grid-template-columns: 1fr;
}
header {
  display: flex;
  flex-wrap: wrap;
  align-items: center;
  gap: 12px 20px;
  justify-content: space-between;
  background: var(--panel);
  padding: 16px 20px;
  border-radius: 16px;
  box-shadow: 0 16px 40px rgba(0, 0, 0, 0.35);
}
header h1 {
  font-size: 20px;
  margin: 0;
  letter-spacing: 0.04em;
}
.status {
  display: flex;
  flex-wrap: wrap;
  gap: 12px 20px;
  align-items: center;
  font-size: 14px;
  color: var(--muted);
}
.status strong {
  color: var(--text);
}
.controls {
  display: flex;
  gap: 8px;
}
button {
  border: none;
  border-radius: 999px;
  padding: 8px 14px;
  cursor: pointer;
  background: var(--accent);
  color: #0b1220;
```

```
  font-weight: 600;
  transition:
    transform 0.15s ease,
    box-shadow 0.15s ease;
}
button:active {
  transform: scale(0.98);
}
button.secondary {
  background: #2a355f;
  color: var(--text);
}
.board-wrap {
  background: var(--panel);
  border-radius: 18px;
  padding: 20px;
  box-shadow: 0 18px 50px rgba(0, 0, 0, 0.4);
  display: grid;
  place-items: center;
}
.board {
  width: min(560px, 90vw);
  aspect-ratio: 1 / 1;
  display: grid;
  grid-template-columns: repeat(8, 1fr);
  border-radius: 12px;
  overflow: hidden;
  background: var(--board);
  border: 2px solid var(--line);
}
.cell {
  position: relative;
  border: 1px solid var(--line);
  background: var(--cell);
  display: grid;
  place-items: center;
  cursor: pointer;
}
.cell.disabled {
  cursor: not-allowed;
}
.cell::after {
  content: '';
  width: 70%;
  height: 70%;
  border-radius: 50%;
  transform: scale(0);
  transition: transform 0.2s ease;
  box-shadow:
    inset 0 0 6px rgba(255, 255, 255, 0.25),
    0 6px 12px rgba(0, 0, 0, 0.35);
}
.cell.black::after {
  background: radial-gradient(circle at 30% 25%, #4b4b4b, #0b0b0b 60%);
  transform: scale(1);
```

```
      }
      .cell.white::after {
        background: radial-gradient(circle at 30% 25%, #ffffff, #c9d4ee 60%);
        transform: scale(1);
      }
      .cell.hint::before {
        content: '';
        position: absolute;
        width: 18%;
        height: 18%;
        border-radius: 50%;
        background: rgba(255, 255, 255, 0.4);
        box-shadow: 0 0 0 2px rgba(0, 0, 0, 0.2);
      }
      .footer {
        background: var(--panel);
        padding: 12px 16px;
        border-radius: 12px;
        font-size: 13px;
        color: var(--muted);
      }
      .footer span {
        color: var(--text);
      }
    </style>
  </head>
  <body>
    <div class="app">
      <header>
        <h1>オセロ（黒：あなた / 白：機械）</h1>
        <div class="status">
          <div>手番：<strong id="turn">黒</strong></div>
          <div>黒：<strong id="blackCount">2</strong> 白：<strong id="whiteCount">2</strong></div>
          <div id="message">あなたの番です。クリックして置いてください。</div>
        </div>
        <div class="controls">
          <select
            id="difficulty"
            class="secondary"
            style="padding:8px 12px;border-radius:999px;background:#2a355f;color:var(--
text);border:none;cursor:pointer;font-weight:600;"
          >
            <option value="1">簡単</option>
            <option value="3" selected>普通</option>
            <option value="5">難しい</option>
          </select>
          <button id="reset">リセット</button>
          <button id="toggleHint" class="secondary">ヒント：ON</button>
        </div>
      </header>

      <div class="board-wrap">
        <div id="board" class="board" role="grid" aria-label="オセロ盤"></div>
      </div>
```

```
  <div class="footer">置ける場所がない場合は自動でパスします。ゲーム終了時に勝者を表示します。</div>
</div>

<script>
  const SIZE = 8;
  const EMPTY = 0;
  const BLACK = 1; // player
  const WHITE = 2; // AI

  const boardEl = document.getElementById('board');
  const turnEl = document.getElementById('turn');
  const messageEl = document.getElementById('message');
  const blackCountEl = document.getElementById('blackCount');
  const whiteCountEl = document.getElementById('whiteCount');
  const resetBtn = document.getElementById('reset');
  const toggleHintBtn = document.getElementById('toggleHint');
  const difficultySelect = document.getElementById('difficulty');

  let board = [];
  let current = BLACK;
  let gameOver = false;
  let showHints = true;
  let aiDepth = 3;

  const directions = [
    [-1, -1],
    [-1, 0],
    [-1, 1],
    [0, -1],
    [0, 1],
    [1, -1],
    [1, 0],
    [1, 1],
  ];

  function initBoard() {
    board = Array.from({ length: SIZE }, () => Array(SIZE).fill(EMPTY));
    const mid = SIZE / 2;
    board[mid - 1][mid - 1] = WHITE;
    board[mid][mid] = WHITE;
    board[mid - 1][mid] = BLACK;
    board[mid][mid - 1] = BLACK;
    current = BLACK;
    gameOver = false;
    messageEl.textContent = 'あなたの番です。クリックして置いてください。';
    renderBoard();
    updateStatus();
    updateHints();
  }

  function renderBoard() {
    boardEl.innerHTML = '';
    for (let r = 0; r < SIZE; r++) {
      for (let c = 0; c < SIZE; c++) {
        const cell = document.createElement('div');
```

```
      cell.className = 'cell';
      if (board[r][c] === BLACK) cell.classList.add('black');
      if (board[r][c] === WHITE) cell.classList.add('white');
      cell.dataset.row = r;
      cell.dataset.col = c;
      cell.addEventListener('click', onCellClick);
      boardEl.appendChild(cell);
    }
  }
}

function onCellClick(e) {
  if (gameOver || current !== BLACK) return;
  const r = Number(e.currentTarget.dataset.row);
  const c = Number(e.currentTarget.dataset.col);
  if (!isValidMove(board, r, c, BLACK)) return;

  applyMove(board, r, c, BLACK);
  current = WHITE;
  switchTurn();
}

function switchTurn() {
  updateStatus();

  if (checkGameEnd()) return;

  if (current === BLACK) {
    const blackMoves = getValidMoves(board, BLACK);
    if (blackMoves.length === 0) {
      current = WHITE;
      messageEl.textContent = 'あなたはパスしました。機械の番です。';
      updateStatus();
      setTimeout(() => {
        aiMove();
      }, 350);
      return;
    }
    messageEl.textContent = 'あなたの番です。クリックして置いてください。';
  } else {
    messageEl.textContent = '機械が考えています…';
    setTimeout(() => {
      aiMove();
    }, 350);
  }
  updateHints();
}

function aiMove() {
  if (gameOver) return;
  const best = findBestMove(board, WHITE, aiDepth);
  if (!best) {
    current = BLACK;
    messageEl.textContent = '機械はパスしました。あなたの番です。';
    updateStatus();
```

```
    if (checkGameEnd()) return;
    updateHints();
    return;
  }

  applyMove(board, best.r, best.c, WHITE, best.flips);
  current = BLACK;
  updateStatus();
  if (checkGameEnd()) return;
  messageEl.textContent = 'あなたの番です。クリックして置いてください。';
  updateHints();
}

function findBestMove(bd, player, depth) {
  const moves = getValidMoves(bd, player);
  if (!moves.length) return null;

  let bestMove = moves[0];
  let bestScore = -Infinity;
  for (const move of moves) {
    const next = cloneBoard(bd);
    applyMove(next, move.r, move.c, player, move.flips, false);
    const score = minimax(next, depth - 1, getOpponent(player), player, -Infinity, Infinity);
    if (score > bestScore) {
      bestScore = score;
      bestMove = move;
    }
  }
  return bestMove;
}

function minimax(bd, depth, turn, maximizingPlayer, alpha, beta) {
  const moves = getValidMoves(bd, turn);
  if (depth === 0 || (!moves.length && !getValidMoves(bd, getOpponent(turn)).length)) {
    return evaluateBoard(bd, maximizingPlayer);
  }

  if (!moves.length) {
    return minimax(bd, depth - 1, getOpponent(turn), maximizingPlayer, alpha, beta);
  }

  const isMax = turn === maximizingPlayer;
  let best = isMax ? -Infinity : Infinity;
  for (const move of moves) {
    const next = cloneBoard(bd);
    applyMove(next, move.r, move.c, turn, move.flips, false);
    const score = minimax(next, depth - 1, getOpponent(turn), maximizingPlayer, alpha, beta);
    if (isMax) {
      best = Math.max(best, score);
      alpha = Math.max(alpha, best);
    } else {
      best = Math.min(best, score);
      beta = Math.min(beta, best);
    }
    if (beta <= alpha) break;
```

```
  }
  return best;
}

function evaluateBoard(bd, player) {
  const opponent = getOpponent(player);
  const weights = [
    [120, -20, 20, 5, 5, 20, -20, 120],
    [-20, -40, -5, -5, -5, -5, -40, -20],
    [20, -5, 15, 3, 3, 15, -5, 20],
    [5, -5, 3, 3, 3, 3, -5, 5],
    [5, -5, 3, 3, 3, 3, -5, 5],
    [20, -5, 15, 3, 3, 15, -5, 20],
    [-20, -40, -5, -5, -5, -5, -40, -20],
    [120, -20, 20, 5, 5, 20, -20, 120],
  ];

  let score = 0;
  for (let r = 0; r < SIZE; r++) {
    for (let c = 0; c < SIZE; c++) {
      if (bd[r][c] === player) score += weights[r][c];
      if (bd[r][c] === opponent) score -= weights[r][c];
    }
  }

  const mobility = getValidMoves(bd, player).length - getValidMoves(bd, opponent).length;
  score += mobility * 4;
  return score;
}

function cloneBoard(bd) {
  return bd.map((row) => row.slice());
}

function getOpponent(player) {
  return player === BLACK ? WHITE : BLACK;
}

function updateStatus() {
  const { black, white } = countPieces(board);
  blackCountEl.textContent = black;
  whiteCountEl.textContent = white;
  turnEl.textContent = current === BLACK ? '黒' : '白';
}

function countPieces(bd) {
  let black = 0;
  let white = 0;
  for (let r = 0; r < SIZE; r++) {
    for (let c = 0; c < SIZE; c++) {
      if (bd[r][c] === BLACK) black++;
      if (bd[r][c] === WHITE) white++;
    }
  }
  return { black, white };
```

```
}

function isValidMove(bd, r, c, player) {
  if (r < 0 || c < 0 || r >= SIZE || c >= SIZE) return false;
  if (bd[r][c] !== EMPTY) return false;
  return getFlips(bd, r, c, player).length > 0;
}

function getValidMoves(bd, player) {
  const moves = [];
  for (let r = 0; r < SIZE; r++) {
    for (let c = 0; c < SIZE; c++) {
      const flips = getFlips(bd, r, c, player);
      if (flips.length) moves.push({ r, c, flips });
    }
  }
  return moves;
}

function getFlips(bd, r, c, player) {
  const opponent = player === BLACK ? WHITE : BLACK;
  const flips = [];
  for (const [dr, dc] of directions) {
    let nr = r + dr;
    let nc = c + dc;
    const line = [];
    while (nr >= 0 && nc >= 0 && nr < SIZE && nc < SIZE && bd[nr][nc] === opponent) {
      line.push([nr, nc]);
      nr += dr;
      nc += dc;
    }
    if (line.length && nr >= 0 && nc >= 0 && nr < SIZE && nc < SIZE && bd[nr][nc] === player) {
      flips.push(...line);
    }
  }
  return flips;
}

function applyMove(bd, r, c, player, flips = null, shouldRender = true) {
  const toFlip = flips ?? getFlips(bd, r, c, player);
  if (!toFlip.length) return false;
  bd[r][c] = player;
  for (const [fr, fc] of toFlip) {
    bd[fr][fc] = player;
  }
  if (shouldRender) renderBoard();
  return true;
}

function updateHints() {
  const cells = boardEl.querySelectorAll('.cell');
  cells.forEach((cell) => {
    cell.classList.remove('hint', 'disabled');
    const r = Number(cell.dataset.row);
    const c = Number(cell.dataset.col);
```

```
    if (gameOver) {
      cell.classList.add('disabled');
      return;
    }
    if (current !== BLACK) {
      cell.classList.add('disabled');
      return;
    }
    if (showHints && isValidMove(board, r, c, BLACK)) {
      cell.classList.add('hint');
    }
  });
}

function checkGameEnd() {
  const blackMoves = getValidMoves(board, BLACK);
  const whiteMoves = getValidMoves(board, WHITE);
  if (blackMoves.length || whiteMoves.length) {
    if (current === BLACK && blackMoves.length === 0) {
      current = WHITE;
      messageEl.textContent = 'あなたはパスしました。機械の番です。';
      updateStatus();
      setTimeout(aiMove, 350);
      return false;
    }
    if (current === WHITE && whiteMoves.length === 0) {
      current = BLACK;
      messageEl.textContent = '機械はパスしました。あなたの番です。';
      updateStatus();
      return false;
    }
    return false;
  }
  gameOver = true;
  updateStatus();
  const { black, white } = countPieces(board);
  if (black > white) {
    messageEl.textContent = `ゲーム終了！ 黒の勝ち（${black} - ${white}）`;
  } else if (white > black) {
    messageEl.textContent = `ゲーム終了！ 白の勝ち（${black} - ${white}）`;
  } else {
    messageEl.textContent = `ゲーム終了！ 引き分け（${black} - ${white}）`;
  }
  updateHints();
  return true;
}

resetBtn.addEventListener('click', initBoard);
toggleHintBtn.addEventListener('click', () => {
  showHints = !showHints;
  toggleHintBtn.textContent = `ヒント: ${showHints ? 'ON' : 'OFF'}`;
  updateHints();
});
difficultySelect.addEventListener('change', (e) => {
  aiDepth = Number(e.target.value);
```

```
        initBoard();
      });

      initBoard();
    </script>
  </body>
</html>
```

成果発表と相互評価

ゲームが完成したら、自分ひとりで遊ぶだけではもったいないです。ソフトウェア、特にゲームは、誰かに使ってもらい楽しんでもらうことで初めて完成したと言えます。

このセクションでは、作った作品を発表し合い、互いに評価することで得られる学びについて解説します。そして、全15回の講義を振り返り、これからのプログラミング学習の道筋を確認します。

ゲーム大会

作ったゲームを持ち寄り、互いにプレイし合うゲーム大会を開催します。これは単なる遊びではなく、開発者として重要な経験になります。

ユーザー視点の獲得

開発中は「ここはこう動くはずだ」という思い込みがかかっています。しかし、初めてプレイする人は、開発者が想定していなかった操作をしたり、ルールの穴を見つけたりします。

- 想定外のプレイ: 「そこをクリックするのか」「その壁、登れると思っていなかった」といった発見があります。
- 難易度の検証: 自分にとっては簡単なステージが、他人にとっては激ムズかもしれません。客観的な難易度を知る良い機会です。

エンターテインメント性の再確認

プログラミングは論理的な作業ですが、その結果生み出されるのは感情を動かす体験です。 自分の作ったゲームで友人が悔しがったり、クリアして喜んだりする姿を見ることは、開発者にとって何よりの報酬です。楽しさを提供するというソフトウェアの本来の目的を、肌で感じられます。

建設的なフィードバック（相互評価）

他人のコードや作品を見て意見を伝え合うことをレビューと呼びます。良いレビューは、双方の成長につながります。

- 良い点を見つける: 「この演出が気持ちいい」「アイデアが斬新」など、良いところを具体的に伝えます。
- 改善の提案: 単に「つまらない」と言うのではなく、「弾の速度をもう少し遅くすると、もっと避けやすくて楽しくなるかも」のように、建設的な提案をします。

> *やってみよう: もし周りに同じ学習者がいなければ、家族や友人にプレイしてもらいます。あるいは、GitHub Pagesなどの無料ホスティングサービスを使ってWeb上に公開し、SNSで感想を募るのも現代的な発表の形です。*

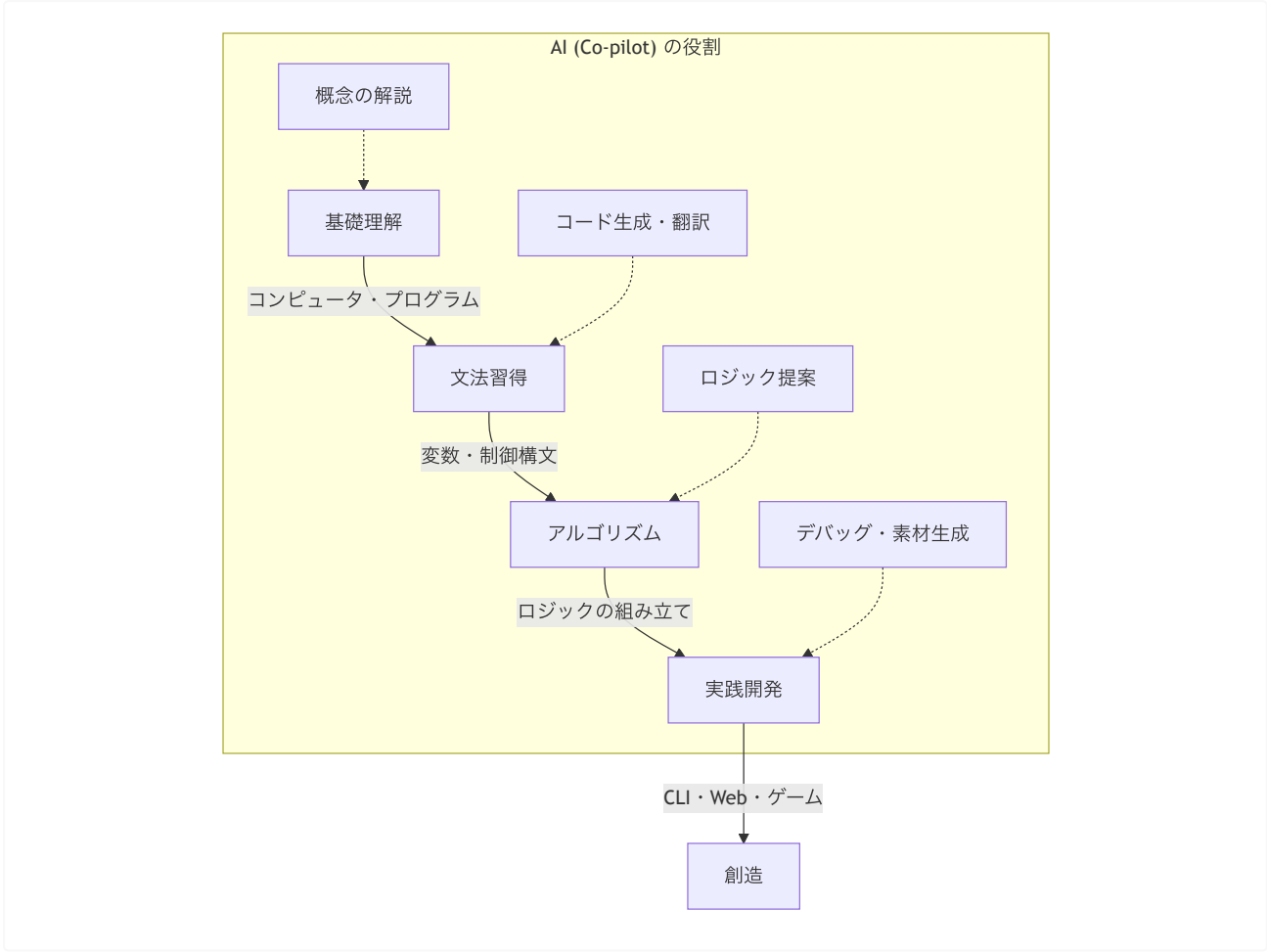
最終振り返り

全15回の講義はこれで終了です。 この講義を通じて何を学び、どのような力を手に入れたのかを振り返ります。

全15回の軌跡：ゼロから創造へ

最初はコンピュータとは何かという座学から始まりました。そこから、プログラミングの基礎文法、アルゴリズム、データ構造といったコンピュータサイエンスの核心に触れ、環境構築を経て、実際に動くアプリケーションを作るところまで到達しました。

以下の図は、この講義で辿ってきた学習の道のりです。



かつてはターミナルの黒い画面が怖かったかもしれません。英語の羅列であるコードが呪文に見えたかもしれません。しかし今、自分のアイデアを形にするための道具として、それらを使いこなす入り口に立っています。

AIを相棒とするマインドセット

この講義の最大のテーマは、生成AIと共に開発するバイブコーディングでした。

従来のプログラミング学習は、文法を丸暗記することに多くの時間を費やしていました。しかし、これからの時代に重要なのは以下のスキルです。

1. 言語化する力、つまりプロンプトエンジニアリング: 何を作りたいか、どう動いてほしいかを、論理的かつ具体的にAIに伝える力です。
2. 読み解く力、つまりコードリーディング: AIが書いたコードが正しいかを判断し、バグがあったときにエラーメッセージから原因を推測する基礎知識です。
3. 全体を設計する力: 細かいコードの記述はAIに任せ、どの部品をどう組み合わせればシステム全体が動くかを考える構成力です。

AIは魔法の杖ではなく、優秀な副操縦士です。機長は自分自身です。方向を示さなければ、飛行機は飛びません。

これからの学習

プログラミングの世界は広大で、この講義で扱った内容はほんの一部に過ぎません。しかし、AIと協力すれば何でも作れるという実感を持てたなら、次のステップに進む準備は整っています。

今後、さらに深く学びたい場合の道しるべをいくつか示します。

- フレームワーク: ReactやVue.js、Unityなど、本格的な開発を効率化するツールを学ぶ。
- データベース: ユーザーのハイスコアを永続的に保存したり、ログイン機能を作ったりするために、SQLやNoSQLを学ぶ。
- 公開: 作ったアプリを世界中の人が使えるように、AWSやVercelなどのクラウドサービスへのデプロイ方法を学ぶ。

作りたいものができた時が、一番の学び時です。 その時はまた、エディタを開き、AIに話しかけてみてください。

「こんなアプリを作りたいんだけど、どこから始めたらいい？」

きっと、頼もしい相棒が答えを返してくれるはずです。

問題1 ブラウザゲーム自由制作

ブラウザで動作するゲームを自由に制作してください。 `index.html` とその表示に必要なファイル一式を提出してください。

制約

- ページは `index.html` のみとする。
- HTMLファイルを直接ブラウザで開いたときに正しく表示・動作するように制作すること。