python_catchup.md

前提

- 「完璧な理解」よりも 「**まず全体像を掴み、動くコードを読んで理解できる**」ことを最優先に
- ロードマップに沿って進めれば以下のことを想定
  - 2週間程度でPythonコードを読み書きする基礎体力と作法が身に付く
  - 自走できるレベルに到達できる

最短キャッチアップ・ロードマップ（目安：1〜2週間）

このロードマップは3つのステップで構成されています。

1. **ステップ1：超速・基礎文法マスター（1〜3日）**
   - **目標:** とにかく「書ける」ようになる。細かい理屈は後回しでOK。
2. **ステップ2：「読める」レベルへの引き上げ（3〜5日）**
   - **目標:** 他人の書いたコードで頻出する、Pythonらしい書き方を理解する。
3. **ステップ3：実践力と「作法」の習得（1週間〜）**
   - **目標:** 綺麗なコードの書き方を学び、簡単なツールを自作・改造できるレベルになる。

ステップ1：超速・基礎文法マスター（1〜3日）

ここでは、コードを読むための最低限の部品を頭に入れます。完璧に覚えようとせず、「こんなものがあるんだな」くらいの感覚で進めましょう。

学習項目：

1. **環境構築:**
   - PCにPythonをインストールする。(公式サイトから最新版をダウンロード)
   - VSCodeなどのテキストエディタにPython拡張機能を入れる。
2. **基本の「型」と「変数」:**
   - `print()` で文字や数値を表示する。
   - 変数: `my_variable = "こんにちは"`
   - データ型: **文字列 (str)**, **整数 (int)**, **浮動小数点数 (float)**, **真偽値 (bool)** の4つだけまず覚える。
3. **基本的な制御構文:**

- **if, elif, else**: 条件分岐。「もしAなら〜、そうでなくBなら〜、それ以外は〜」
- **for ループ**: 繰り返し処理。「リストの各要素に対して〜」「10回繰り返す」
  - `for item in my_list:`
  - `for i in range(10):`
- **while ループ**: 条件が満たされている間の繰り返し。

4. **基本的な関数:**
   - `def my_function(argument):` で関数を定義する方法と、`return` で値を返すことだけ理解する。

このステップでのコツ:

- **手を動かす:** 必ず自分でコードを書いて実行してください。コピペでもOK。
- **エラーを恐れない:** エラーメッセージを読む練習にもなります。
- **おすすめ教材:** Progate や Paizaラーニング の無料部分を高速で終わらせるのが効率的です。

ステップ2：「読める」レベルへの引き上げ（3〜5日）

ここが**コード読解力向上のキモ**です。Pythonのコードで非常によく使われるデータ構造と、Python特有の効率的な書き方を学びます。

学習項目：

1. **主要なデータ構造（超重要）:**
   - **list (リスト)**: `[]`
     - 複数の要素を順番に格納。 `my_list[0]` でアクセス。 `my_list.append(item)` で追加。
   - **dict (辞書)**: `{}`
     - キーと値のペアで格納。 `my_dict['key']` でアクセス。JSONと似ている。
   - **tuple (タプル)**: `()`
     - リストに似ているが、**変更不可能 (immutable)**。関数の戻り値などでよく使われる。
   - **set (集合)**: `{}`
     - 重複しない要素を格納。順序は気にしない。
2. **Pythonらしい書き方 (Pythonic Way):**
   - **リスト内包表記 (List Comprehensions):**

        - forループを1行で書く方法。他人のコードでは**頻出**します。
        - 例: `new_list = [i * 2 for i in old_list if i > 0]`
    - **f-string (フォーマット済み文字列リテラル):**
        - 文字列の中に変数を埋め込む、最もモダンで簡単な方法。
        - 例: `name = "Taro"`, `print(f"こんにちは、{name}さん")`
    - **ファイルの読み書き:**
        - `with open('file.txt', 'r') as f:` という書き方を覚える。これが安全な定石。
3. **モジュールのインポート:**
    - `import os` や `from datetime import datetime` のように、他の人が作った便利な機能（モジュール）を読み込む方法を理解する。

このステップでのコツ:

- **比較する:** 例えば、同じ処理を `for` ループで書いた場合とリスト内包表記で書いた場合を見比べて、後者の簡潔さを実感してください。
- **小さなコードを読む:** GitHubなどで短いPythonスクリプトを探し、ここで学んだ知識（`list`, `dict`, `for`, `if`など）がどう使われているかを確認します。

ステップ3：実践力と「作法」の習得（1週間〜）

最後に、コードの品質を高めるための「作法」と、より複雑なコードを読むための知識を身につけます。

学習項目：

1. **PEP 8 (Pythonの公式スタイルガイド):**
    - これが**「基礎的な作法」**の答えです。全部覚える必要はありません。
    - **絶対に押さえるべき点:**
        - インデントは**スペース4つ**。
        - 1行の長さは79文字（または99文字）程度に。
        - 変数名や関数名は `snake_case` (例: `my_variable`)。
        - クラス名は `CamelCase` (例: `MyClass`)。
    - **対策:** `black` (自動フォーマッタ) や `flake8` (リンター) というツールをエディタに導入すれば、自動で準拠できます。
2. **仮想環境 (venv):**
    - プロジェクトごとにPythonの環境を分離する仕組み。「Aの案件ではライブラリXのv1.0を、Bの案件ではv2.0を使う」といった状況で必須。

    - `python -m venv myenv` のように作ることを知っておくだけでOK。
3. **パッケージ管理 (pip):**
    - `pip install requests` のように外部ライブラリをインストールする方法。
    - `requirements.txt` に使っているライブラリを記述する、という慣習を理解する。
4. **クラスの基礎 (読解のため):**
    - 自分で複雑なクラスを作る必要はありません。
    - `class MyClass:` という定義、`__init__` (初期化メソッド)、`self` の意味、`object.method()` (オブジェクトのメソッドを呼び出す) という構文が読めれば十分です。
5. **例外処理 (try...except):**
    - エラーが起きそうな処理を囲み、エラー発生時の代替処理を書く方法。堅牢なコードには必須。

このステップでのコツ:

- **人のコードを積極的に読む:** GitHubで興味のある分野の小〜中規模なプロジェクトのコードを読んでみましょう。最初は分からなくても、「この関数は何をしているんだろう？」「この変数の型は何だろう？」と推測する訓練が重要です。
- **簡単なツールを作る:** 「特定のフォルダにある画像をリサイズする」「Webサイトから特定の情報を取ってくる」など、自分のための小さな自動化ツールを作ってみるのが一番の近道です。

おすすめの教材・リソース

- **書籍:**
    - **『退屈なことはPythonにやらせよう』:** 実用的な例が多く、ステップ2以降の学習に最適。
    - **『Python実践入門』:** より深く、Pythonらしい書き方を学べる良書。
- **Webサイト:**
    - **Python公式チュートリアル:** 辞書的に使うのに便利。
    - **Real Python:** 質の高い記事が多い（英語）。
- **ツール:**
    - **Visual Studio Code (VSCode):** Python拡張機能、Pylance、black、flake8を入れると最強の開発環境になります。