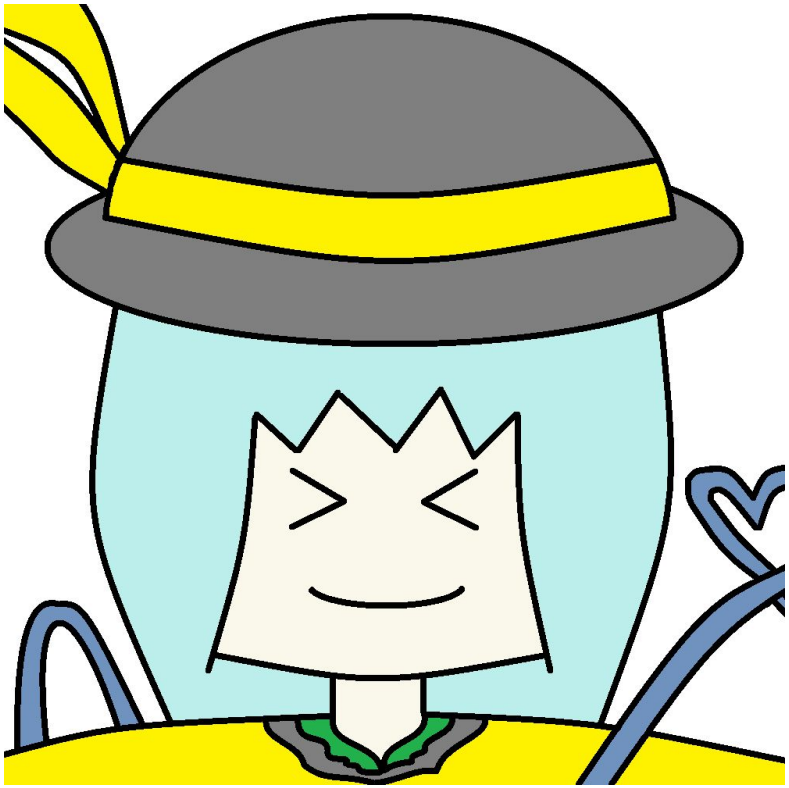
静的型付け言語Python

こんにちは

- @utgw (うたがわ)
- 工学部情報学科

  計算機科学コース2回生
- 第39代副広報・root
- Twitter: @utgwkk
- GitHub: @utgw



アンケートです

- 動的型付け言語を書いたことがある
- Python を書いたことがある
- 型理論に詳しい

前置き

- 型に関する踏み込んだ話はしません(できません)
- あくまで言語機能の紹介としてやっていきます
- Python 3 系の話をします
  - Python 2 系は各位うまくやってくれ
  - https://blog.ymyzk.com/2016/02/python-2-type-hints/

    とかに載ってそう

Pythonの型ヒントの経緯

- 2006/12/2: PEP 3107 - Function Annotations
  - Python 3.0 で関数の型ヒントだけ導入された
  - あくまでもヒント
- 2014/9/29: PEP 484 - Type Hints
  - 変数の型ヒントやジェネリクスなど、より踏み込んだ型ヒントに関する提案
- 2015/9/14: Python 3.5 リリース
  - PEP 484 を実現するモジュール typing が導入された

動的型付け言語に静的型を導入する試み

1. TypeScript の場合
    1.1. TypeScript のコードを書く
    1.2. コンパイラが型チェックをする
    1.3. JavaScript にコンパイルされる
2. Python + 型ヒントの場合
    2.1. Python のコードに型の情報を書く
    2.2. 各位うまくやってくれ！！！！！

各位うまくやってくれ！！！！！

- 各位うまくやっていきましょう
  - 実行時に型チェックがされるわけではない
  - そもそも言語としては動的型付け言語のまま
  - むやみに互換性を壊したくない
- 型チェックツールを使いましょう
  - mypy を使いましょう
  - そのうち PyCharm とかでも対応しそう

動的型付け言語に静的型を導入する試み(修正)

1. TypeScript の場合
    1.1. TypeScript のコードを書く
    1.2. コンパイラが型チェックをする
    1.3. JavaScript にコンパイルされる
2. Python + 型ヒントの場合
    2.1. Python のコードに型の情報を書く
    2.2. mypy 等で型チェックをする
    2.3. よさそうだったら本番投入

従来のコード

```
def fib(n):
    a, b = 0, 1
    i = 0
    while i <= n:
        yield a
        a, b = b, a + b
        i += 1
```

型情報のあるコード

```
from typing import Iterator

def fib(n: int) -> Iterator[int]:
    a, b = 0, 1
    i = 0
    while i <= n:
        yield a
        a, b = b, a + b
        i += 1
```

型情報のあるコード

```
from typing import Iterator

def fib(n: int) -> Iterator[int]:
    a, b = 0, 1
    i = 0
    while i <= n:
        yield a
        a, b = b, a + b
        i += 1
```

どのようにして型情報を記述するのか

- ざっくり見ていきましょう
  - 変数の型ヒント
  - 関数の型ヒント
  - cast
  - Callable
  - ジェネリクス
  - 直和型
  - Optional
  - Any
  - その他

変数の型ヒント

- 変数名 = 値 # type: 型名
- コメントに型名を記述することで型ヒントとする
- 型名の alias が定義できる
  - 別名 = 型名

関数の型ヒント

- `def 関数名(x: 型, y: 型, ...) -> 返り値の型:`
- 引数の既定値は `x: 型 = 値` のように書く

cast

- `cast(キャストする型, 値)`
- 値をある型にキャストしていることを示す
  - 実際にはキャストしない

Callable

- `Callable[[引数の型のリスト, ...], 返り値の型]`
- 呼び出し可能であることを示す

ジェネリクス

- `T = TypeVar('T')`
- `def hoge(val: T) -> T:`
- ジェネリックなクラスが作れる

直和型

- `Union[A, B, ...]`
- A, B, ...のいずれかの型であることを示す
- 複数の型をとりうるが、ジェネリクスほどは自由でないときに使う

Optional

- `Optional(X) = Union[X, None]`
- X もしくは None であることを示す
- 失敗するかもしれない操作に使う？
  - それ例外処理でよくない？

Any

- すべての型の親
- auto 型のような感じで使うのかな
- ~~型ヒント書かなきゃいいじゃん~~

その他

- List[T]: T型のリスト
- Dict[K, V]: K型をキーとしてV型の要素を格納する辞書型
- Iterable[T]: T型の一連の要素を持つ型
- Container: 何らかのコンテナ
- SupportsInt: int(x) という操作ができる型

実演

まとめ

- Python は言語全体として静的型になるつもりはない
- あくまで各位うまくやってくれ
- mypy を使って型検査していこう
  - まだうまくいってない部分もありそう？
  - ぼくの型の指定がバグってたら許してくれ
- 型理論よく分かってないので戸惑った

ありがとうございました

- 質問など
  - 型のことは型に詳しい人に聞いてくれ

参考

- typing — 型ヒントのサポート
  - http://docs.python.jp/3/library/typing.html
- PEP 484
  - https://www.python.org/dev/peps/pep-0484/
- PEP 3107
  - https://www.python.org/dev/peps/pep-3107/
- mypy で静的型付け Python プログラミング
  - http://t2y.hatenablog.jp/entry/2014/12/22/004525