PyCharmのススメ 第1部 Type Hints編
もしくは、Pythonと型アノテーション

K. Yamaguchi

2017-05-19

Pythonと型アノテーション

型アノテーションの歴史

- Python3.0～ （Function Annotations）
  - メソッド定義の引数と返り値の型アノテーションの追加
- Python3.5～ （Type Hints）
  - typing モジュール
    （ジェネリクス/型変数/型エイリアス/Any/Union/Optional/etc ...）
- Python3.6～ （Variable Annotations）
  - 変数アノテーション

Function Annotations （Python3.0）

- 関数の引数と、返り値の型アノテーション

def 関数名(引数1: 型,引数2: 型,...) -> 返り値の型:

```
def zero_fill(n: int, digits: int) -> str:
    return '{{:0{:d}d}}'.format(digits).format(n)
```

- ただしインタプリタは型アノテーションを無視する
- サードパーティ製ツールが必要

- そこで PyCharm の出番ですよ!!

https://www.jetbrains.com/pycharm/

Function Annotations （Python3.0）

若干の嘘があります

- 返り値の型アノテーションのおかげで
  . を入力した時点で補完候補に型strの候補が出現する

```
def main():
    zero_fill(3, 3).

if __name__ == '__
    main()
```

```
format(self, args, kwargs)                    str
join(self, iterable)                          str
split(self, sep, maxsplit)                    str
startswith(self, prefix, start, end)          str
_abc_cache
_abc_negative_cache
_abc_negative_cache_version
_abc_registry
capitalize(self)                              str
center(self, width, fillchar)                 str
Press ^. to choose the selected (or first) suggestion and insert a dot afterwards >> π
```

- 引数の型アノテーションのおかげで
  警告が出る

```
def main():
    zero_fill(3.0, 3)
```

Expected type 'int', got 'float' instead more... (⌘F4)

Function Annotations （Python3.0）

- 型アノテーションに書けるのは1つの値（式）だけ
- list と書くと list なのは分かるけれど
  そこに何が入っているのかはアノテーション
  できない
- 「int型 か float型」みたいな直和型が書けない
- 現実的にはあまり役に立たない

型アノテーションの歴史

- Python3.0～ （Function Annotations）
  - メソッド定義の引数と返り値の型アノテーションの追加
- Python3.5～ （Type Hints）
  - typing モジュール
    （ジェネリクス/型変数/型エイリアス/Any/Union/Optional/etc …）
- Python3.6～ （Variable Annotations）
  - 変数アノテーション

Type Hints （Python3.5）

- Python3.5でtypingモジュール登場!!
  （ただし暫定モジュール）
  - ジェネリクス/型変数/型エイリアス
  - Any型/Union型/Optional型

- Python3.0 の関数アノテーションで決まっていたのは構文だけで、何をどう書くのか? の仕様はなかった
  - →Python3.5で劇的に改善

Type Hints （Python3.5）

- ジェネリクス

ジェネリック版のlist型ヒント

型指定を [] で括る

```
from typing import List


def q03(s: str) -> List[int]:
    return list(map(lambda x: len(list(filter(str.isalpha, x))), s.split()))


def main():
    pi = q03(
        'Now I need a drink, alcoholic of course, '
        'after the heavy lectures involving quantum mechanics.')
    # => [3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5, 8, 9, 7, 9]
```

- Dict[str, Tuple[int, str, List[Tuple[str, List[int]]]]] 型の変数dがあるとき、d['key'][2][3][0] の型は?

```
d['key'][2][3][0].
__name__ == '__ma
main()
```

```
split(self, sep, maxsplit)          str
format(self, args, kwargs)          str
join(self, iterable)                str
startswith(self, prefix, start, end) str
```

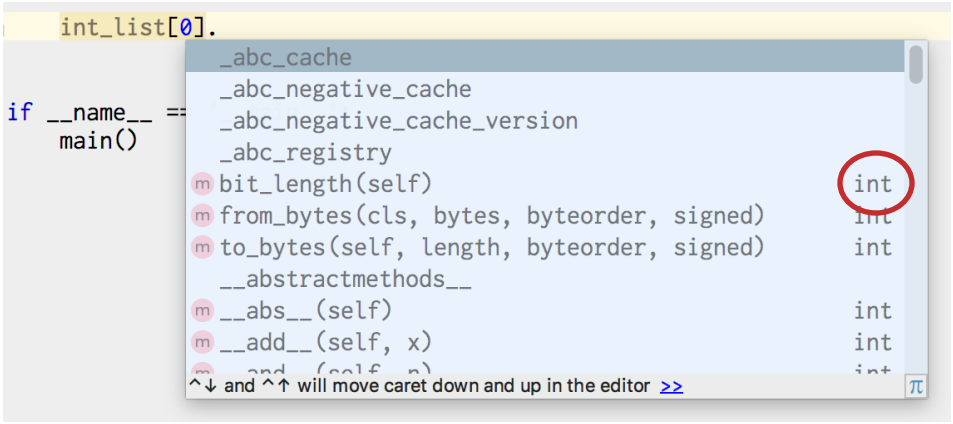
Type Hints （Python3.5）

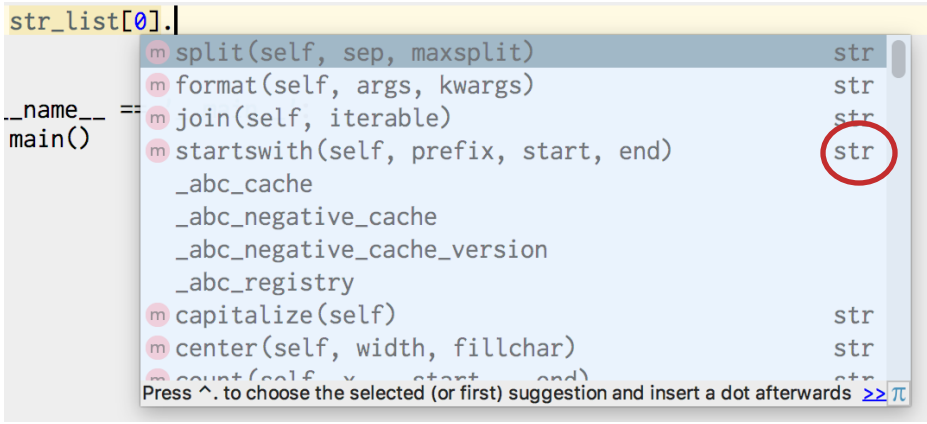
- 型変数

```
from typing import List, TypeVar

T = TypeVar('T')


def even_items(l: List[T]) -> List[T]:
    return [x for i, x in enumerate(l) if i % 2 == 0]


def main():
    int_list = even_items([1, 2, 3, 4, 5])  # => [1, 3, 5]
    str_list = even_items(['a', 'b', 'c'])  # => ['a', 'c']
```

型変数

パラメータ化された型





Type Hints （Python3.5）

- Union型（直和型）
  - どれでもいい

```
def norm(l: List[Union[int, float]]) -> float:
    return math.sqrt(sum(map(lambda x: x ** 2, l)))


def main():
    print(norm([1, 2, 3]))
    print(norm([1.0, 2.0, 3.0]))
    print(norm([1, 2.0, 3]))
    print(norm(['a', 'b']))
```

警告

- Any型
  - なんでもいい
  - object を指定するのは
    object が持っている属性しか
    使えなくなるからダメ

```
from typing import Any


def add_any(x: Any, y: Any):
    return x + y


def add_object(x: object, y: object):
    return x + y
```

警告

- （単純な）generator

```
def repeat(item: T) -> Iterable[T]:
    while 1:
        yield item


def main():
    for s in repeat('a'):
        s.
```

split(self, sep, maxsplit) str
format(self, args, kwargs) str

実は……PyCharmなら昔からできました


```
def q03(s):
    """
    :param str s:
    :rtype: list[int]
    """
    return list(map(lambda x: len(list(filter(str.isalpha, x))), s.split()))


def main():
    pi = q03(
        'Now I need a drink, alcoholic of course, '
        'after the heavy lectures involving quantum mechanics.')
    # => [3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5, 8, 9, 7, 9]

    pi[0].

if __nam
    mai
```

PyCharm独自の
docstringによる型アノテーション

```
_abc_cache
_abc_negative_cache
_abc_negative_cache_version
_abc_registry
m bit_length(self)                                  int
m from_bytes(cls, bytes, byteorder, signed)         int
m to_bytes(self, length, byteorder, signed)         int
__abstractmethods__
m __abs__(self)                                     int
m __add__(self, x)                                  int
Press ^. to choose the selected (or first) suggestion and insert a dot afterwards >> π
```

https://www.jetbrains.com/help/pycharm/2017.1/type-hinting-in-pycharm.html#legacy

実は……PyCharmは単純な型推論もできます


```
def q03(s):
    return list(map(lambda x: len(list(filter(str.isalpha, x))), s.split()))


def main():
    pi = q03(
        'Now I need a drink, alcoholic of course, '
        'after the heavy lectures involving quantum mechanics.')
    # => [3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5, 8, 9, 7, 9]

    pi[0].

if __nam
    mair
```

```
_abc_cache
_abc_negative_cache
_abc_negative_cache_version
_abc_registry
m bit_length(self)                                  int
m from_bytes(cls, bytes, byteorder, signed)         int
m to_bytes(self, length, byteorder, signed)         int
__abstractmethods__
m __abs__(self)                                     int
m __add__(self, x)                                  int
m __and__(self, n)                                  int
Press ^. to choose the selected (or first) suggestion and insert a dot afterwards >> π
```

アノテーションなしで
型推論してる

型アノテーションの歴史

- Python3.0～ （Function Annotations）
  - メソッド定義の引数と返り値の型アノテーションの追加
- Python3.5～ （Type Hints）
  - typing モジュール
    （ジェネリクス/型変数/型エイリアス/Any/Union/Optional/etc …）
- Python3.6～ （Variable Annotations）
  - 変数アノテーション

Variable Annotations（Python3.6）

- 変数の型アノテーション

変数名: 型 = 値
オブジェクト.属性: 型 = 値

```
FOO: int = 1


class Container:
    def __init__(self) -> None:
        super().__init__()
        self.foo: str = ''
        self.bar: int = 0
```

- 一度これを書いてしまうと3.5以前では実行できない
  - 後方互換性を壊してまで書く気はしないなぁ

Variable Annotations（Python3.6）

- クラススタイル NamedTuple 型付き版

```
import json
from typing import List, NamedTuple

DATA = """[[1000, "札幌市", ["中央区", "北区", "東区", "白石区", "豊平区", "南区", "西区", "厚別区",  """ \
       """"手稲区", "清田区"]],[4100, "仙台市", ["青葉区", "宮城野区", "若林区", "太白区", "泉区"]]]"""


class Data(NamedTuple):
    code: int
    name: str
    wards: List[str]


def main():
    data: List[Data] = [Data(*l) for l in json.loads(DATA)]

    data[0].
```

```
class Data(NamedTuple):
    code: int
    name: str
    wards: List[str]
```

```
if __name_
    main()
```

```
_abc_cache
_abc_negative_cache
_abc_negative_cache_version
_abc_registry
f code                                  Data
m count(self, x)                        tuple
m index(self, x)                        tuple
f name                                  Data
f wards                                 Data
__abstractmethods__
m __add__(self, x)                      tuple
^↓ and ^↑ will move caret down and up in the editor >>
```

※タプルの各要素に
添え字ではなくて
名前でアクセス
できるようになる

※他にもいろいろ

Variable Annotations（Python3.6）

- クラススタイル NamedTuple 型付き版

```
import json
from typing import List, NamedTuple

DATA = """[[1000, "札幌市", ["中央区", "北区", "東区", "白石区", "豊平区", "南区", "西区", "厚別区",  """ \
       """"手稲区", "清田区"]],[4100, "仙台市", ["青葉区", "宮城野区", "若林区", "太白区", "泉区"]]]"""


class Data(NamedTuple):
    code: int
    name: str
    wards: List[str]


def main():
    data: List[Data] = [Data(*l) for l in json.loads(DATA)]

    data[0].wards.

if __name__ ==
    main()
```

```
append(self, object)                    list
_abc_cache
_abc_negative_cache
_abc_negative_cache_version
_abc_registry
clear(self)                             list
copy(self)                              list
count(self, object)                     list
extend(self, iterable)                  list
index(self, object, start, stop)        list
Press ^. to choose the selected (or first) suggestion and insert a dot afterwards >>
```