1　コード規約

プログラムを書いて機能を実装するだけというのは、内容が短くかつ一人で管理する際には対応できるかもしれないが、複数人長期的に保守・運用するときには、コードを書くだけでなく、他の人にも読みやすいコードを書く配慮が重要になってくると考える。そのなかでも、コーディング規約を守って書くということは、その見やすさの客観的な指標になりうると思うので、1 つ目のトピックとしては、「コーディング規約」を取り上げる。

1.1　コードフォーマット

講義を受ける前にもの位置や演算子の前後には空白を入れるといった、いくつかのコード規約は学んだこともあり、知っていることもあったが、この講義を通してまだ学べていなかったコード規約についても学ぶことができた。

実際に、講義内で学んだ内容をあげると、引数が複数行に渡るときのインデントの書き方や、演算子の前で改行するか演算子の後で改行するかといった議論は納得するところもあり、曖昧にしてしまっていた部分でもあったので勉強になった。

1.2　コードチェッカー・フォーマッタ

また、コードのチェッカーやフォーマッタを VScode 内に導入していなかったこともあり、VScode 内に flake8 と autopep を導入して設定を行った上で、自動保存によりコードのフォーマットを整えられるようにした。これまでは、コードフォーマットを整えようと意識している反面、全てに気を配ることは難しく整えきれていないコードも混在してしまって、統一感が薄れてしまう部分もあったが、実際にこれらの拡張機能を追加し活用していくことで、読みやすいコードを意識しつつ統一性のあるコードを書けるようになっていると感じる。

1.3　実践

次は、実際に今回のレポート用にいくつかの関数を作成し、それらを PEP8 のコード規約に従うように実践したのでそれについてまとめる。これらは講義の演習だけではなく、実際に各トピックの実践を行うことでより活用できるように考えている。

下記のコードでは、後述する github 上に tips.py としてあげている私がよく使うコー

ドを関数化したものである。コード規約にのっとって考えると、演算子の前後に空白を加えることだったり、”,”の後に空白を加えることは注意して書くべきところであるが注意して書き、コードフォーマッタで、最終的なチェックを行った。

下記のプログラムで実装した関数は以下の 3 つ。

- current_date：現在の日付を出力する関数
- read_csv：csv を読み込む関数 (.csv の省略)
- to_csv_date：csv を書き出す関数 (日付つきのファイル名)

下記の例では、コード量が少ないこともあり、PEP8 のコード規約を十分に実践できたわけではないが、他のプログラムも書いていく過程で複数行に渡るコードを書く場合には曖昧にしてしまうと迷ってしまうことも多い。そのような中で PEP8 の規則を学びそれらを実践することで統一性があり、可読性の高いコードを書くことができると感じているので今回はその練習として実践できてよかったと感じる。また、学びきれなかったコード規則もあると思うので、勉強していきたいと感じるとともに、プロジェクトの開発チームごとにも従うべきコード規則というものがそれぞれあると感じるので、基礎を整えた上でそれらにも柔軟に対応できるようにしていきたいと思う。

Listing 1　tips.py

```
1  import pandas as pd
2  import datetime
3
4  # 現在の日付を出力する関数
5  def current_date():
6      dt_now = datetime.datetime.now()
7      dt_now_format = dt_now.strftime('%Y%m%d')
8      return dt_now_format
9
10 # csv を読み込む関数(.csv の省略)
11 def read_csv(df_name):
12     return pd.read_csv(df_name + ".csv")
13
14 # csv を書き出す関数( 日付つきのファイル名)
15 def to_csv_date(df, filename):
16     df.to_csv(filename + '_' + current_date() + '.csv')
```

2　ドキュメンテーション

前の課題では、コード規約について取り上げたが、今回のトピックもコード規約と同じく、自分がコードを書いたときに人に読んでもらいやすくするために必要な「ドキュメンテーション」について取り上げる。

コード規約にのっとったコードを書いて読みやすくなったとしても、他の人が書いたコードを 1 からすべてに目を通して理解するといったことは現実的ではない。実際にライブラリを使うときに実装コードをすべて見て理解してから使うのでは効率的とは言えない使い方になってしまう。そのようなときに、関数ごとにまとめられたドキュメントがあると、使いたい関数の検索性も高まり、理解したい部分を効率的に理解できるなど重要性を強く感じることができる。さらに、コード中に組み込むことで関数間の関係性も維持したまま、抜け漏れがなく書ける工夫もとても興味深く重要なものであると感じた。

2.1　コメント

コメント一つをとっても、良いコメントと悪いコメントが存在する。私も漠然と他の人が書いているコードを読みながら意識していたところでもあったが、それらが文面上で講義内では示していただいていて、納得できるとともに今後のコメントを書く上での指針にしようと感じた。実際にコードが行っている処理内容に関して記述するのであれば、1 行ずつコードを読んでいくことと大差ない。その一方、コードが存在する理由を書くことで、コードのかたまりとしての概観をつかむことができるので、可読性の向上につながると感じる。

2.2　docstring

ドキュメントを作成するということは、リファレンスマニュアルを作ることになるが、関数名とその機能、引数と返り値をはじめとした使用例といった、関数やクラスの仕様を説明するためのコメントとして、docstring というものがある。これらの基本要素は決まっているので、VScode にはそれを自動生成する Python Docstring Generator があり、この講義を通して知ったので、実際に導入してこのレポートで後述するドキュメント作成にも活用した。このように、フォーマットが決まっていることによって、関数によって説明する内容がばらばらになるということを防ぎ、統一感のある説明をしやすくできるという点で重要であるといえる。

2.3　ドキュメント作成ツール

ドキュメントをコードと別々に作成していては、作業量が倍になるだけではなく、一方の更新や作成のし忘れが起こりやすくなってしまう。そこで、コード内に書いたコメントからドキュメントを生成するツールとして、Doxygen と Sphinx があげられる。

Doxygen は、C/C++ を用いた開発を部活で行っていた際に使っていたため、知っているが、コード内にコメントとして加えるだけで html が作成されプログラム外からその関数の特性や、探したい関数を見つけやすくなった点で、一覧性と検索性の 2 点から優れていたと感じる。しかし、毎回 @ からはじめなければいけない点はコードを書く上で煩雑さを感じた。

次に、Sphinx についてはこの講義で名前については知ったが、スライドを見たらドキュメントで良くみたことのある形式であり、それらのライブラリが Sphinx で書かれていたことを知り Sphinx の人気の高さを実感した。コメントとしても、前述の拡張機能と合わせて最小限のコードで書くことができるようにも感じる。

2.4　Sphinx を用いた実践

前課題につづいて、自作した 3 つの関数についてのドキュメントを Sphinx を用いて生成を行ったところ図 1 のように生成することができた。Sphinx を使ったことは講義前はなかったが、実際に使ってみて使いやすさを感じたので、今後の自分が関わるプロジェクトでもこのようにドキュメントを作成していき、共有しやすくできるようにしたいと感じた。(これらのコード,html に関しても後述する github に push している)

図 1　Sphinx を用いた自作関数のドキュメント

3　バージョン管理・GitHub

3 つ目のトピックは、バージョン管理と Github についてである。これもこれまでにまとめた、「コード規約」、「ドキュメンテーション」と同じく、複数人でプロジェクトを進める上で重要になってくる考え方・ツールであり、個人開発をしている際にも有用であるので、講義前から使っていたが、この講義を通して改めて、バージョン管理や Github の使い方について学ぶことができた。

3.1　バージョン管理

講義内で紹介されている通り、私も git を使う以前はバージョンを管理する際にファイルをコピーした上でファイル名を変えて管理していた。パワーポイントや Google スライドなどでも最近ではバージョン履歴が保存されるようにもなってきているので、そのような機会は最近だと減っているかもしれないが、Git のバージョン管理では、単一ファイルの履歴を管理するだけではなく、branch をきって複数人で複数機能の開発を同時に行うことができるといったことは最大の強みであると思う。

3.2　Git VScode 拡張機能

また、Git については学び始めてからコマンドベースで操作を覚えたり、調べたりしていたためほとんど GUI ベースで操作することはなかったのだが、VScode にも拡張機能があり便利そうだと講義を聞き感じたので、実際に使ってみた。コマンドラインベースでは、コミットログを確認する際に見にくい部分があったが、拡張機能である Git Graph を活用することで、ブランチのマージなども含めてとても見やすく表示できるようになったので、今後自分で開発する際も活用していこうと感じた。

また、このレポートを書くのに際して Github にリポジトリを作りバージョン管理を行っているが、実際にそのログを見てみると図 2 のようになり、バージョンの管理ができていることが確認できる。

3.3　Github

GitHub についても講義以前から使っていたが、この講義を通じて体系的にかつ重要な部分を学ぶことができて改めて勉強になったと感じる。また、過去に作ったリポジトリで

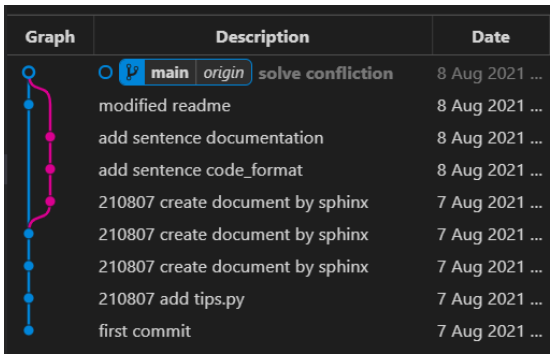

図 2　このレポートの Git Graph

https で Github とつないでいたものが多くあったので ssh への変更をしなければと思いつつできていなかったものもあったが、講義資料にあったリモート先を https から ssh へ変更する過程を参考に移行作業をすることができたことは、この講義で学んだことを活かせたことの一つである。

3.4　Github 実践

このレポートでは、現在レポートを書いている tex ファイル、出力した pdf ファイルも含め、これまでのトピックで実行した python のコードや、ドキュメンテーションを作る際に活用したディレクトリを Github でバージョン管理を行うことで、この講義で学んだことの実践を行った。

今回作成したリポジトリは、下記の software-report である。

https://github.com/takemi853/software-report

また、このリポジトリに関しては、はじめからリモート先を ssh と設定して作成した。

実際にレポート用として GitHub を活用したが、個人で使っていてもバージョンを管理しながらレポートを書けたり、コードを書く際の整理にもなるので、有用であると感じた。この講義で学んだことも活かし、今後も Git 及び GitHub を活用していきたいと強く感じる。

4　コンテナ管理

近年、注目される技術の一つに Docker をはじめとしたコンテナ管理があげられると思う。実際に講義前にも Docker を使ったことはあったものの、コマンドの意味については理解しきれていなかったことも多く、今回の講義を通して、仕組みやコマンドについての理解が進んだので、今後はただ使うだけでなく、動作の意味も踏まえた上で使えるようにしていきたいと思う。

4.1　仮想化

仮想化を行うことで、本来 windows PC で動かすことのできないものでも、Linux を動かして実行できたり、ゲストの環境を隔離した上でテストできたりするなどメリットは大きい。実際、私も windowsPC 内で Linux を動かせるようにしているが、便利に感じる面は多々ある。その一方で、仮想化ではハードウェアも含めて仮想化するため動作が重くなってしまうというデメリットもあり、後述するコンテナ仮想化の考えが重視されてきていることがよく分かる。

4.2　コンテナ仮想化

仮想化の欠点を踏まえ、アプリが動作するために必要な最小限のものを仮想化するだけで良いという考えのもとに、それらをまとめてコンテナと考え、コンテナごとに仮想化を行うというのがコンテナ仮想化である。

これに関しては、仮想環境をそのまま共有してはかなり大きなデータとなり、重複も生まれやすいが、コンテナという単位にまとめて必要最低限の環境を共有することではリソースを効率よく活用できるという点で非常に有用であると考えられる。

4.3　Docker

Docker では先述の通り、コンテナ単位で仮想化されたものを活用することができるので、環境構築がほぼ不要でプログラムを実行できるという点が便利だと感じ、講義前からも使っていたが、その際に勉強していて感じたことは以下の通りである。

まず、繰り返しになるがプログラムを書き実行するため必要な環境構築が容易になる点である。これは、自分のよく使っている PC でよく使っているような環境のまま動作する

ものであれば問題ないが、新しくインストールする必要があるライブラリなどがあるとバージョンの整合をはじめとしたエラーがつきものである。これらの関係性を一度コンテナ内で作ってしまい、テスト環境・デプロイ環境として隔離できることはかなり重要であり、チーム開発を行っている場合では、チームメンバー内でのローカル環境を統一させたりすることが不要であったり、新メンバーが入ってきたときの環境構築がしやすくなる。さらに本番環境にのせるときにもコンテナを介して環境構築することで 2 二度手間になることを防げることもあり重要であることがわかる。

次に、DockerHub 内でのイメージの豊富さである。Github 同様に Docker にも自分が作成したコンテナイメージを共有できる Hub として DockerHub があるが、イメージの豊富さにも驚いた。公式のイメージをとっても、Ubuntu や Python をはじめとして postgres や mysql などの SQL 系や openjdk などが提供されていることがわかり、これらを活用して仮想環境を立てるだけでなく、これらをベースに自分が使いたい固有のイメージを作っていくことが容易となることを考えると、応用しやすくなると感じた。

4.4 実践

この講義では、後半の実践系の講義を通じて Docker 及び、Docker-compose を活用してプログラムを実行してきた。

デバックや、テスト、ドキュメンテーションをはじめとして、これらの内容をすべて 1 から環境構築した場合だと、かなりの煩雑さがあったことと思うが、docker を使うことによって環境構築に使う時間をほとんど割くことなく本質的なコードの解釈や、コーディングに時間を費やすことができたという点は、この講義において Docker の有用性を特に感じたことの一つである。

また、講義前となるが、Rails と PostgreSQL を用いた簡単なアプリケーションを実装するために、それぞれをコンテナとして立てた上で docker-compose としてまとめる Dockerfile を作成して環境構築を行ったことがある。そのときには、コンテナ内部の構成を理解する必要とあると感じつつ作成していたが、今回の講義で一度イメージを作ってしまえばその後の実行が容易になるということを改めて実感することができたので、DockerHub からイメージを clone して使うだけでなく、自分で Dockerfile を書いたりしてイメージを作るところまでも積極的にできるように実践したいと感じた。