『ロバスト Python』を読もう

Let's Read "Robust Python"

Hayao Suzuki

BPStudy#189

May 29, 2023

Who am I ?

お前誰よ

名前	Hayao Suzuki（鈴木　駿）
Twitter	@CardinalXaro
仕事	Software Developer @ BeProud Inc.

- 株式会社ビープラウド
  - IT 勉強会支援サービス connpass 

  - オンライン学習サービス PyQ 

  - システム開発のためのドキュメントサービス Tracery 

Who am I ?

監訳した技術書

- ロバスト Python(O'Reilly Japan)
- 入門 Python 3 第 2 版 (O'Reilly Japan)

査読した技術書（抜粋）

- Effective Python 第 2 版 (O'Reilly Japan)
- SQL ではじめるデータ分析 (O'Reilly Japan)
- マイクロフロントエンド (O'Reilly Japan)
- マイクロサービスアーキテクチャ 第 2 版 (O'Reilly Japan)

https://xaro.hatenablog.jp/ にリストがあります。

Who am I ?

発表リスト（抜粋）

- レガシー Django アプリケーションの現代化 (DjangoCongress JP 2018)
- SymPy による数式処理 (PyCon JP 2018)
- Python と楽しむ初等整数論 (PyCon mini Hiroshima 2019)
- 君は cmath を知っているか (PyCon mini Shizuoka 2020)
- インメモリーストリーム活用術 (PyCon JP 2020)
- 組み込み関数 pow の知られざる進化 (PyCon JP 2021)

`https://xaro.hatenablog.jp/` にリストがあります。

今日のテーマ

ロバスト Python

- Patrick Viafore『Robust Python』 (O'Reilly Media)
- 長尾さん翻訳、鈴木監訳
- 長尾さん、鈴木のペアは『入門 Python 3 第 2 版』以来 2 回目

『ロバスト Python』を読もう

- ざっと、どんな事が書いてあるのか紹介します
- 384 ページの内容を 30 分で読んだ気分になろう！

ロバスト Python の構成

4 部構成

第 I 部 型アノテーション
第 II 部 ユーザ定義型
第 III 部 大規模な変更への対応
第 IV 部 セーフティネットの構築

なお、第 1 章「ロバスト Python 入門」は導入となる章

従来の技術書との違い

Python と型ヒント

- 型ヒントは Python 3.5 で導入（PEP 484）
- Python 3.5 は 2015 年 9 月にリリース

型ヒントの付け方はすでに知ってるよ

- 型ヒントの付け方の解説記事や書籍は多い（8 年の蓄積）
- 型ヒントの運用方法や考え方に踏み込んだ本は存在するのか

そもそも、中級者以上向けの Python 本の絶対数が少ないよね

ロバスト Python のここがすごい

ロバスト Python の特徴

- 型ヒントについて 14 章にわたって詳細に解説（第 I 部、第 II 部）
- ロバストな設計論（第 III 部）
- 型ヒントや型だけではカバーしきれない要素も対応（第 IV 部）

第 1 章 ロバスト Python 入門

ロバスト Python ってどんな本

要するに、本書はロバスト（頑丈/頑健/堅牢）な Python を書くための本である。（P. 1）

そもそもロバストって何？

コードベースのロバストネスとは、絶えず変化しても耐久性が高く、エラーを起こさないことである。（P. 4）

なぜロバストにする必要があるのか？

その答えの核心はコミュニケーションにある。（中略）そのためには、将来のメンテナと直接会わなくても自分の論拠と意図が伝わるようにしたい。

コミュニケーションの分類

近接性 情報の発信者と受信者の間にある時間的な距離

コスト コミュニケーションに必要な労力

問：時間的な距離は数学的な距離（空間）の性質を満たすか。

コミュニケーションの分類

低コスト、近接性が必要	直接的な対話、Slack
高コスト、近接性が必要	会議、カンファレンス
高コスト、近接性が不要	メール、設計書、アジャイルボード
低コスト、近接性が不要	README、git ログ、コードのコメント

なぜロバストにする必要があるのか？

その答えの核心はコミュニケーションにある。（中略）そのためには、将来のメンテナと直接会わなくても自分の論拠と意図が伝わるようにしたい。

そのためには？

未来のメンテナに意図を伝えるには、低コストで近接性が不要なコミュニケーションが必要である。

低コストで近接性が不要なコミュニケーション

- コードを読むだけで理解できればよい
- コードを理解するのに必要な時間を最小限に抑えたい

そのためには？

データ型の選択は将来の開発者に対して自分の意図を表現することであり、正しいデータ型を選択すれば保守性が向上する。（P. 19）

2 章 Python データ型入門

- Python は強い型付け言語
- Python は動的型付け言語

動的型付け言語は本質的にロバストではないのか

- 動的型付け言語でもロバストに書ける、少し大変なだけ
- 静的型付け言語の場合よりもよく考える必要がある
- 静的型付け言語を使えば必ずしもロバストに書けるわけではない

3 章 型アノテーション

- 型ヒントの構文的な付け方は知っているよね（8 年の蓄積）
- 型ヒントはツール（mypy など）と組み合わせて真価を発揮する

型ヒントも mypy もタダではない

- 公開 API、ライブラリのエンドポイント
- 複雑な型、わかりにくい型
- mypy 先生の指示に従う

4 章 型制約

- Optional でぬるぽバグを潰そう
- Union で表現可能な状況を減そう
- Literal でさらに制約をかけよう
- Final で定数であると主張しよう
- NewType で潜在的なバグを潰そう

Annotated のこと、時々でいいから、思い出してください

- 任意のメタデータを追加できるが、単なるメタデータ
- 専用のツールができたら熱くなるかも

5 章 コレクション型

- 同種コレクションと異種コレクションの概念を理解しよう

同種コレクション

格納されたすべての値が同じデータ型であるコレクション

異種コレクション

異なるデータ型を含むコレクション

典型的な例：JSON を辞書に変換する

- `dict[str, Union[str, int, float, None]]` のような型は書きたくない！
- `dict[str, Any]` では型ヒントが形なしだ！

そんな時は TypedDict

型ヒントを内蔵した辞書

そんな時は dataclass

自分で JSON の形式を制御できるなら `dataclass` がよい。

6 章 型チェッカのカスタマイズ

- mypy では `Optional` や `None` に関するチェックを有効にすると良いよ
- mypy が遅い？ ならばデーモンモードやリモートキャッシュで高速化しよう

型チェッカは mypy だけじゃない

Pyre	Facebook 製、静的解析ツール Pysa もついてくる
Pyright	Microsoft 製、VSCode 拡張の Pylance のベース

7 章 実践的な型チェックの導入

ペインポイントを特定してコストをかけ過ぎずに型ヒントをいれよう

まずはここから

- 新規コードのみ型ヒント
- ボトムアップ（ユーティリティ、ライブラリ）に型ヒント
- 利益を生み出すビジネスロジックに型ヒント
- よく書き換える場所に型ヒント
- 複雑な部分に型ヒント
- MonkeyType や Pytype で自動的に付与も有効かも

8 章 列挙型

静的な値のコレクションから値を 1 つだけ表現したい場合に使う

列挙型のアンチパターン

- 動的に値が変化する列挙型。辞書を使おう。
- `IntEnum` や `IntFlag` は後方互換性のために使う。新規に使うのはバグの元。

第 II 部 ユーザ定義型

9 章 データクラス

Python の世界を大きく変えたデータクラス（原著者一押し）

データクラスの使いどころ

- 異種コレクションはデータクラス、辞書は同種コレクション
- TypedDict とデータクラスは、まずはデータクラスを検討しよう
- namedtuple は後方互換性のために使う

データクラスは万能なのか？

- データクラスは属性同士が独立している場合のみ有効

第 II 部 ユーザ定義型

10 章 クラス

- 君はクラスの本当の使いどころを知っているか
- クラスとは、不変式である（筆者のテーゼ）

ユーザ定義型の使いどころ

列挙型	同種データ、スカラー値
辞書	同種データ、コレクション
データクラス	異種データ、不変式がない
クラス	異種データ、不変式がある

図 10-1 は印刷して壁に貼ろう

第 II 部 ユーザ定義型

11 章 インタフェース

- 使いやすいインタフェースとは
- この章はやや観念的である

使いやすいインタフェースを追及する

- 利用者のように考える（TDD、README 駆動開発、ユーザビリティテスト）
- 特殊メソッド、コンテキストマネージャの活用

12 章 部分型

- 継承は置換可能性（Liskov の置換原則）に注意する
- 継承よりもコンポジション（合成）

正方形は長方形か？

- 数学的には、正方形は長方形である。
- 正方形を長方形の継承として表すと破綻する。

問：正方形と長方形の定義を述べよ。また、命題「正方形は長方形である」を正方形と長方形の定義に基づいて証明せよ。

13 章 プロトコル

- 静的型チェッカの穴を埋める存在
- ダックタイピングと静的型チェッカをつなぐプロトコル
- 構造的部分型と名目的部分型をつなぐプロトコル

14 章 pydantic による実行時型チェック

- 静的型チェッカの穴を埋める存在
- 自然に入力値などをチェックできる

pydantic はパースライブラリでもある

- `int` ならば `"123"` も `5.5` も `int` に変換する
- `StrictInt` などを使う必要もあるかも

第 III 部 大規模な変更への対応

15 章 拡張性

- 拡張性とは、システムの既存部分を変更せずに新機能を追加できるというシステムの性質
- 開放閉鎖の原則がベース（どちらが先だろうか？）

原則違反の見つけ方

- 簡単なことが難しくなっていないか
- 類似の機能の実装が遅れていないか
- 見積もりがいつも大規模になっていないか
- コミットに大規模な差分が含まれていないか

第 III 部 大規模な変更への対応

16 章 依存関係

依存関係は 3 種類ある

- 物理的依存関係
- 論理的依存関係
- 時間的依存関係

依存関係を可視化せよ

- サードパーティライブラリ同士の依存関係
- インポートモジュール間の依存関係
- 関数の呼び出しの可視化

第 III 部 大規模な変更への対応

17 章 コンポーザビリティ

コードをポリシーとメカニズムに分離せよ

ポリシー ビジネスニーズを解決するための直接の責任を担うコード

メカニズム ポリシーを実現する仕組みを提供するコード

18 章と 19 章

15 章「拡張性」、16 章「依存関係」、17 章「コンポーザビリティ」の具体例として読もう。

第 IV 部 セーフティネットの構築

20 章 静的解析

静的解析　本書では Pylint が紹介されているが、個人的には flake8 がよいと思っている。

複雑度チェッカ　MeCabe の循環的複雑度、空白数ヒューリスティクス

セキュリティ　Bandit

静的解析だけに絞ってはいけない

複数の解析ツールを導入して防衛線を複数構築しよう

21 章 テスト戦略

以下の 2 冊を読んでいる人にはものたりないかもしれない。

- テスト駆動 Python（翔泳社）
- 単体テストの考え方／使い方（マイナビ出版）

pytest を使うと自然に実践できるかもしれない

第 IV 部 セーフティネットの構築

22 章 受け入れテスト

Gherkin 言語を紹介しているが、個人的にはどうだろう、という。

あるプログラマの思い出

過去、仕事を始めたばかりの頃、まともにプログラミングができない状態で BDD や Cucumber を含む Ruby on Rails のチュートリアルを読んで、BDD や Cucumber がかなり嫌いになってしまった。それを未だに引きずっているのかもしれない。

第 IV 部 セーフティネットの構築

23 章 プロパティベーステスト

Hypothesis によるプロパティベーステストの紹介

プロパティベーステストとは

固定した入出力に基づくテストではなく、入出力が満たすべき性質を記述するテスト。その性質を満たす値をツールが自動生成してテストケースを生成する。

第 IV 部 セーフティネットの構築

24 章 ミューテーションテスト

mutmut によるミューテーションテストの紹介

ミューテーションテストとは

目的としては、テスト自体をテストするテスト。ソースコードをツールで書き換えて既存のテストを走らせて、テストが成功してしまったものを炙りだす。

まとめ

『ロバスト Python』の全体的な主張

- 自分がコードで表現したいことをデータ型を使って明確に伝えよう
- データ型とはコミュニケーション手段である