コードの解析と言語習得の心得

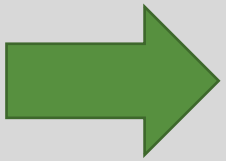
～見て学び、知って使う～

自己紹介~Introduction~

- 奈良高専電気工学科に所属
- 電気系情報発信サイト“JinProduction”の運営者 兼 ライター
  ⇒現“JinProductionDiary”、来年よりリニューアル
- 電気技術研究会という同好会の部長
  奈良高専祭電気科展の代表者

講演目的

- 読む能力を身に付け、自分のプログラムをデバッグする力を身に付ける
- プログラムを書く+読むことで、より良い問題解決能力を得る
- 読む力を駆使し、言語の学習をより効率的に行えるようになる



読むことによって知識を付け、書くことによってそれを活かす力

※別スライド"知識と技術"参照

目的達成のために

- 解析と可読性の2パートで読む＋書く力について解説
- 言語の学習法を最後に組み入れ、理解を深める
- 筆者作成のアプリを用いた具体的手順を解説

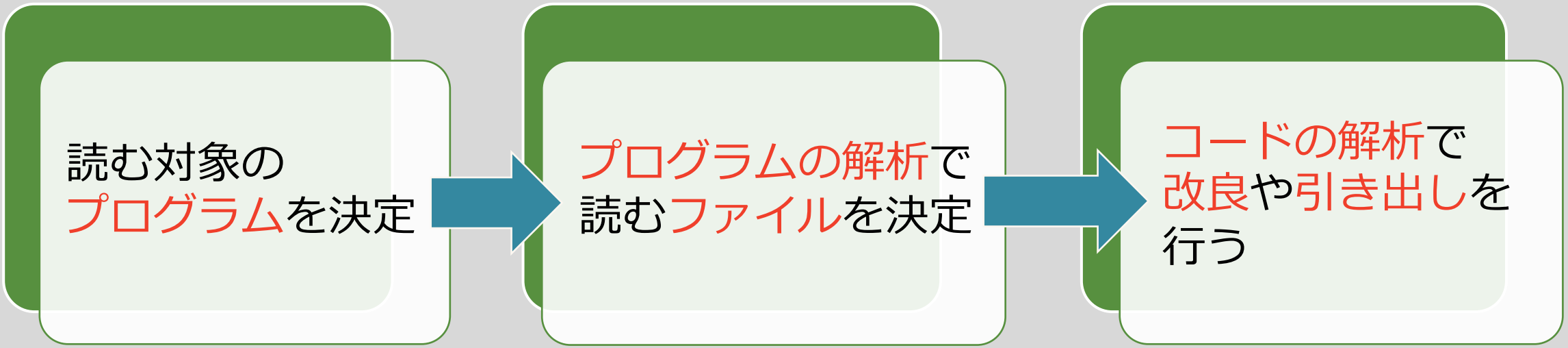
プログラミングの解析

～処理を見極める～

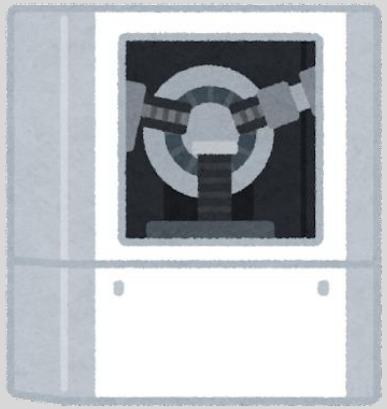
プログラミングを読む目的

- オープンソースのプロジェクトを使用する機会が増えると…
  様々な場面で、ソースコードの解析が必須になる
- 必要最低限の情報を抜き出し、応用する技術が必要
- 自分の書いたプログラムをデバッグするときなんかも…

プログラミングを読む手順

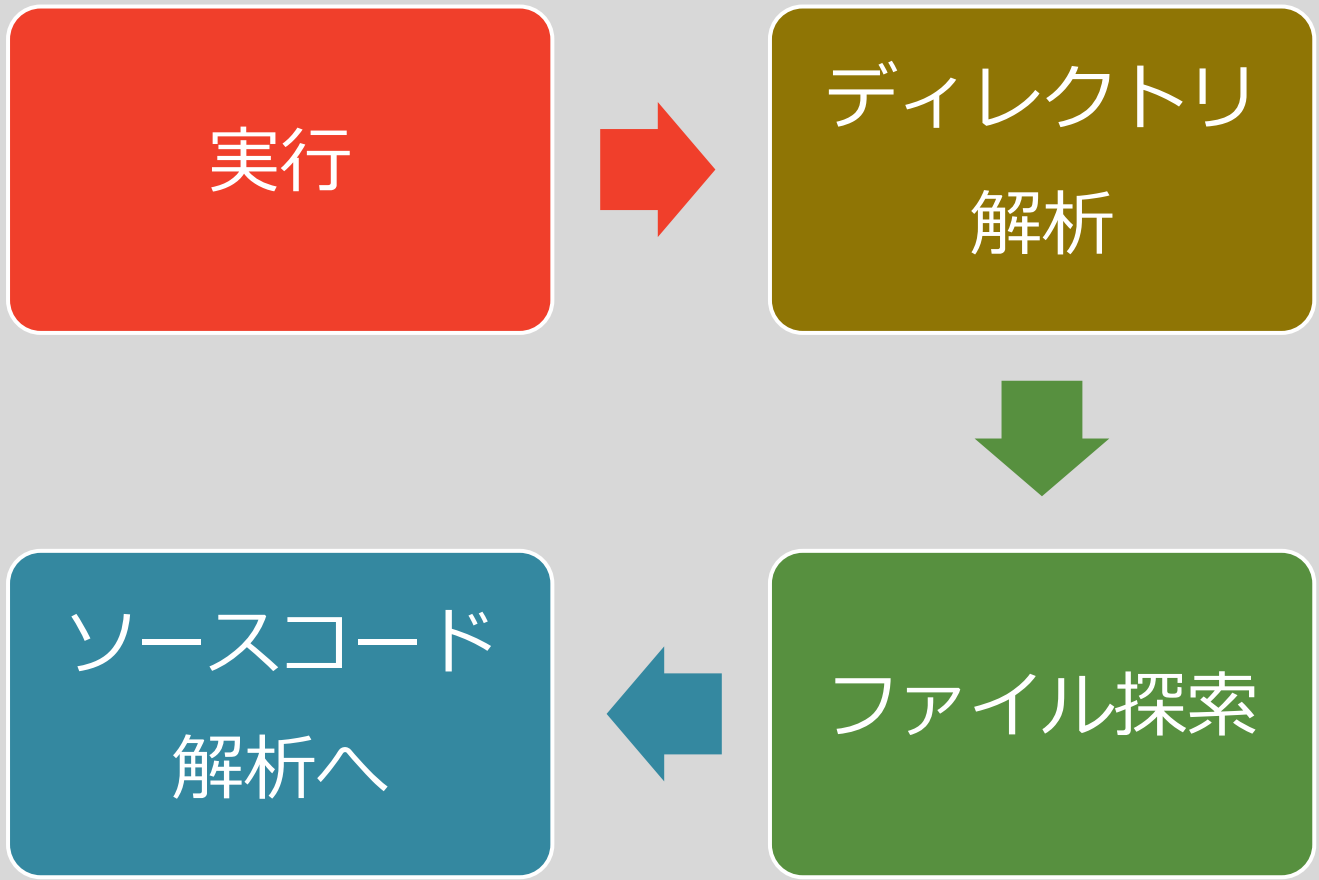


プログラム解析



- プログラムとは
  ソースコードという言語ファイルをひとまとめにしたもの
- ソースコードを解析するには、プログラムの構造を知っておく必要が
- ファイル名やディレクトリ名などから、ヒントを得る必要がある

プログラム解析の手順
～目当てのファイルはどこに？～



ディレクトリ解析

- 大きなプログラムは、多くのディレクトリが存在
- ディレクトリ名から大まかに予測

  例えば…典型的なウェブページではassetsフォルダやsrcフォルダなど
- ディレクトリの構造を読み取り、目的に合わせたファイルの探索に

ファイル探索

- 解析したディレクトリに入り、ファイルを探っていく
- 拡張子を確認し、ファイルの種類を把握
  例えば…C#は.cs、実行ファイルは.exeなど
- ファイルを見つけたら、まずはデバッグを行う！

ソースコード解析

- プログラミングの解析における醍醐味

- ソースコードを解析するには、以下の手段が存在
  - 動的解析
    デバッグを用いる方法であり、変数の値などを随時確認できる

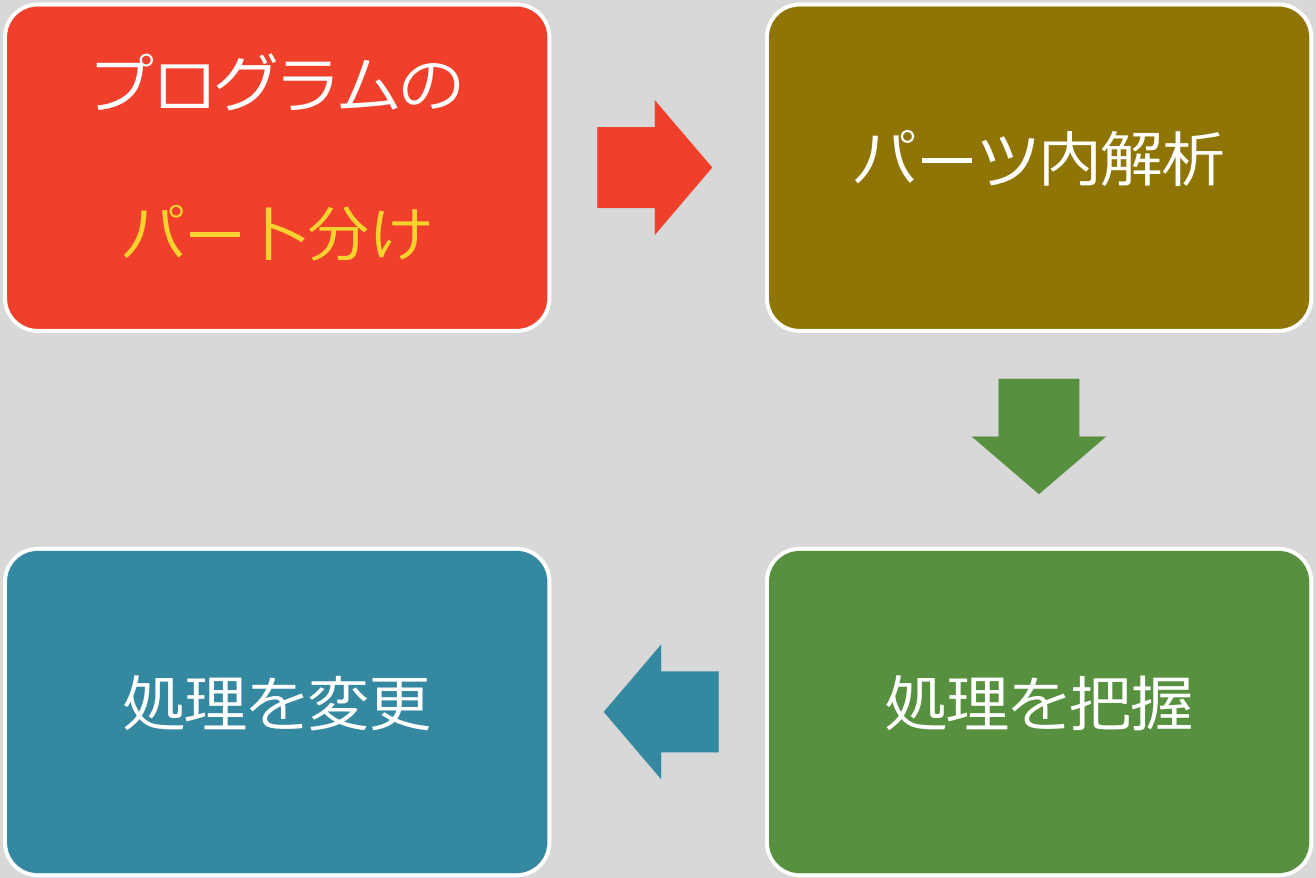
  - 静的解析
    手動で読み解く方法であり、変数の値などは完全に予測する

ソースコードの動的解析
～デバッグ～

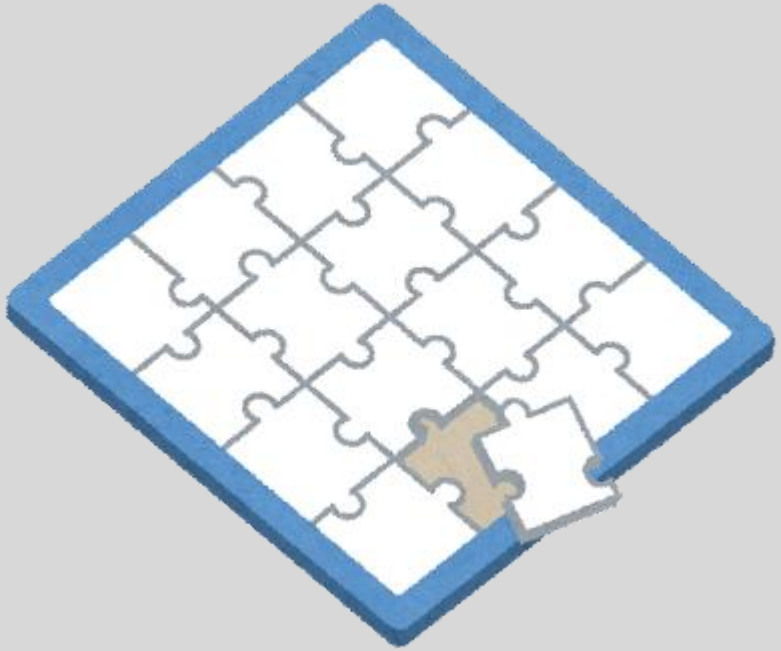
- コードを実行し、任意の場所を観測することで解析する方法

- デバッガーを使用する方法
  VisualStudioなどのデバッグツールを用いて
  コードを改変せずに必要な情報を得る
- デバッガーを使用しない方法
  printfやconsoleなどの標準関数を確認する場所に挿入する

ソースコード静的解析の手順



パート分け

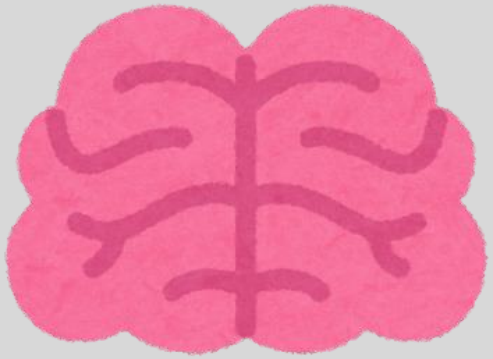


- プログラムとは大きな処理の流れのようなもの
- 膨大なプログラムを読むには？？

  ⇒書くとき同様、論理的思考を活用しよう！！
- ソースコードを解析するうえでも、論理的思考は必須スキル

論理的思考（ロジカルシンキング）とは？

- ものごとを体系的に整理し、道筋を立てて考える思考法
- 問題解決の際、原因特定や解決策の立案に役に立つ
- ソースコードを読む場合、処理の流れを論理的に読み解く必要がある



解析への活用

- ソースコード内の全要素を確認し、道筋を立てて大きな処理の流れを把握
  ⇒コメント文を参考にして予測したり
  必要に応じて動的解析を用いて確認する

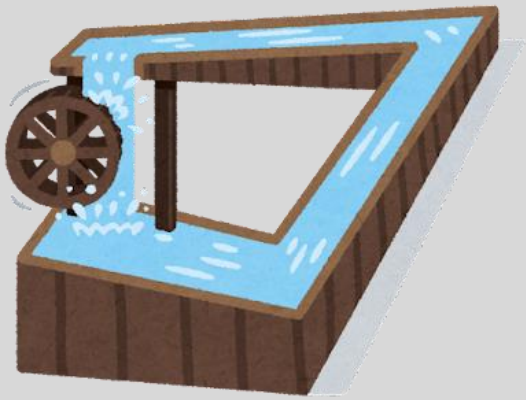
- 処理の流れを確認したら
  パズルのように処理をまとめてパート分け

- 分けたパートごとに細かく解析を行う

パーツ内解析

- 分けたパーツを確認し、任意のパーツを絞り出す
- 絞り出したパーツで、変数や関数の役割を予測したり
  動的な解析を用いてひとつずつ動作を確認したりする
- 一つ一つの変数や関数を、パーツの中のパーツとみる

処理の流れを把握



- １つ目の手順でしたように
  次は絞ったパーツで小さな処理の流れを把握

- 確認した役割のパーツをつなぎ合わせ、パズルを完成させる

- コメント文を参考にして予測したり
  必要に応じて動的解析を行ったりして動作を確認する

処理改良の手順

- 処理の改良にも、論理的思考を用いた手順が存在

1. 原因解明
   区切ったパーツで、変更すべき構文や変数、処理を確認する

2. 解決策の立案
   どう変更するか、また新しくどのような処理を挿入するか、考案する

プログラミング解析のまとめ

- パーツ分け、パーツ内解析、処理の把握、改変を繰り返し、
  目的の動作へ近づけていく
- 論理的思考を用いた道筋に基づき処理の流れを理解する
- 動作や処理の予測と、実際に動かしたときの確認がキーポイント
- 大きなソースコードを、小さく区切って考えていく

コードの可読性

～読みやすいコードのために～

より良い可読性を求めて

- プログラミングを読みやすくするために可読性という概念が存在

- 可読性は大まかに以下の2つに分かれる
  - ソースコード内の記述
  - 論理的思考を用いた処理の流れ

命名法

- 変数や関数を適切に命名すると、誰にでも読みやすくなる

- 変数名は最初を小文字、関数名は大文字にする…
  など、ルールを決めて記述しているとより分かりやすく

- 名前自体は変数の役割を、より分かりやすく命名する必要がある
  例えば…カウントする変数：cnt、一時的な変数：tmp
  以前の値を格納：pre~、など…

演算子の選択

- 演算において、右辺に変数が来る場合
  複合代入演算子を用いると、よりコードが見やすくなる
  ⇒演算の章を参照

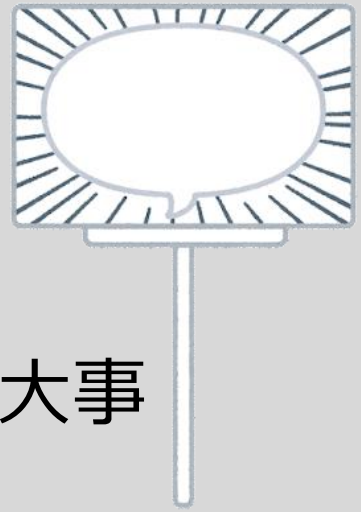
- if構文を使わなくても、3項演算子で解決する場合
  積極的に用いると、簡潔なコードで見やすくなる

- ifネストを避けるときには、論理演算子を用いると簡潔に

インデント

- プログラムにおいて、字下げ（インデント）は非常に重要な要素
- 基本的には、中括弧は4字or2字下げる
  ネストが深くなると、字下げが多く読みにくくなる場合もある
- インデントで関数の区切りなどを決めている言語もある
  （Pythonなど）

コメント文

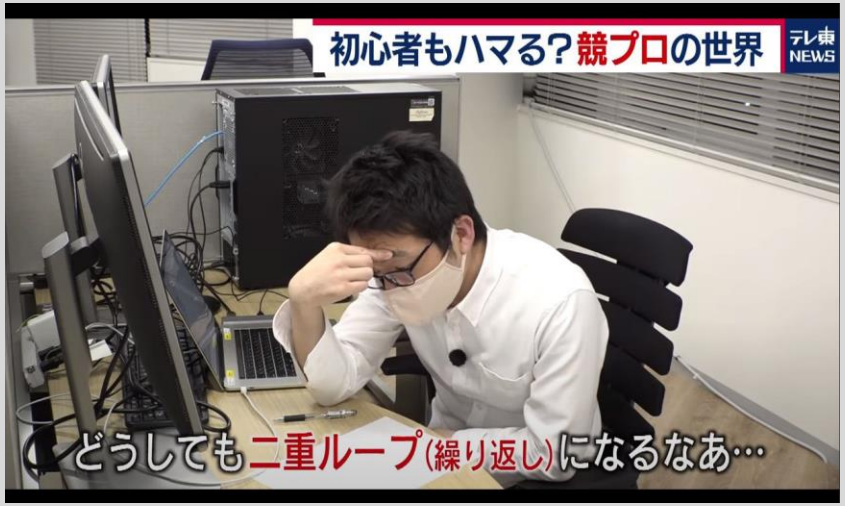


- プログラムを書くと、随時その処理の役割を
  コメントで注釈することが大事

- コメントは完結に要点だけを書く、抽象的なもので可
  具体的に書きすぎると長くなり、かえって可読性が悪くなってしまう…

- 注釈がないと、可読性だけではなく、自分でも忘れてしまうことが…

ネストの回避



- ネストはなぜ使う？
  - ifのネストでは複雑な条件分岐が可能
  - 多重ループでは多次元配列の処理が楽に
- ネストの問題
  - 複雑すぎると、記述と処理両方の可読性が悪くなる…
- 解決策
  - ifネストでは条件文に論理演算子を用いたり、
  - 多重ループを回避するには、配列の次元を減らすなど

関数やクラス分け

処理をメインルーチンに詰め込むと…

どこからどこまでが、何の処理の範囲か不明になる

そこで…

⇒適切に関数やクラスを分けることが重要に

関数に処理をまとめると、ほかの関数からも参照が可能になる

⇒構造化プログラミングの反復処理にあたり、可読性が改善

配列やオブジェクト

同じ系統の変数を大量に作成すると…

多すぎる変数に対し、処理の役割が見えにくい！

そこで…

処理を関数にまとめるように

変数も配列やオブジェクトにまとめることが重要！！

ファイル分け

ファイル名と全く別の処理が記述されていると…

処理だけではなく、ファイルの解析も困難になる

そこで…

別ファイルやライブラリの自作によって

処理の混同を避けることを意識しよう！ ！

ライブラリの多用

「ちょっとねぇ、僕はちょっと怒ってます」おぢさんは間違っている！！

複雑な処理を頑張って自作して追加しても…

　　重要な処理に注目できず、手間や時間が取られる

そこで…

　　先人に感謝し、ライブラリを使わせてもらおう！！！

　　⇒ライブラリ自体の理解は最低限必要

コードの可読性のまとめ

- コードを読みやすくするためには、可読性を意識する必要がある
- 記述の可読性

  命名や演算子、インデント、コメント、ネストなどに

  注意して記述する必要がある
- 処理の可読性

  関数やクラス、配列、オブジェクト、ファイル分けやライブラリなどに

  注意して記述する必要がある

言語の学習法

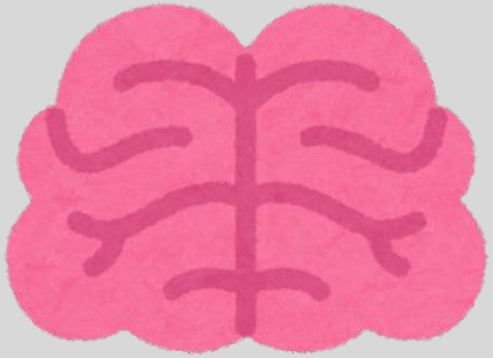
～いろんなところに適応しよう～

言語習得の手順

新しく言語を学ぶときの手順

1. 記述方法（インデントの決まりや文末のセミコロンなど）を知る
2. 基本の骨格（制御構文やデータ型など）について調べ、理解する
3. 既存のコードを解析したり、独自でコードを記述することにより習得

論理的思考と問題解決能力



- 言語を学ぶときに最重要なのは…
  論理的思考である

- 言語の文法を身に付けるのは非常に簡単だが…
  初めての言語では、慣れないエラーにてこずる可能性も

論理的思考と問題解決能力

- エラーに出くわした時、論理的思考を最大限に活用しよう
  - 対象のプログラムに対し、プログラミング解析をかける
  - 解析した結果、発見した問題に対し、解決策を立案
  - 可読性を意識したうえで、コードを再び記述、デバッグの繰り返し

論理的思考と問題解決能力

- エラー対処だけではなく…
  ものづくりをする際に非常に重要になるのはエラー処理…？？？
  ⇒間違い
- もしエラー無くコードが実現しても
  本質の課題を解決できてなければ意味がない！！

つまり…

課題の本質理解と、それに沿った真の課題解決が必要！

言語を習得するには

基本の知識を身に付けたうえで、プログラミング解析を繰り返す
⇒動的なデバッグで処理の確認をすることで
だんだんと予測が可能になる

課題解決の予測を立て、コーディングして確認して言語習得が完成

これは独学による、課題解決能力を駆使した
最も効率的な言語習得法であり、本質理解につながる

言語の習得法のまとめ

- 記述方法、基本の骨格を知り
  それを活かしてプログラミング解析を重ねて言語習得をする

- モノづくりではエラーを対処するだけでなく
  課題の本質理解から真の問題解決を実現することが重要

- 問題解決を繰り返すことで予測と確認の手順が身に付き
  最も効率の良い言語習得法が完成する

最後に

～全体のまとめ～

プログラミングを読むこと

- プログラミング解析の手順
  プログラムの解析⇒ソースコードの解析

- 読み、処理の流れを理解することで改良し、問題解決につなげる
  ⇒論理的思考の最大活用によって効率が上がる

プログラミングを書くこと

- コードを書くとき、可読性を意識したコーディングが重要に

- 記述の面では見やすいコードを、
  処理の面では読みやすいコードを心がける

プログラミングを学ぶこと

- 基本的な知識を活用し、プログラミング解析を用いて学習する
- エラーの対処だけではなく、問題自体の解決を心がける
- 予測と確認の手順で、言語を本質から身に付けよう！！

問題解決能力を駆使したものづくり

- 読んで学び、知って使う手順はプログラミングだけか…

⇒全くの誤解です！

- 電気や数学などの理数分野や
  環境問題や政治問題などの社会など
  広くにわたって応用が可能！

- ぜひ、論理的思考を身に付け、課題解決に取り組んでください！

ご清聴
ありがとうございました

これでこの講義は終了です。お疲れさまでした。