

# 業務で使える一歩進んだ Python使いになるために

Motoki Hirao

This session will be conducted in Japanese.

English slides are attached after the Japanese slides on another day.

# 発表者のメタ情報

- Motoki Hirao, Python と出会って8年くらい、Python 3.4 から
- 株式会社コンテンツデータマーケティング (CDM) 所属
- Django, Flask, Airflow など毎日 Python 漬け
- PyCon APACは当日スタッフ参加中
- SNS
  - X: [\\_\\_yumechi](#) misskey: [@\\_\\_yumechi@misskey.systems](#)
  - GitHub: [yumechi](#)

# ちょっとだけ会社の紹介

コンテンツの届け方を、再発明する。

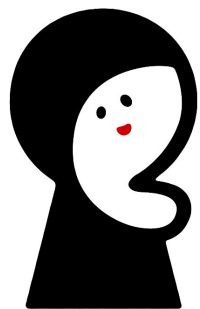
私たちの使命は、コンテンツの届け方の再発明を通して、コンテンツと生活者との出会いを圧倒的に増やすこと。

<https://contentdata.co.jp/> より



# ちょっとだけ製品の紹介

ID基盤、コンテンツビジネス向けのDXソリューションをやってます




# Unikey

ユニキー



# Unikey Experience Cloud

<https://contentdata.co.jp/products/>  
より

技術力でチームを  
底上げできる方、ぜひ  
お声掛けください 

トーク本編に戻りまして…

# 想定対象（Web、ツールよりかも）

- 他言語から Python を始めた方
- Python を初めてのプログラミング言語にしてる方
- 実際に業務で Python を使いたい方



# 今回の発表の概要

- (3分) 入門 Python エンジニアが越えなければいけない壁
- (5分) 業務で Python を書き開発していくには？
- (3分) 4年Pythonを業務で使って感じたメリット・デメリット

# 今回の発表の概要

30分のトークプロポザルで出したものを  
15分に再構成している関係で、巻き気味で  
進行しますのでよろしくお願いします

入門 Python  
エンジニアが  
越えなければいけない壁

# 個人開発から業務での開発に移行する難しさ

- （書く）チームで書くべきでないコードが含まれている
- （読む）チームメンバーのコードを読むのが大変

# ビルドインのlist関数を破壊しよう

- 実は Python で書くと簡単に壊れるコードっていうのがある

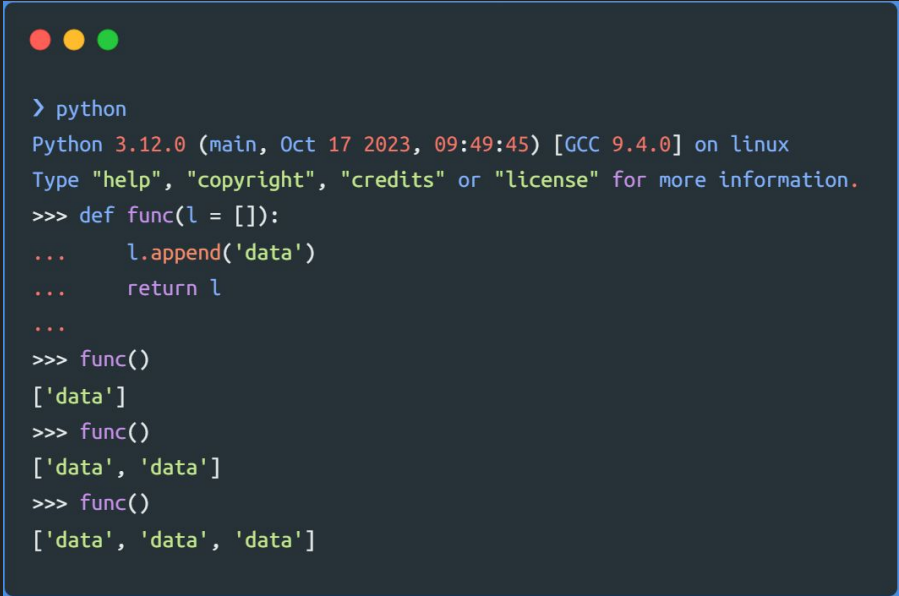


```
> python
Python 3.12.0 (main, Oct 17 2023, 09:49:45) [GCC 9.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> list = ['a', 'b', 'c']
>>> input_list = map(int, input().split())
1 2 3
>>> input_list
<map object at 0x7f1d655ac340>
>>> list(input_list)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'list' object is not callable
```

<https://carbon.now.sh/>  
で左の画像は作りました

# デフォルト値に値を貯めてみよう

- 実は Python で書くと簡単に壊れるコードっていうのがある



```
> python
Python 3.12.0 (main, Oct 17 2023, 09:49:45) [GCC 9.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> def func(l = []):
...     l.append('data')
...     return l
...
>>> func()
['data']
>>> func()
['data', 'data']
>>> func()
['data', 'data', 'data']
```

<https://carbon.now.sh/>  
で左の画像は作りました

# チームで書くべきでないコードが含まれている

- Python自体の言語特性をよく知らないといけない
  - Python のビルドイン関数は実は上書き可能
  - オブジェクトを引数にとりデフォルト値がある関数で、引数を直接変更するパターンの場合、デフォルト値の値が書き換わり続けてしまう
  - スコープの管理が難しい etc...
- 技術ブログサイトやポータルを見ていると、サンプルコードの例でこの書き方をしているケースをいまだに見る…😭

# 他のチームメンバーのコードを読むのが大変

# 1 2 3 4 5 を受け取って [1, 2, 3, 4, 5] にするタスク

```
def solve1(_input: str) -> str:  
    return list(map(int, _input.split()))
```

```
def solve2(_input: str) -> str:  
    return [int(x) for x in _input.split()]
```

```
def solve3(_input: str) -> str:  
    ret = []  
    for x in _input.split():  
        ret.append(int(x))  
    return ret
```

```
if __name__ == '__main__':  
    _input = input()  
    print(solve1(_input) == solve2(_input) == solve3(_input))
```

競技プログラミングなどでよく問われる  
標準入力を分割して配列にする、  
というロジック。

map, append, 内包表記と3通りある。  
(ちなみに内包表記が一番早いはず)

<https://carbon.now.sh/>  
で左の画像は作りました



# チームメンバーのコードを読むのが大変

- コードが正しく、読みやすい状態を維持する必要がある
  - 「Python はだれが書いても同じようになるので読み書きがしやすい言語！」と一度以上は聞いたことがある… ありますよね？
- しかしあれくらい短いロジックですら、書き方が複数ある
  - 実際に業務で考えていく問題を考えると、さらに複雑になり、考えるべき問題が多い
- 研究用のコードが GitHub に上がってるけど動かないとか…

業務で Python を  
書き開発していくには？

# Python は業務で採用すべきでない？

- そんなことはない…
  - いろいろ言ってくる人たちはポジショントークだと思って一意見だと思ったほうが良い、自分のトークも一意見に過ぎない
- ライブラリは充実しているし、先人の知恵も多い
- いろいろな環境で動く強みもある
- （初手は）学習コスト低め

# Python は業務で採用すべきでない？

Python を始めるのは学習リソースも豊富で、ライブラリもあって簡単。  
一方で継続的に開発して、価値のあるものを開発し続けることは学習コストが高い。  
動的型付けの言語全般にも共通しそう。

なので、私たちは  
より良いやり方を  
学び続けるしかない…！

# 業務での開発に求められること

- (前提) 運用保守可能な継続的な開発
  - 特にWeb開発の場合、開発要件や顧客ニーズの変化により機能リリースが継続的に行われることが珍しくない
- メンバーが理解できるコード
  - 読み書きが不自由なく快適に行える状態を維持したい
- 再現性のある環境
  - 今は1人で開発していても、いつか自分が開発しなくなったり、新しい人が増えるかもしれない

# 業務での問題を解決するための5つのツール

仮想環境

パッケージ  
管理

Lint

Formatter

テスト  
フレームワーク

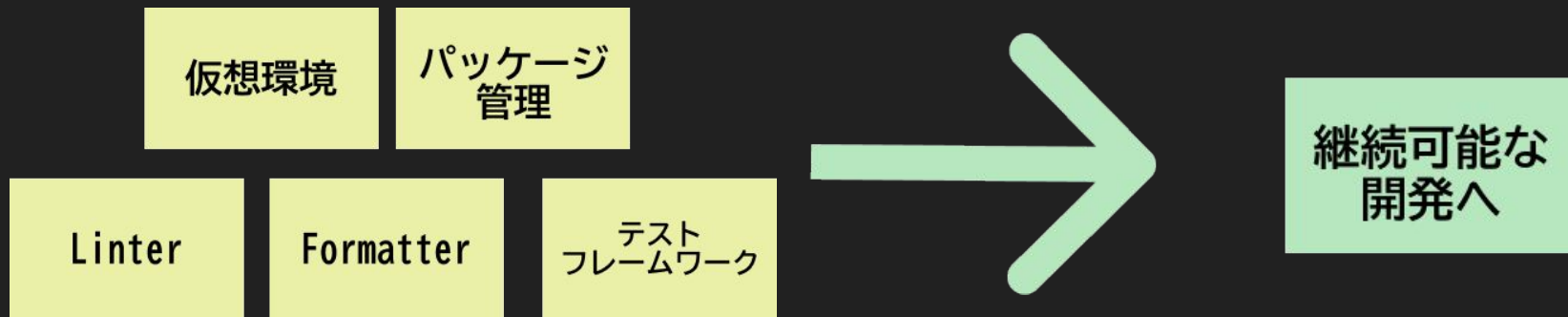
# ツールの役割

- 仮想環境: 仮想的に同じ環境で開発することにより、開発者の環境に影響せず開発できるようにする
- パッケージ管理: 利用しているライブラリを明確にし、ライブラリの管理・インストールを容易にする
- Linter: ルールに沿ってコードを書けるよう検証する
- Formatter: 自動的にコードを整形する
- テストフレームワーク: 自動テストを行い、保守性を高める



# 業務での開発に求められることへの問題解決

- Linter, Formatter, テストフレームワーク → チームメンバーや明日の自分自身が読めるコード
- 仮想環境、パッケージ管理 → 再現性のある開発環境



# 時間の関係で具体的な設置値は省きます

- 直近で読んだ記事だとこちらがまとまっていた
  - 開発品質とDeveloper eXperienceを高めるコンテナ開発環境のご紹介 (Python) - ABEJA Tech Blog
  - <https://tech-blog.abeja.asia/entry/container-environment-202310>

# 仮想環境

- Docker が実質のデファクトスタンダード
  - 手元に Docker が動く環境を用意する必要あり
  - クラウド環境で動くOS環境も比較的再現しやすい
- devcontainer と GitHub CodeSpaces の活用によってローカル環境に全く環境を置かずとも開発することも可能になった
  - ※業務では試してないです
- ローカル環境だと pyenv, asdf など複数バージョン動かせるようにしとくと便利かも

# 実際に Codespaces で体感してみる

- Setting up a Python project for GitHub Codespaces – GitHub Docs
  - <https://docs.github.com/en/codespaces/setting-up-your-project-for-codespaces/adding-a-dev-container-configuration/setting-up-your-python-project-for-codespaces>
- これに沿ってまずやってみるのがよさそう、付属するツール類も全部ポチポチ～ってやるとそろあの感動します

# パッケージ管理

- Docker 使わない場合でも最低限このどれかは使ってほしい
- Poetry
  - 最近のパッケージ管理だと今のところ一番だと思っている
  - パッケージングのしやすさも魅力
- Pipenv
  - 環境変数読んでくれるのが楽
- Rye
  - 最近出てきたツールでRust製、今後に期待…？

# パッケージ管理、何がうれしいか？

- 仮想環境（venv）で区切れる
  - ライブラリをOSに直インストールしてしまうと、ほかのアプリケーションに影響が出てしまう
  - 特定のPythonバージョンで動かしていることを明示的にしたい
    - Python 3.12 だから Python 3.8 では動かないかも！とか
- インストールしているパッケージを明確にしたい
  - requirements.txt はインストールしているもののみ、依存がどうなっているかはわからないため

# 奥深いパッケージ管理

- 直近の stapy で発表されていた内容がとても参考になりました
  - Pythonのパッケージ管理の中級者の壁を超える stapy#98 - Speaker Deck
    - <https://speakerdeck.com/vaaaaanguish/pythonnopatukeziguan-li-nozhong-ji-zhe-nobi-wochao-eru-stapy-number-98>

# Linters

- ruff
  - 割と最近出た Rust 製の Linter
  - これまでの Linter より高速に動作
- Flake8
  - カスタマイズやプラグインが充実している Linter
  - 比較的早い、活用実績も多い
- Mypy
  - 型アノテーションのチェックや、型チェックの強化



# Formatter

- ※ 個人的には PyCharm のフォーマットでも十分では？と思っている
- Black
  - 設定をあまりしなくても割といい感じにフォーマットできる
- yapf
  - フォーマットに細かい設定をしたい場合は有効（らしい）

# テストフレームワーク

- unittest
  - 標準ライブラリに含まれている
  - これだけでも十分パワフル
- pytest
  - unittestよりも多くの機能を提供していて、柔軟なテストが可能
  - parametrize が便利だなあとと思っている…

# 業務での問題を解決するための5つのツール

仮想環境

パッケージ  
管理

Lint

Formatter

テスト  
フレームワーク

# 業務での問題を解決するための5つのツール

どのツールを選ぶべきかはおかれている  
環境によって変わりますが、  
継続的に開発しやすいように整えよう

# 他にも押さえておきたいツール

エディタ

ログ管理

API  
ドキュメント

ロードテスト  
ツール

セキュリティ  
テストツール

APMツール

# ツールの役割

- エディタ: コードリーディング、コード記述
- ログ管理: 適切なログ出力を行い、調査を容易にする
- APIドキュメント: APIの I/F を示し、保守性を高める
- ロードテストツール: 想定される負荷を類似的に再現し、要求される性能であることを示す
- セキュリティテストツール: セキュリティに問題のあるコードを検知
- APMツール: ボトルネックを検知する

# 対応するPythonのツール類

- エディタ：好きなものを…
- ログ管理：標準パッケージ、structlog、loguruなどなど
- APIドキュメント：Swagger、Sphinx などなど…
- ロードテストツール：Locust
- セキュリティテストツール：Bandit、Safety
- APMツール：New Relic SDK など

今回は時間の関係で  
説明を省略します





# PyCharmありがとう！



- ・ Typo発見に役立つ
- ・ フォーマットGod
- ・ めっちゃいい感じにインデントしてくれる
- ・ てかないと、マジ無理

# エディタだけは紹介させてほしい

- PyCharm Professional を愛用しています
  - <https://www.jetbrains.com/pycharm/>
  - 一通り必要そうな設定がデフォルトで入っている
  - コードジャンプ機能が VSCode に比べて恐ろしいくらい優秀
    - VSCode でもいけるやろと思ってた時期もありました
  - 有料エディタですがぜひ検討してください
  - PyCharm for Education という学習者向けのものもあります
- エディタにもこだわって、開発環境を育てましょう

# 自分の環境のカラーテーマとフォント

- カラーテーマ

- Tokyo Night <https://plugins.jetbrains.com/plugin/18820-tokyo-night-theme>

- フォント

- プログラミングフォント 白源 (はくげん/HackGen)  
<https://github.com/yuru7/HackGen>
- プログラミング用フォント Utatane <https://github.com/nv-h/Utatane>
- サイズは16くらいでやや大きめ

# 他紹介しきれなかったトピック

- AI をガンガン使う！ コードサンプルをひたすら得よう
  - GitHub Copilot
  - チャットツール (Copilot Chat, ChatGPT, Bard) 今回の資料でもフル活用
- ツール類については紹介したものの、手法的にどうすべきなのか？
  - printデバッグでなく、デバッガを使おう
  - CI で自動的に Lint, unittest しよう
  - コードレビューをしてよいコードを伝えていく
  - Type hinting とにかく書く、DocString も絶対に書く

4年 Python を  
業務で使って感じた  
メリット・デメリット

# メリット

- ライブラリが本当に豊富
- dict 型で何でも受けられる
  - json を使った通信で、パラメーターの増減に本当に強いと思う
  - もう少し固くやりたい場合は TypedDict や dataclass という選択肢もある
    - (実際最近はそのようなコードを増やしている)
- インデントで適切にシンタックスを読むことができる
- コードの補助や参考となる情報が多い、読み書きできる人多い
- (自分の感覚で自然な言語でアルゴリズム考えやすい)

# デメリット

- 型に対して緩やかすぎる時がある
  - この dict には何が入っているんだ…！（人生で一番時間を使っている）
- インデントが深いと脳への負荷が高すぎる、5回インデントは無理
  - class, method, for, for, if みたいなネストは結構厳しい
- Python っぽい書き方のマスターが難しい
  - 自分も昔は Java や C++ っぽい書き方だったなと感じている
- すごく設計が難しい、と思う
  - 型の問題もあるし、クラス変数、インスタンス変数の外からの書き換え etc...

# デメリットに対して

- 型に対して緩やかすぎる時がある → TypedDict や dataclass など  
などを活用しよう
- インデントが深すぎると脳への負荷が高すぎる → 複雑度をツールで  
計測したり、for 文になったら分割。とにかく小さくする。
- Python っぽい書き方のマスターが難しい → ライブラリのコードを  
読もう (いいコードを読む)
- **すごく設計が難しい、と思う… ← これだけ本当に難しい**



# 設計まで踏み込みこむ？

- Python だけでなく他言語の文化にも触れたほうがいいかも？
  - PHP に学ぶものは多い
  - 去年まで PHP のエンジニアやっていたんですが、動的型付け言語で長年 Web 開発を支えていた PHP 言語でのノウハウ・設計は学ぶべきところが多い
    - Python も動的型付け、Type Hinting 後付けと似たところが多い
- ドメイン駆動設計、アジャイル開発、などなど、高速で運用可能な形で開発し続けるプラクティスもたくさんあり、身につけておきたい
- (Django で究極的に設計で完成されている本教えてください)

# Pythonに限らずよかった本

- アジャイルプラクティスガイドブック チームで成果を出すための開発技術の実践知
  - <https://www.shoeisha.co.jp/book/detail/9784798176826>
- 読みやすいコードのガイドライン —持続可能なソフトウェア開発のために
  - <https://gihyo.jp/book/2022/978-4-297-13036-7>
- 良いコード／悪いコードで学ぶ設計入門 —保守しやすい 成長し続けるコードの書き方
  - <https://gihyo.jp/book/2022/978-4-297-12783-1>
- 「Clean Agile 基本に立ち戻れ」
  - <https://www.kadokawa.co.jp/product/302007001102/>

テストコード書くべし！  
型が緩い部分とはにかく  
自動テストでカバーする。  
之、即ち動的型付け言語の  
鉄則なり。

他にも話したかったことが  
たくさんありますが、  
懇親会でお話ししましょう！

# まとめ

- 1人での開発とチームでの開発は壁がある
- チームで開発する場合は同じ環境で開発できるように整備する
  - 仮想環境、Linter, Formatter, unittest ツール, CI/CD
- Python 自体の言語の特性を良く知っておく
  - 動的型付けの言語であること、GIL、やってはいけないいくつかのこと、同様の目的を達成するコードは複数あること
- ほかの言語圏からもいろいろ学ぼう
  - Web なら PHP, 設計なら Java などなど…

# 利用情報

- スライド作成: Google Slide <https://www.google.com/slides/about/>
- フォント: BIZ UDGothic <https://fonts.google.com/specimen/BIZ+UDGothic>

# English Slide

(Please note that the text was created using Google Translate, so there are some unnatural parts.)

To become an advanced  
user of Python that can  
be used at work

Motoki Hirao



# Presenter meta information

- Motoki Hirao, I've been using Python for about 8 years now, starting with Python 3.4.
- Belongs to Content Data Marketing Co., Ltd. (CDM)
- Django, Flask, Airflow, etc. Immersed in Python every day
- PyCon APAC has staff participating on the day
- SNS
  - X: [\\_\\_yumechi](#) misskey: [@\\_\\_yumechi@misskey.systems](#)
  - GitHub: [yumechi](#)

# A little introduction to the company

Reinventing the way content is delivered.

Our mission is to dramatically increase the number of encounters between content and consumers by reinventing the way content is delivered.

From <https://contentdata.co.jp/>



# A little introduction to the product

We provide DX solutions for ID infrastructure and content businesses.



Unikey  
ユニキー



Unikey  
Experience  
Cloud

From

<https://contentdata.co.jp/products/>

X: [@\\_yumechi](#)   misskey: [@\\_yumechi@misskey.systems](#)

Returning to the main  
talk...

# Assumed target (web, maybe more than tools)

- Those who started using Python from another language
- For those using Python as their first programming language
- People who actually want to use Python in their work

# Summary of this announcement

- (3 min) Introductory Python barriers that engineers must overcome
- (5 min) How to write and develop Python at work?
- (3 min) Advantages and disadvantages I felt after using Python at work for 4 years

## Summary of this announcement

We are reorganizing the 30-minutes talk proposal into a 15-minutes talk proposal, so it will move forward in a slow manner, so please bear with us.

Introductory Python  
barriers that engineers  
must overcome



# Difficulties in transitioning from personal development to professional development

- (write) Contains code that should not be written by a team
- (Read) It's hard to read your team members' code

# Let's destroy the built-in list function

Actually, there is some code that can easily break when written in Python.




```
> python
Python 3.12.0 (main, Oct 17 2023, 09:49:45) [GCC 9.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> list = ['a', 'b', 'c']
>>> input_list = map(int, input().split())
1 2 3
>>> input_list
<map object at 0x7f1d655ac340>
>>> list(input_list)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'list' object is not callable
```

I created the image on the left with  
<https://carbon.now.sh/>

# Let's save the value to the default value

Actually, there is some code that can easily break when written in Python.

A terminal window with a dark background and light-colored text. At the top, there are three colored circles (red, yellow, green). The terminal shows a Python prompt where a function 'func' is defined with a default argument 'l = []'. The function appends 'data' to the list and returns it. Three calls to 'func()' are shown, each resulting in a list containing 'data' repeated the number of times the function was called.

```
> python
Python 3.12.0 (main, Oct 17 2023, 09:49:45) [GCC 9.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> def func(l = []):
...     l.append('data')
...     return l
...
>>> func()
['data']
>>> func()
['data', 'data']
>>> func()
['data', 'data', 'data']
```

I created the image on the left with  
<https://carbon.now.sh/>

# Contains code that should not be written by a team

- Must be familiar with the language characteristics of Python itself
  - Python's built-in functions can actually be overwritten
  - In a function that takes an object as an argument and has a default value, if the pattern is to directly change the argument, the default value will continue to be rewritten.
  - Difficult to manage scope etc...
- When I look at Japanese technical blog sites and portals, I still see cases where sample code is written in this way... 😭

# It's hard to read other team members' code

# 1 2 3 4 5 を受け取って [1, 2, 3, 4, 5] にするタスク

```
def solve1(_input: str) -> str:  
    return list(map(int, _input.split()))
```

```
def solve2(_input: str) -> str:  
    return [int(x) for x in _input.split()]
```

```
def solve3(_input: str) -> str:  
    ret = []  
    for x in _input.split():  
        ret.append(int(x))  
    return ret
```

```
if __name__ == '__main__':  
    _input = input()  
    print(solve1(_input) == solve2(_input) == solve3(_input))
```

Frequently asked questions in competitive programming, etc.  
Split standard input into an array,  
That logic.

There are three types: map, append,  
and comprehension.  
(Incidentally, comprehension should  
be the fastest)

I created the image on the left with  
<https://carbon.now.sh/>

# It's hard to read your team members' code.

- Code must remain correct and readable
  - Have you heard at least once that "Python is an easy language to read and write because it's the same no matter who writes it!"?
- However, even with such short logic, there are multiple ways to write it.
  - When you think about the problems you actually have to consider in your work, they become even more complex, and there are many problems to consider.
- The research code is posted on GitHub, but it doesn't work...

How to write and develop  
Python at work?

# Should Python be adopted in business?

- No such thing...
  - It's better to think of people who say various things as positional talk and just one opinion, and your own talk is just one opinion.
- The library is extensive and there is a lot of wisdom from our predecessors.
- Has the strength to work in a variety of environments
- (For beginners) Learning cost is low



# Should Python be adopted in business?

Getting started with Python is easy with a wealth of learning resources and libraries.

On the other hand, continuous development and development of something valuable has a high learning cost.

This seems to be common to all dynamically typed languages.

So we  
a better way  
I have no choice but to  
keep learning...!

# What is required for business development

- (Prerequisite) Continuous development that can be operated and maintained
  - Particularly in the case of web development, it is not uncommon for feature releases to occur continuously due to changes in development requirements and customer needs.
- Code that members can understand
- I want to maintain a state where I can read and write comfortably without any inconvenience.
  - reproducible environment
  - Even if you are currently developing by yourself, one day you may no longer be doing the development, or new people may be added.

# 5 tools to solve problems at work

仮想環境

パッケージ  
管理

Lint

Formatter

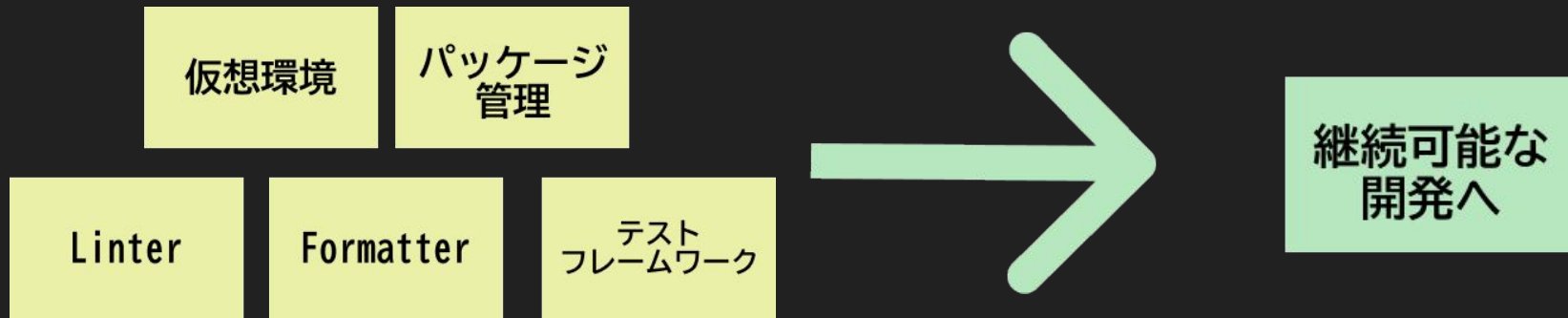
テスト  
フレームワーク

# Role of tools

- Virtual environment: By developing in the same virtual environment, it is possible to develop without affecting the developer's environment.
- Package management: Make it clear which libraries are used and make library management and installation easier
- Linter: Verify that you can write code according to rules
- Formatter: Automatically format code
- Test framework: Automate tests and improve maintainability

# Problem solving for what is required for business development

- Linters, Formatters, Test Frameworks → Code that can be read by your team members and yourself tomorrow
- Virtual environment, package management → reproducible development environment



# Due to time constraints, we will omit specific installation values.

- These are the articles I read recently:
  - Introducing a container development environment that improves development quality and Developer eXperience (Python) – ABEJA Tech Blog
  - <https://tech-blog.abeja.asia/entry/container-enironment-202310>
  - (Sorry, This article is in Japanese)

# virtual environment

- Docker is the de facto standard
  - You need to have an environment where Docker can run on hand.
  - It is relatively easy to reproduce the OS environment running in a cloud environment.
- By using devcontainer and GitHub CodeSpaces, it is now possible to develop without having to set up any local environment.
  - ※ I have not tried it at work.
- In a local environment, it may be useful to be able to run multiple versions using pyenv, asdf, etc.



# Experience it yourself with Codespaces

- Setting up a Python project for GitHub Codespaces – GitHub Docs
  - <https://docs.github.com/en/codespaces/setting-up-your-project-for-codespaces/adding-a-dev-container-configuration/setting-up-your-python-project-for-codespaces>
- It seems like it would be a good idea to try it first according to this, and if you click all the included tools, you will be impressed.

# package management

- Even if you don't use Docker, I want you to use at least one of these.
- Poetry
  - I think it's the best package management in recent times.
  - The ease of packaging is also attractive.
- Pipenv
  - Easy to read environment variables
- Rye
  - Rust is a recently released tool, what are your expectations for the future?

# What's good about package management?

- Can be separated by virtual environment (venv)
  - If you install the library directly to the OS, it will affect other applications.
  - I want to explicitly state that I am running a specific Python version.
    - Since it is Python 3.12, it may not work with Python 3.8! And
- I want to clarify which packages are installed.
  - requirements.txt only shows what is installed and does not know what the dependencies are.

# Deep package management

- The content presented at the recent Stapy was very helpful.
  - Overcoming the barrier of intermediate Python package management stapy#98 – Speaker Deck
  - <https://speakerdeck.com/vaaaaanguish/pythonnopatukeziguang-li-nozhong-ji-zhe-nobi-wochao-eru-stapy-number-98>
  - (Sorry, This slide is in Japanese. Stapy is a Python study group in Japan.)

# Linters

- `ruff`
  - A Linter made by Rust that came out relatively recently.
  - Runs faster than previous linters
- `Flake8`
  - Linter with plenty of customization and plugins
  - Relatively fast and has many usage records
- `Mypy`
  - Checking type annotations and strengthening type checking

# Formatter

- ✂Personally, I think the PyCharm format is sufficient. I think so
- Black
  - You can format it fairly well without much settings.
- yapf
  - It seems to be effective if you want to make detailed settings for the format.

# testing framework

- unit test
  - included in the standard library
  - This alone is powerful enough
- pytest
  - Offers more features than unittest and allows for flexible testing
  - I think parametrize is useful...

# 5 tools to solve problems at work

仮想環境

パッケージ  
管理

Lint

Formatter

テスト  
フレームワーク



# 5 tools to solve problems at work

It's hard to know which tool to choose.  
It varies depending on the environment,  
but Make it easy for continuous development

# Other tools you should know

エディタ

ログ管理

API  
ドキュメント

ロードテスト  
ツール

セキュリティ  
テストツール

APMツール

# Role of tools

- Editor: code reading, code writing
- Log management: Proper log output to facilitate investigation
- API documentation: Indicates API I/F and improves maintainability
- Load testing tools: Reproduce the expected load and demonstrate the required performance
- Security testing tools: Detect insecure code
- APM Tools: Detect Bottlenecks

# Supported Python tools

- Editor: Anything you like...
- Log management: standard packages, structlog, loguru, etc.
- API documentation: Swagger, Sphinx, etc...
- Road test tool: Locust
- Security testing tools: Bandit, Safety
- APM tools: New Relic SDK, etc.

This time due to time  
constraints  
I'll skip the explanation



# PyCharmありがとう！



- Typo発見に役立つ
- フォーマットGod
- めっちゃいい感じにインデントしてくれる
- てかないと、マジ無理

# I would like to introduce the editor only.

- I love PyCharm Professional
  - <https://www.jetbrains.com/pycharm/>
  - All necessary settings are included by default.
  - The code jump function is surprisingly superior compared to VSCode.
  - There was a time when I thought I could do it with VSCode.
  - Although it is a paid editor, please consider it.
- There's also something for learners called PyCharm for Education
- Focus on the editor and nurture your development environment.

# Color themes and fonts for my environment

- color theme
  - Tokyo Night <https://plugins.jetbrains.com/plugin/18820-tokyo-night-theme>
- font
  - Programming font Hakugen (Hakugen/HackGen)  
<https://github.com/yuru7/HackGen>
  - Programming font Utatane <https://github.com/nv-h/Utatane>
  - Size is about 16, slightly larger



# Other topics that could not be introduced

- Make full use of AI! Get all the code samples
  - GitHub Copilot
  - Chat tools (Copilot Chat, ChatGPT, Bard) Fully utilized in this document
- We have introduced the tools, but what should we do in terms of methods?
  - Use a debugger instead of print debugging
  - Lint and unittest automatically with CI
  - Conduct code reviews and communicate good code
  - Write Type hitting anyway, definitely write DocString too

Advantages and  
disadvantages I felt  
after using Python at  
work for 4 years

# merit

- Really rich library
- You can receive anything with the dict type.
  - I think communication using json is really strong when it comes to increasing and decreasing parameters.
  - If you want to do something a little more rigid, there are also options like TypedDict and dataclass.
    - (Actually, I've been adding more codes like that recently.)
- Indentation allows you to read syntax properly
- There is a lot of code assistance and reference information, and there are many people who can read and write.
- (It's easy to think of algorithms in natural language using your own senses)

# Demerit

- Sometimes it's too loose for the type
  - What's in this dict...! (I spend the most time in my life)
- Deep indentation puts too much stress on the brain; indenting 5 times is impossible.
  - Nesting such as class, method, for, for, if is quite strict.
- It is difficult to master Python-like writing style.
  - I also feel that I used to write in a Java or C++ style.
- I think it's very difficult to design.
  - There are also type issues, rewriting class variables, instance variables from outside, etc...

# Against the disadvantages

- Sometimes the type is too loose → Use TypedDict, dataclass, etc.
- If the indentation is too deep, the load on the brain is too high. → Measure complexity with a tool, or split when it becomes a for statement. Make it smaller anyway.
- It is difficult to master Python-like writing style → Read the library code (read good code)
- I think it's really difficult to design... ← This is really difficult.

# Do you go into the design?

- Maybe it would be a good idea to experience not only Python but also the culture of other languages?
  - There's a lot to learn from PHP
  - Until last year, I worked as a PHP engineer, and I have a lot to learn about the know-how and design of PHP, a dynamically typed language that has supported web development for many years.
    - Python also has many similarities with dynamic typing and type hinting retrofits.
- There are many practices that allow you to continue developing at high speed and in an operational manner, such as domain-driven design, agile development, etc., which you should acquire.
- (Please tell me a book that is completely designed using Django)

# A good book not only about Python

- (Sorry, Those articles are in Japanese)
- Agile Practice Guidebook Practical knowledge of development techniques for achieving results in teams
  - <https://www.shoeisha.co.jp/book/detail/9784798176826>
- Guidelines for easy-to-read code – for sustainable software development
  - <https://gihyo.jp/book/2022/978-4-297-13036-7>
- Introduction to design through good code/bad code – How to write code that is easy to maintain and continues to grow
  - <https://gihyo.jp/book/2022/978-4-297-12783-1>
- “Clean Agile Back to Basics”
  - <https://www.kadokawa.co.jp/product/302007001102/>

Write test code!

Anyway, the part where the shape is loose  
is Cover it with automated tests.

That is, dynamically typed languages.

It's a golden rule.



Something else I wanted to talk about  
There are many,  
Let's talk at the social gathering!

# summary

- There are barriers between development by one person and development in a team.
- When developing in a team, make arrangements so that development can be done in the same environment.
  - Virtual environment, Linter, Formatter, unittest tools, CI/CD
- Be familiar with the characteristics of the Python language itself
  - It's a dynamically typed language, the GIL, some things you shouldn't do, and multiple pieces of code that accomplish similar goals.
- Let's learn a lot from other language areas
  - PHP for web, Java for design, etc.

# Usage information

- Slide creation: Google Slide <https://www.google.com/slides/about/>
- Font: BIZ UDGothic <https://fonts.google.com/specimen/BIZ+UDGothic>