



Pythonによる メタプログラミング

小泉 守義 (moriyoshi)

自己紹介

- ・ 現在フリーエンジニア、会社設立準備中
- ・ BASIC → Assembly → C → … → Python
- ・ Pythonは5年ほど
- ・ PHPのコミッター (2003~)
- ・ なぜ呼ばれたのか

メタプログラミングとは

- メタプログラミングの定義？
 - Metaprogramming is the writing of computer programs that write or manipulate other programs (or themselves) as their data, or that do part of the work at compile time that would otherwise be done at runtime. – *wikipedia:en:Metaprogramming*
- プログラムを生成するプログラム
 - either at compile time or at runtime

これじゃあ...

- 具体的に何かわからないよ！
- どこからがメタプログラミングで、どこからが普通のプログラミングなの？
 - リフレクション、introspection...
- プログラミング言語によっても違うよね！
- 静的言語、スクリプト言語...
- つまり...

**メタプログラミングの定義
は、みんなの心の中にある**

メタプログラミングの道具

- LISP (の仲間) のマクロ
- C / C++のプリプロセッサマクロ
- C++のテンプレートおよびその特殊化
- Javaのリフレクション、動的バイトコード生成・加工
- 動的言語のランタイムでのクラス / メソッド生成
- DSL (ドメイン固有言語)

※個人の感想です

DISCLAIMER: THIS IS A PERSONAL OPINION

メタプログラミングの道具 とされないもの

- JITコンパイラ
- アセンブリのテクニックとしての自己書き換え

※個人の感想です

DISCLAIMER: THIS IS A PERSONAL OPINION

メタプログラミングの何が うれしいのか？

- 根底にはDRY (Don't Repeat Yourself)
- つまり、楽をするためのもの！
- メタプログラミング 자체もまた、楽しい！

Pythonでの メタプログラミングの道具

- exec / eval
- メタクラス
- デコレータ
- クラスや関数の各種メタ情報
- compile() / astモジュール

exec / eval

```
>>> exec("for i in range(0, 5): print(i)")  
0  
1  
2  
3  
4  
>>> eval("1 + 2 * 3")  
7
```

- exec は文を実行する、eval は式を実行する
- Python では文と式の区別が明確

メタクラス

- a type that generates a type
- type クラス

```
>>> Foo = type('Foo', (), {'hello': lambda self, name:\n                      'hello,' + name})\n\n...\n>>> Foo\n<class '__main__.Foo'>\n>>> Foo.__class__\n<class 'type'>
```

メタクラス

```
class FooType(type):
    def __new__(self, name, bases, members):
        members['hello'] = lambda self: self.__hello__ + \
                                self.name()
        return type.__new__(self, name, bases, members)

class Foo(metaclass=FooType):
    __hello__ = 'hello, '
    def name(self):
        return 'world'

    """
    >>> Foo.__class__
    <class '__main__.FooType'>
    >>> Foo().hello()
```

デコレータ

- 関数を生成する関数を呼び出す
- just a syntactic sugar

```
>>> def hello(fn):  
...     def wrap():  
...         return 'hello,' +\  
...                 fn()  
...     return wrap  
...  
>>> @hello  
... def world():  
...     return 'world'  
...  
>>> world()  
hello, world
```

各種メタ情報

- クラス

```
>>> Bar.__mro__
(<class '__main__.Bar'>, <class '__main__.Foo'>, <type
'object'>)
>>> Foo.__subclasses__()
[<class '__main__.Bar'>]
```

compile() / astモジュール

```
>>> import ast
>>> node = compile('1 + 2', '', 'eval', 0x400)
>>> node
<_ast.Expression object at 0x1004ee850>
>>> ast.dump(node)
'Expression(body=BinOp(left=Num(n=1), op=Add(), right=Num
(n=2)))'
>>> eval(compile(node, '', 'eval', 0))
3
```

言語内DSL

- Pythonは言語内DSLが苦手
 - インデントでブロック構造を表す特徴が制約に
- 演算子オーバロードが可能
- いくつか使えそうな言語構造
 - with構文

言語内DSL

```
with reaction_rules:
    # Ligand-receptor binding
    egfr(l,r) + egf(r) <> egfr(l[1],r).egf(r[1]) \
        | MassAction2(1.667e-06, .06)
    # Note changed multiplicity
    # Receptor-aggregation
    egfr(l[1],r) + egfr(l[2],r) <> \
        egfr(l[1],r[3]).egfr(l[2],r[3]) \
        | MassAction2(5.556e-06, .1)

    # Transphosphorylation of egfr by RTK
    egfr(r[1],Y1068(Y)) > egfr(r[1],Y1068(pY)) \
        | MassAction(.5)
    egfr(r[1],Y1148(Y)) > egfr(r[1],Y1148(pY)) \
        | MassAction(.5)
```

化学反応の式を記述

Pythonから別の言語を生成

- Pyjamas (<http://pyjs.org/>)
 - Python から JavaScript を生成
 - UI を Python で記述できる
- など、他にもあるはず

謝辞

- 以下の方にご協力いただきました
- @mopemope (まつばらさん)
- @aodag (あおだぐさん)
- @kozo2 (にしださん)