Python 3.9 時代の
型安全な Python の極め方

小笠原みつき / @yamitzky

JXPRESS

Python ~~3.9~~ 3.10 時代の 型安全な Python の極め方

小笠原みつき / @yamitzky

JXPRESS

Agenda

JXPRESS

1. 導入
2. 基本編: 型ヒントの概要と基本文法
3. 応用編
    1. 型チェック
    2. 高度な型でモデリング
    3. ランタイムの型ヒント
4. まとめ

1. 今日の目標

JXPRESS

Q. 普段、Python で型書きますか？

型を書けることを知らなかった

あまり型を書かない

← 「型書きたい!」

基本的な型を書くようにしている ← 「これも使ってみよう!」

ジェネリクスなど、型を使いこなしている ← 「知らんかった!」

23票・最終結果

1. 今日の目標

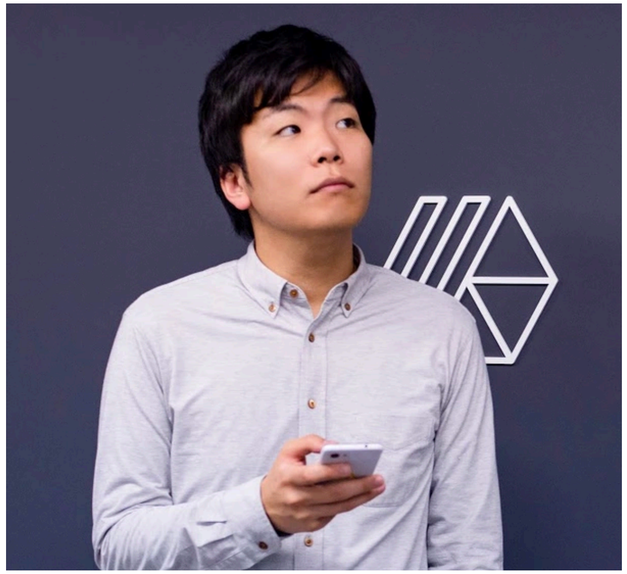
JXPRESS

話さないこと

- 3.8 時代に出番がないもの
- 継承の複雑な話（mroとか）
- 文法*だけ*

1. お前、誰よ？

JXPRESS



- 小笠原 光貴 / @yamitzky
- JX通信社 取締役CDO
- 好きな技術:
  TypeScript、Python(2013年〜)
  サーバーレス、GraphQL etc…

※ TSロゴ: CC-BY 4.0 Microsoft

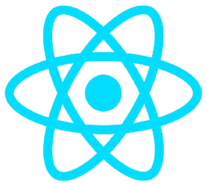
1. JX通信社について

JXPRESS

JX通信社

News Digest
ニュースダイジェスト

XWire
FASTALERT



- 「今起きていることを明らかにする」報道テクノロジーベンチャー
- PyCon JP 2020 Silver Sponsor
- もちろん「型」書いてます!

1. Python の不満

JXPRESS

Python のチーム開発や

大規模な Python プロジェクト

不満がありませんか？

1. Python の不満

JXPRESS

```
AttributeError                            Traceback (most recent call last)
<ipython-input-6-10ebb3994871> in <module>
----> 1 text.lower()

AttributeError: 'NoneType' object has no attribute 'lower'
```

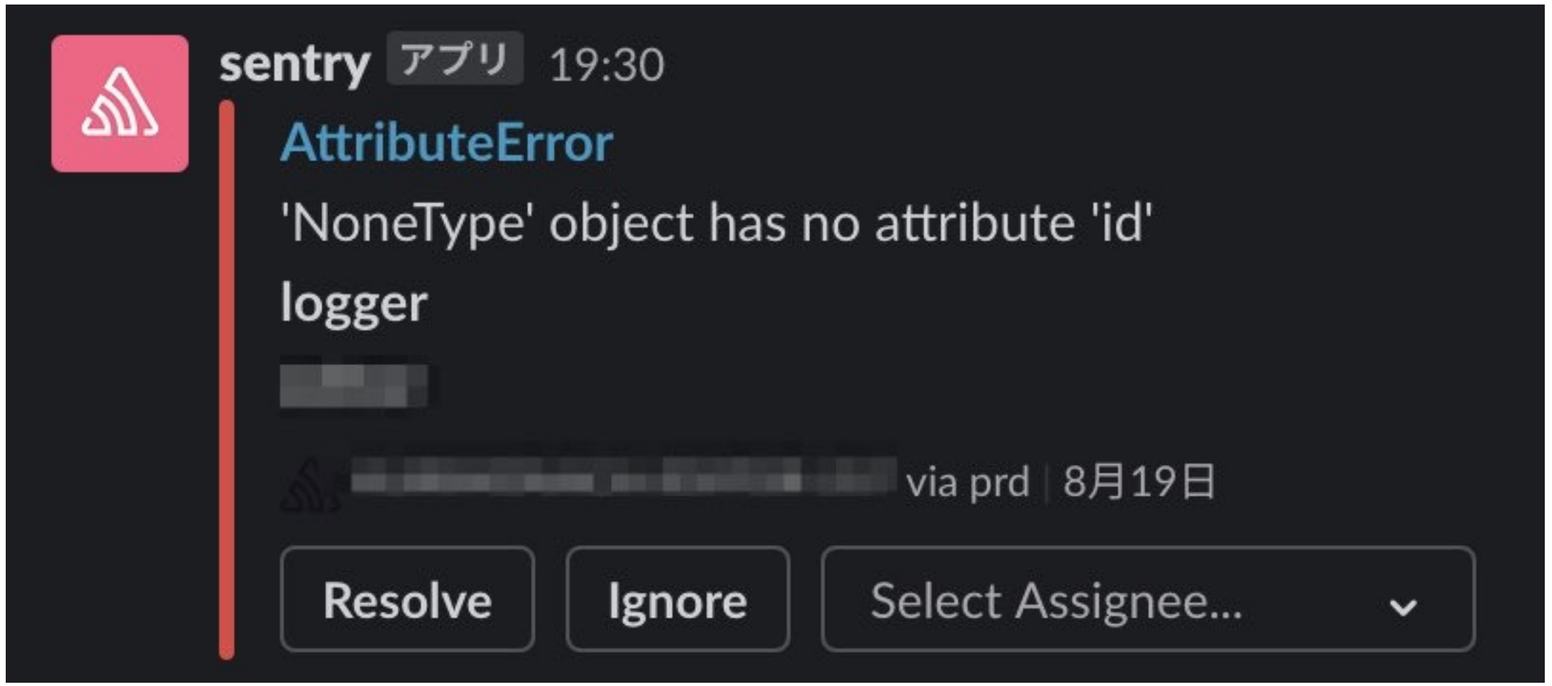
```
KeyError                                  Traceback (most recent call last)
<ipython-input-15-91480d94728d> in <module>
----> 1 user['name']

KeyError: 'name'
```

1. Python の不満

JXPRESS

※弊社某プロジェクトにもありました🤦‍♀️



JXPRESS

エディタの補完

```
def convert_text(text):
    return text.
```

```
def convert_text(text: str):
    return text.
```

- capitalize
- casefold
- center
- count

1. Python の不満

JXPRESS

API通信で返ってくる内容がよくわからない🤔

```
result = requests.get('http://.../hoge.json').json()
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback
<ipython-input-22-07e351311344> in <module>
----> 1 result['count']

KeyError: 'count'
```

1. Python の不満

JXPRESS

動的型付き言語の不満

「型」で解決できます💡

JXPRESS

02

基本編

Python の型の概要と基本文法

JXPRESS

Python の型ってなんだろう

- 正式名称は「Type Hint(型ヒント)」
- PEP 484 で Python3.5(2015〜) から入った文法
- 目的: 静的解析、リファクタリング、コード生成など
- Python が静的型付き言語になったわけではなく、
  任意・後付けの型システム

※ このスライドで「型」と言及する際は「型ヒント」のことを言います

JXPRESS

静的解析

```
"str" has no attribute "lowr"; maybe "lower"? mypy(error)
問題を表示 (⌥F8)    利用できるクイックフィックスはありません
text.lowr()
```

リファクタリング

```
class User:
    id: str


user = User()
user.id
```

```
class User:
    email: str


user = User()
user.email
```

JXPRESS

Python は「静的型付き言語」ではない

Q1: このコードは実行時エラーになる？ 🤔

```
text: str = 42
```

Q2: 型の情報は str？ int？ 🤔

```
print(type(text))
```

JXPRESS

Python は「静的型付き言語」ではない

A1: 実行時にはエラーは発生しない 🤦

A2: type()の戻り値としては型ヒントは無関係

サードパーティーのツールを組み合わせて活用 👍

2. 「型」の概要

JXPRESS

Python の「型」を取り巻くサードパーティー

参考: https://github.com/typeddjango/awesome-python-typing

JXPRESS

ここまでのまとめ

・正式名「Type Hint」を紹介

・Python は動的型付き言語であり、型ヒントは強制されない

・Python 自体は型チェックを行わない。3rd party ツールを組み合わせる

JXPRESS

変数

```
text = '文字列'

age = 100

deleted = False
```

```
text: str = '文字列'

age: int = 100

deleted: bool = False
```

2. 型ヒントの文法: 関数

JXPRESS

関数

```
def is_around_30(age):
    return age >= 27 and age <= 33
```

```
def is_around_30(age: int) -> bool:
    return age >= 27 and age <= 33
```

2. 型ヒントの文法: 関数

JXPRESS

引数だけ書いてもOK 🤷

```
def is_around_30(age: int):
    return age >= 27 and age <= 33
```

戻り値だけ書いてもOK 🤷

```
def is_around_30(age) -> bool:
    return age >= 27 and age <= 33
```

JXPRESS

Exercise 💪

次のコードを型チェックツールで解析すると…？

```
def is_around_30(age: int):
    return age >= 27 and age <= 33


is_around_30(29)
```

もちろんOK 👌

JXPRESS

Exercise 💪

次のコードを型チェックツールで解析すると…？

```
def is_around_30(age: int):
    return age >= 27 and age <= 33


is_around_30("文字列を入れる")
```

もちろんNG 🙅

2. 型ヒントの文法

JXPRESS

「型」のエラーは

赤波線で表現します

2. 型ヒントの文法: class

JXPRESS

class

```
class User:
    name: str
    age: int
    def __init__(self, name: str, age: int):
        self.name = name
        self.age = age

def is_around_30(user: User):
    return user.age >= 27 and user.age <= 33

user = User("Batman", 29)
is_around_30(user)
is_around_30("xxx")
```

← プロパティの定義

← メソッド定義

← 変数の型として利用

JXPRESS

継承の扱い

```python
class Animal:
    name: str
    age: int

class Dog(Animal):
    ...

class Tomato:
    ...
```

```python
animal: Animal = Animal()
```

↓派生クラスは基底クラスに代入できる👌

```python
dog: Animal = Dog()
```

↓継承してないともちろんダメ 🙅

```python
tomato_is_not_animal: Animal = Tomato()
```

JXPRESS

プロパティの型がわからないとき

```python
from typing import Any


class User:
    name: str
    age: int
    x: Any
```

2. 型ヒントの文法: Any

JXPRESS

暗黙的な Any

```
def is_around_30(age):
    ...
```

≒

```
from typing import Any


def is_around_30(age: Any) -> Any:
    ...
```

2. 型ヒントの文法: データ構造

JXPRESS

list, dict, set

```
users = []
kvs = {'key': 'value'}
id_set = {1, 2, 3}
```

```
users: list[User] = []
kvs: dict[str, str] = {'k': 'v'}
id_set: set[int] = {1, 2, 3}
```

2. 型ヒントの文法: 3.8までのデータ構造

JXPRESS

～Python 3.8

```
from typing import List, Set, Dict


users: List[User] = []
kvs: Dict[str, str] = {'k': 'v'}
id_set: Set[int] = {1, 2, 3}
```

Python 3.9～

※現在 RC 版で正式リリースは2020/10予定

```
users: list[User] = []
kvs: dict[str, str] = {'k': 'v'}
id_set: set[int] = {1, 2, 3}
```

2. 自分でデータ構造を作りたいとき

JXPRESS

Generics（※今日は紹介しません）

List[T] などの [ ] の部分を汎用的にする

```python
from typing import Generic, TypeVar, List

T = TypeVar('T')

# Familiy[User] や Family[Dog] 型になる
class Family(Generic[T]):
    children: List[T]
    def __init__(self, children: List[T]):
        self.children = children

# 関数にも使える
def double_list(li: List[T]) -> List[T]:
    return li * 2
```

2. 型ヒントの文法: Optional

JXPRESS

None が入るかもしれない変数(“ぬるぽ”防止)

```python
from typing import Optional


def find_user(id: int) -> Optional[User]:
    ...


user = find_user(3)
print(user.name)  # user は None かも


if user:
    print(user.name)
```

← Noneの可能性の指摘

← None じゃないことを確かめると
安全に使える

2. 型ヒントの文法: Union

JXPRESS

数字 ＝ int または float

```
def half(num: int | float) -> float:
    return num / 2.0
```

Optional[User] ＝ None または User

```
def find_user(id: int) -> Optional[User]:
    ...
```

＝

```
def find_user(id: int) -> str | None:
    ...
```

2. 型ヒントの文法: Union (3.8, 3.9)

JXPRESS

～Python 3.9

```python
from typing import Union


def half(num: Union[int, float]) -> float:
    return num / 2.0
```

Python 3.10

※ 来年(?)リリース予定

```python
def half(num: int | float) -> float:
    return num / 2.0
```

JXPRESS

ここまでのまとめ

- 変数、関数
- class、継承
- list、dict、set
- Optional
- Union
- Any

JXPRESS

03

応用編

なぜ「型」を極めるのか？

JXPRESS

「型」を極めると何ができるのか？

- 意図しない実行時エラーを事前に防げる
- さらに型安全にしたり、正しくモデリングできる
- プログラムを DRY にできる

JXPRESS

03

応用編1

Pythonプログラムを型安全にしてみよう

JXPRESS

Python の型ヒントの原則

・Python は静的型付き言語ではない

・Python 自体は型チェックをしない

→ ツール「mypy」で型チェック

JXPRESS

mypy

・Python の静的な型チェックをするツール
・GitHub の python/mypy にあり、
Guido もコアチームメンバー
・CLI としても使えるし、 VSCode や
PyCharm とも連携

3-1. mypy

JXPRESS

```
a: int = ''
```

```
(variable) a: Literal['']
```

```
Incompatible types in assignment (expression has type "str", variable has type
"int") mypy(error)
```

← これです

Peek Problem (⌥F8) No quick fixes available

```
a: int = ''
```

JXPRESS

mypy の使い方

pip install mypy して

mypy [ディレクトリ または ファイル名.py]

```
$ mypy .
test.py:106: error: Item "None" of "Optional[User]" has no attribute "name"
Found 1 error in 1 file (checked 1 source file)
```

JXPRESS

GitLab CI の例

```yaml
type_check:
  stage: test
  image: python:3.8-slim
  script:
    - pip install mypy==0.782
    - mypy worker
    # 例）今回のソースは「worker」ディレクトリ
```

アップデートで突然賢くなって
突然 CI が通らなくなるので
バージョン固定を強く推奨

3-1. mypy のつらいところ

JXPRESS

ライブラリに型がないとエラーになる😱

```
$ mypy src
src/libtest.py:5: error: Skipping analyzing 'boto3': found module but no type hints
or library stubs
src/libtest.py:5: note: See https://mypy.readthedocs.io/en/latest/running_mypy.html#
missing-imports
Found 1 error in 1 file (checked 2 source files)
```

訳: “boto3” には型ヒントないよ

※ 型ヒントのないライブラリは結構多い

3-1. 型ヒントのないライブラリ

JXPRESS

どうする？ 🤔

- スタブを生成する/書く
- スタブパッケージをインストールする/作る

JXPRESS

stub とは

・プログラム/ライブラリに外付けの型定義
・*.pyi という名前で、型の定義だけ記載
・第三者が作ってパッケージ化できる

※ TypeScript の「.d.ts」「@types/***」

JXPRESS

stub の例

型のないプログラム

```
age = 29


def is_around_30(age):
    return age >= 28 and age <= 32


print(is_around_30(age))
```

hoge.py

対応するスタブ

```
age: int


def is_around_30(age: int) -> bool: ...
```

hoge.pyi

JXPRESS

stubgen

- mypy と一緒についてくる自動生成ツール
- stubgen [ディレクトリ or ファイル名]

```
$ ls src
libtest.py      stubtest.py     test.py
$ stubgen src
Processed 3 modules
Generated files under out/
$ ls out
libtest.pyi     stubtest.pyi    test.pyi
```

対応するファイルが生成

3-1. スタブを書く(自動生成)

JXPRESS

stubgen

- stubgen -m [モジュール] で外部ライブラリも
- 完全ではないので、手動修正おすすめ

```
import logging
from boto3.session import Session as Session
from typing import Any, Optional

DEFAULT_SESSION: Any

def setup_default_session(**kwargs: Any) -> None: ...
def set_stream_logger(name: str = ..., level: Any = ..., format_string: Optional[Any] =
def client(*args: Any, **kwargs: Any): ...
def resource(*args: Any, **kwargs: Any): ...
```

※ 実際、2行目を削除しないと mypy は動かない

3-1. スタブパッケージのインストール

JXPRESS

pip install ***-stubs

- 作ったスタブは pypi パッケージにできる
- OSS として第三者によって公開されている
- 例）boto3-stubs、django-stubs、etc…

※ 詳細は PEP 561

JXPRESS

どうする？🤔

- スタブを生成する/書く
- スタブパッケージをインストールする/作る
- **諦める**

3-1. ライブラリの型を諦める

JXPRESS

ときには、あきらめが肝心

- --ignore-missing-imports を指定する
- import に # type: ignore のコメントをつける
- 実際には関数などでラップしてAnyの範囲を広げないのが大事

3-1. mypy 以外の型チェッカー

JXPRESS

mypy 以外にも結構ある

- microsoft/pyright
- google/pytype
- facebook/pyre

3-1. 型チェッカーの比較(一例)

JXPRESS

mypy

return 漏れるとエラー

```
def check(name: str) -> bool:
    # return わすれ
    name == 'Bruce'
```

pyright/pytype

戻り値の方も推論

```
def return_str():
    return "PyCon JP"

def runtime_error():
    return return_str() + 2020
```

pyre

プロパティの初期化チェック

```
class User:
    name: str

    def __init__(self):
        # 初期化忘れ
        pass
```

※ 設定で厳しく／緩くできる

JXPRESS

型は推論されるので

型ヒントを書かなくても(一部)型安全に

```python
# pyright の場合
import random


def random_str():  # 戻り値は str と推論
    seed = 'abcdefg'
    return seed[random.randint(0, 6)]


numeric = random.randint(0, 100)  # int と推論
numeric + random_str()  # NG
```

※ 引数、戻り値、クラスプロパティの型は書くのがおすすめ

JXPRESS

03

応用編2

もっと型安全で、もっと良いモデリングをしよう

JXPRESS

型ヒントの公式機能の紹介

TypedDict

Protocol

NewType

Literal

JXPRESS

ユースケース①

「API のレスポンスを型定義したい」

```python
import requests


response = requests.get('http://.../user/me.json')
user = response.json()
# レスポンスはだいたい dict
# {
#   "name": "Bruce Wayne"
# }
```

JXPRESS

Dict のまま引き回すと起こること😱

```
---------------------------------------------------------------------------
KeyError                                                  Traceback
<ipython-input-22-07e351311344> in <module>
----> 1 result['count']

KeyError: 'count'
```

→ KeyError を型チェックしたい

JXPRESS

TypedDict

```python
from typing import TypedDict


class User(TypedDict):
    name: str


response = requests.get('http://.../user/me.json')
user: User = response.json()
user['name']      # OK
user['name'] + 1  # NG: 値の型も見る
user['aaa']       # NG: 存在しないキーはエラー
user.name         # NG: class のようには使えない
```

※ . でアクセスはできない＝ class に詰め替えたほうが良い場合も

JXPRESS

ユースケース②

抽象に依存する設計にしたい

```
# 例）🦆が飛ぶシミュレーションゲーム
def play(duck): ←適切な型は？
    duck.fly()
```

3-2. 抽象に依存した設計

JXPRESS

よくない例 🙅

```
def play(duck: Duck):  # ハトを追加したくなったら？
    duck.fly()
```

抽象に依存した例 👌

```
def play(bird: Bird):  # ハトでも動く！
    bird.fly()
```

JXPRESS

抽象的な「Bird」を定義してみよう

例:「抽象クラス」をつかう

```python
from abc import ABC, abstractmethod


class Bird(ABC):  # 抽象基底クラス
    @abstractmethod
    def fly(self):
        ...

class Duck(Bird):  # 継承
    def fly(self):
        ...
```

3-2. 抽象クラスの問題点

JXPRESS

型安全にはなったが…

元のバージョン

```
def play(bird):
    bird.fly()


class Duck:
    def fly(self):
        ...
play(Duck())
```

抽象クラス版

```
def play(bird: Bird):
    bird.fly()


class Duck(Bird):
    def fly(self):
        ...
play(Duck())
```

継承＝ランタイムの挙動変化が必要

動的型付き言語の柔軟性が失われ、Pythonic でない

3-2. Protocol による抽象化

JXPRESS

抽象クラス

名前的部分型

「class を継承しているか」

Protocol

構造的部分型

「必要なメソッドを持っているか」

3-2. Protocol による抽象化

JXPRESS

```python
def play(bird):
    bird.fly()


class Duck:
    def fly(self):
        ...
play(Duck())
```

Protocol

```python
# 抽象を定義する側
from typing import Protocol
class Flyable(Protocol):
    def fly(self):
        ...
def play(bird: Flyable):
    bird.fly()


# 呼び出し側
class Duck:           # 継承していない
    def fly(self):  # fly を持ってるからOK
        ...
play(Duck())
```

型安全かつ Duck の挙動が変わらない 😏

JXPRESS

ユースケース③

List や Set など「in」できるものを受け取りたい

```
# よくない例: list や set 以外だとダメ？
def contains_batman(heroes: list[str] | set[str]):
    return 'Bruce Wayne' in heroes
```

JXPRESS

公式で Protocol が定義されている

※ Container ＝ __contains__ があるモノ

```
from typing import Container


def contains_batman(heroes: Container[str]):
    return 'Bruce Wayne' in heroes


contains_batman(list())
contains_batman(set())
contains_batman(dict())
contains_batman(dict())
contains_batman(YourAwesomeStructure())
```

3-2. 定義済みの Protocol

JXPRESS

その他の公式 Protocol

ABC	Inherits from	Abstract Methods	Mixin Methods
Container		__contains__	
Hashable		__hash__	
Iterable		__iter__	
Iterator	Iterable	__next__	__iter__
Reversible	Iterable	__reversed__	
Generator	Iterator	send, throw	close, __iter__, __next__
Sized		__len__	
Callable		__call__	
Collection	Sized, Iterable, Container	__contains__, __iter__, __len__	
Sequence	Reversible, Collection	__getitem__, __len__	__contains__, __iter__, __reversed__, index, and count
MutableSequence	Sequence	__getitem__, __setitem__, __delitem__, __len__, insert	Inherited Sequence methods and append, reverse, extend, pop, remove, and __iadd__
ByteString	Sequence	__getitem__, __len__	Inherited Sequence methods
Set	Collection	__contains__, __iter__, __len__	__le__, __lt__, __eq__, __ne__, __gt__, __ge__, __and__, __or__, __sub__, __xor__, and isdisjoint
MutableSet	Set	__contains__, __iter__, __len__, add, discard	Inherited Set methods and clear, pop, remove, __ior__, __iand__, __ixor__, and __isub__
Mapping	Collection	__getitem__, __iter__, __len__	__contains__, keys, items, values, get, __eq__, and __ne__
MutableMapping	Mapping	__getitem__, __setitem__, __delitem__, __iter__, __len__	Inherited Mapping methods and pop, popitem, clear, update, and setdefault
MappingView	Sized		__len__
ItemsView	MappingView, Set		__contains__, __iter__
KeysView	MappingView, Set		__contains__, __iter__
ValuesView	MappingView, Collection		__contains__, __iter__
Awaitable		__await__	
Coroutine	Awaitable	send, throw	close
AsyncIterable		__aiter__	
AsyncIterator	AsyncIterable	__anext__	__aiter__
AsyncGenerator	AsyncIterator	asend, athrow	aclose, __aiter__, __anext__

https://docs.python.org/ja/3/library/collections.abc.html

※ 必ずしもProtocolを継承していない

3-2. 定義済みの Protocol

JXPRESS

Callable
関数っぽいもの(__call__)

```python
from typing import Callable

def func():
    pass

class Func:
    def __call__(self):
        pass

callback: Callable = func
callback = Func()
callback = lambda x: x * 2
```

Iterable
for in できるもの(__iter__)

```python
from typing import Iterable

def scan_items(it: Iterable):
    for item in it:
        print(item)

class MyIterable:
    def __iter__(self):
        ...

scan_items([])
scan_items({})
scan_items(MyIterable())
```

Sequence
破壊的変更できない list

```python
from typing import Sequence

def scan_items(immutable_list: Sequence = []):
    for item in immutable_list:
        print(item)
    immutable_list.append('破壊的変更')  # NG

scan_items([])
```

（参考）Protocol を活用した Immutable なデータ構造について

https://yamitzky.hatenablog.com/entry/2020/08/23/183852

JXPRESS

ユースケース④

int や str に「意味」を持たせたい

```
class Car:
    id: int


class User:
    id: int
    prefecture: str
```

```
# 同じ int でも代入できるべきでない
car.id = user.id


# 存在しない都道府県が代入できるべきでない
user.prefecture = 'ちばらき県'
```

3-2. 特殊なプリミティブ型

JXPRESS

NewType

```python
class Car:
    id: int

class User:
    id: int
```

```python
from typing import NewType
CarId = NewType('CarId', int)
UserId = NewType('UserId', int)

class Car:
    id: CarId
class User:
    id: UserId
```

```python
# 同じ int でも代入できるべきでない
car.id = user.id
```

```python
# 同じ int でも代入できるべきでない
car.id = user.id
# ランタイムでは int だが、型は CarId
car.id = CarId(300)
print(type(car.id))  # int
```

キリがないので、ID など紛らわしい一部の型だけ推奨

JXPRESS

Literal

```python
from typing import Literal

Prefecture = Literal['北海道', '青森', '秋田']  # 略

prefecture: Prefecture = '北海道'
prefecture = '北海道道道道道道道道道'  # typo
```

JXPRESS

紹介したもの

- TypedDict
- Protocol
- Iterableなどの公式 Protocol
- NewType、Literal

JXPRESS

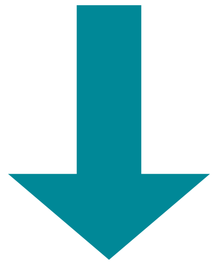
03

応用編3

ランタイムをもっと便利にする

JXPRESS

今まで：静的な型チェックの話



実行時(ランタイム)に

型の情報は取得できる

3-3. 実行時の型の情報取得

JXPRESS

こんなプログラムを書いて実行してみる

```
class User:
    name: str
    age: int

print(User.__annotations__)
```

```
{'name': <class 'str'>, 'age': <class 'int'>}
```

実行時に __annotations__ に格納された型の情報を活用できる！

※ get_type_hints() 使ったほうが良いです

例）dict への変換

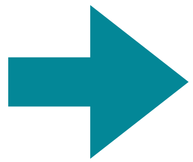
```
def serialize(user: User) -> dict:
    return {
        prop: getattr(user, prop)
        # 型ヒントにはプロパティ一覧が含まれる
        for prop in User.__annotations__
    }
```

JXPRESS

例）dataclass

```python
class User:
    name: str
    age: int
    def __init__(self, name: str, age: int):
        self.name = name
        self.age = age

user = User(name='mitsuki', age=28)
```



```python
from dataclasses import dataclass

@dataclass
class User:
    name: str
    age: int

user = User(name='mitsuki', age=28)
```

@dataclass デコレーターをつかうと

実行時に __init__ を自動で定義してくれる

JXPRESS

例）pydantic

≒ dataclass ＋バリデーション＋シリアライズ

```python
from pydantic import BaseModel, PositiveInt


class User(BaseModel):
    name: str
    age: PositiveInt  # 年齢は0歳以上


User(name='マイナス1歳さん', age=-1)  # OK???
```

```
ValidationError: 1 validation error for User
age
  ensure this value is greater than 0 (type=value_error.number.not_gt; limit_value=0)
```

JXPRESS

例）FastAPI

```python
from fastapi import FastAPI
from pydantic import BaseModel
app = FastAPI()

class User(BaseModel):
    id: int
    name: str

@app.get("/users/{id}")
def user(id: int) -> User:
    return User(id=id, name="Kara Zor-El")
```

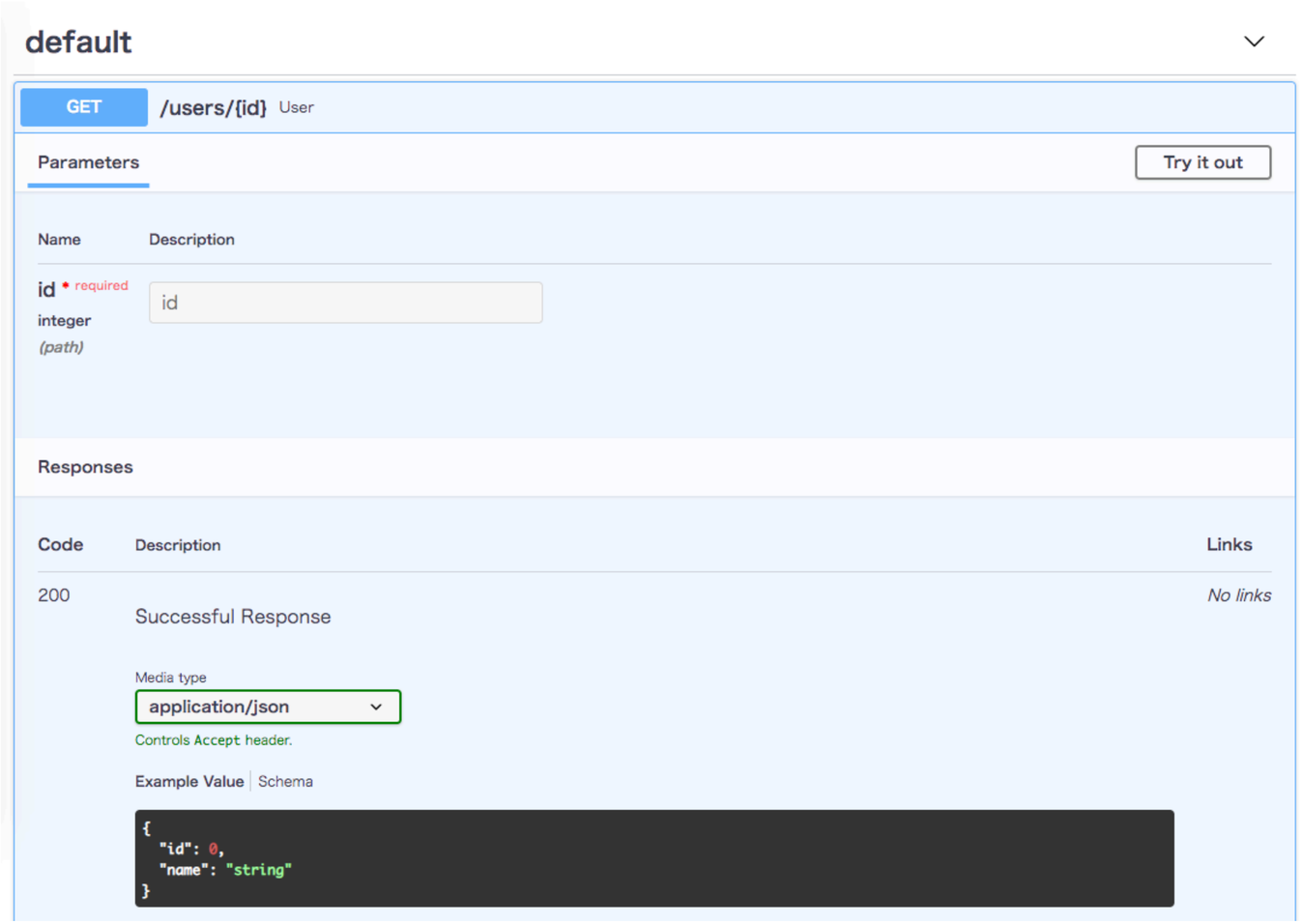
```
~ $ http localhost:8000/users/abc
HTTP/1.1 422 Unprocessable Entity
content-length: 99
content-type: application/json
date: Wed, 26 Aug 2020 17:41:52 GMT
server: uvicorn

{
    "detail": [
        {
            "loc": [
                "path",
                "id"
            ],
            "msg": "value is not a valid integer",
            "type": "type_error.integer"
        }
    ]
}
```

型ヒント付きの関数を定義していくだけで、
バリデーションつきの API ができる

JXPRESS

型ヒントから Swagger（OpenAPI）も自動生成



※ 他言語でも使われる API 仕様。API の型が定義できる

JXPRESS

ランタイムでの型ヒントを紹介

・__annotations__ に型の情報が含まれる

・デコレーター、メタクラスと組み合わせるとDRYに

・型の動的チェック、API のバリデーション、Swagger 定義などもできる

JXPRESS

04 まとめ

4. まとめ

JXPRESS

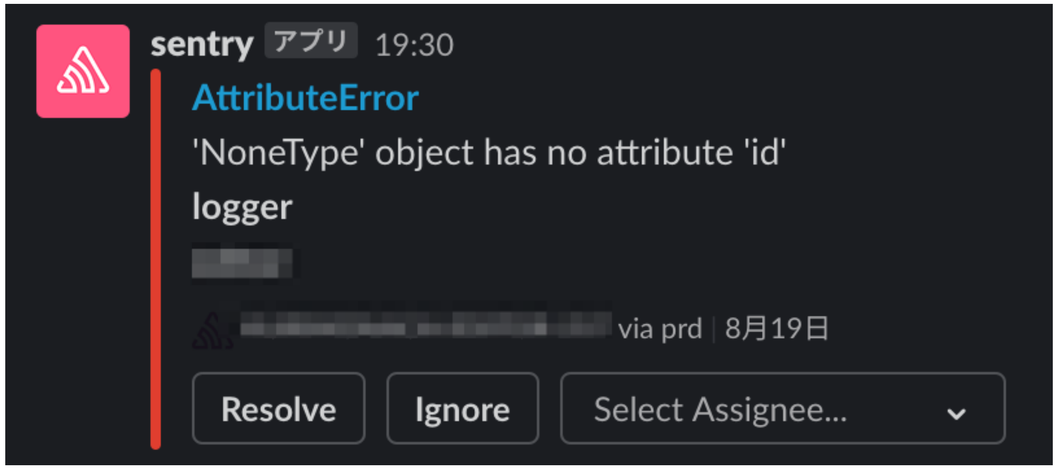
型ヒントを書くと None や
型のミスをチェックできる

```python
from typing import Optional

def find_user(id: int) -> Optional[User]:
    ...

user = find_user(3)
print(user.name)  # user は None かも

if user:
    print(user.name)
```



JXPRESS

mypy や pyright で型チェック

一部は推論される＝書いてないのに型安全

```python
# pyright の場合
import random


def random_str():  # 戻り値は str と推論
    seed = 'abcdefg'
    return seed[random.randint(0, 6)]


numeric = random.randint(0, 100)  # int と推論
numeric + random_str()  # NG
```

4. まとめ

JXPRESS

Protocol や TypedDict で「継承」に縛られない型安全

```python
from typing import TypedDict


class User(TypedDict):
    name: str


response = requests.get('http://.../user/me.json')
user: User = response.json()
user['name']      # OK
user['name'] + 1  # NG: 値の型も見る
user['aaa']       # NG: 存在しないキーはエラー
user.name         # NG: class のようには使えない
```

```python
from typing import Sequence


def scan_items(immutable_list: Sequence = []):
    for item in immutable_list:
        print(item)
    immutable_list.append('破壊的変更')  # NG


scan_items([])
```

JXPRESS

「型」を書くだけでランタイムが便利に

```python
from dataclasses import dataclass

@dataclass
class User:
    name: str
    age: int

user = User(name='mitsuki', age=28)
```

```python
from fastapi import FastAPI
from pydantic import BaseModel
app = FastAPI()

class User(BaseModel):
    id: int
    name: str

@app.get("/users/{id}")
def user(id: int) -> User:
    return User(id=id, name="Kara Zor-El")
```

JXPRESS

3.9/3.10 時代、
なぜ動的型付き言語の Python で
「型」を極めるのか？

4. まとめ

JXPRESS

Python の型ヒントを極めると
動的型付き言語の良さを損なうことなく
安全で Pythonic なプログラムを
生産性高く書けるから

※一意見です。皆様の意見を聞かせてください


Thank you!
JXPRESS