第5回 Python講座

2017/07/20

1　クラス

Python はオブジェクト指向プログラミング言語であるためクラスが用意されています．早速，非常に簡単なクラスを作成してみましょう．下記のコードを実行すると「Hello Python!!」と出力されます．これは hello メソッドを持つ「SimpleClass」クラスを作成後，「SimpleClass」のインスタンス生成，hello メソッド呼び出しを行なっています．

w5e1.py

```
# coding: utf-8

class SimpleClass:
''' A very simple class.'''

  def hello(self):
    print("Hello Python!!")

c = SimpleClass()
c.hello()
```

1.1 データ属性

C++ では「データメンバ」，Java では「フィールド」にあたるものが「データ属性」です．Python ではローカル変数同様に宣言をする必要がありません．実際に例を見てみましょう．データ属性は「self. 名前」で使用することができます. また,「__init__」メソッドはこのメソッドが定義されているクラスのインスタンス生成時に呼び出されます．これによりインスタンスが適切に初期化されます (C++ や Java のコンストラクタのようなもの).

w5e2.py

```
# coding: utf-8

class Person:
    '''A sample class for data attribute.'''
    def __init__(self, name):
        self.name = name

    def greet(self):
        print(f"My name is {self.name}.")

taro = Person("Taro")
jiro = Person("Jiro")

taro.greet()
jiro.greet()
```

1.2 クラス変数

Python にもクラス変数があります．データ属性 (インスタンス変数) がインスタンス固有のデータであるのに対して，クラス全体で共有するデータがクラス変数です．簡単な例を見てみましょう．下記の例の場合，Person クラスのインスタンスが生成されるたびにカウンターをインクリメントしていきます．counter の値を見てみると生成したインスタンスの数になっていることがわかります．

w5e3.py

```
# coding: utf-8

class Person:
    counter = 0 # クラス変数

    def __init__(self, name):
        Person.counter += 1 # クラス名.変数名
        self.name = name

taro = Person("Taro")
jiro = Person("Jiro")

print(Person.counter)

saburo = Person("Saburo")

print(Person.counter)
```

1.3 プライベートアクセス

Java などの場合は private,public や protected などのアクセス修飾子がありましたが Python にはありません．Java などではメンバ変数を外部から隠蔽したい場合は「private」をつけることによって実現できました．しかし,Python はそのようなキーワードがないため基本的に外部からアクセスできてしまいます．ただし，慣例として「_名前」と書くことで非 public として扱われます．この場合でも外部からアクセスすることは可能なのでプログラマが気をつけなければなりません. 下記の例では Person クラスのデータ属性が「self._name」となっています．「j._name」とすればデータにアクセスできてしまうので注意してください．

w5e4.py

```
# coding: utf-8

class Person:
    def __init__(self, name):
        self._name = name

    def greet(self):
        print(f"Hello! My name is {self._name}")

j = Person("John")
j.greet()
print(j._name) # 呼び出すことはできる
```

1.4 getter/setter

前節で話したように Python ではメンバを隠蔽化しようとしても外部からアクセスすることが可能である．だからと言って値を直接呼び出して値の変更や取得はしたくないと思います．基本的には getter や setter を使いたいと思うでしょう．自分で getX() や setX() としても良いですが Python にはデコレータが用意されています. 実際に例を見て見ましょう.@property 以下に「self._x」についての getter の定義を記述しています.@x.setter 以下に「self._x」の setter の定義を記述しています．

w5e5.py

```
# coding: utf-8

class Point:
    def __init__(self, x, y):
        self._x = x
        self._y = y

    @property
    def x(self):
        return self._x

    @x.setter
    def x(self, value):
        self._x = value

    @property
    def y(self):
        return self._y

    @y.setter
    def y(self, value):
        self._y = value

p = Point(0.0, 1.0)
print(f"({p.x}, {p.y})")

p.x = 0.5
p.y = p.y + 1.5
print(f"({p.x}, {p.y})")
```

1.5 練習

それでは，ここまでの内容を踏まえて複素数クラスを作成してみましょう．最低限以下の条件を満たしてください．それ以外の実装は自由で構いません．

- 属性として実数部と虚数部を持つ
- 他の複素数との和を計算して，新しい複素数クラスのインスタンスを生成する add メソッドを持つ
- setter を使用しない

1.6 継承

もちろん，Python のクラスにおいても継承はあります．クラスの継承は次のように行います．

```
class DerivedClassName(BaseClassName):
 定義 ...
```

実際に例を見てみましょう．

w5e6.py

```
# coding: utf-8

class Enemy:
    ''' A base class.'''

    def __init__(self, hp, level, ap):
        self._hp = hp
        self._level = level
        self._ap = ap

    def show_status(self):
        print(f"HP:{self._hp}, Levle:{self._level}, AP:{self._ap}")

class Magician(Enemy):
    ''' This class is derived from Enemy class.'''

    def __init__(self, hp, level, ap, mp):
        super().__init__(hp, level, ap)
        self._mp = mp

    def show_status(self):
        print(f"HP:{self._hp}, Level:{self._level}, AP:{self._ap}, MP:{self._mp}")

e = Enemy(50, 1, 10)
m = Magician(50, 1, 20, 20)
e.show_status()
m.show_status()
```

Enemy クラスを継承して Magician クラスを作りました．Magician クラスの__init__で Enemy クラス (スー

パークラス) の__init__を呼び出すことができています．また，show_status メソッドもオーバーライドできていることがわかります．

2　モジュール

Python において機能ごと等に分割してスクリプトを作成したい時があります．その場合にそれらのファイル (〜.py) を「モジュール」と呼びます．モジュールにある関数やクラスはモジュールを import することで使用することができます．Python には標準モジュールが用意されています．例として数学関数を使用できる math モジュールを import してみましょう．

w5e7.py

```
# coding: utf-8
import math

print(math.gcd(48, 32))
print(math.log10(100))
print(math.radians(90))
```

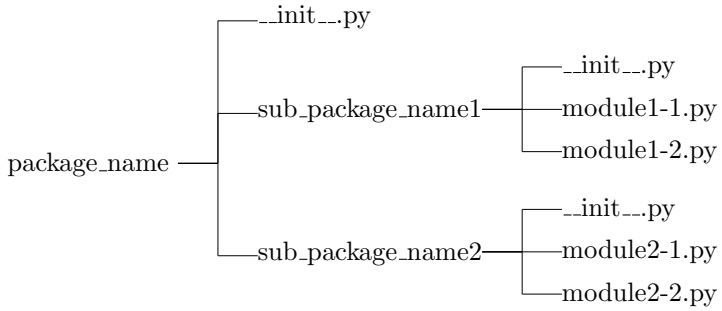
同様に自作モジュールも import することで使用することができます．先ほど作成した w5e5.py の Point クラスを他のモジュールから使用してみましょう．

w5e8.py

```
# coding: utf-8
import math
import w5e5

class Line:

    def __init__(self, start, end):
        self._start = start
        self._end = end
        self._length = math.sqrt(pow(self._start.x, 2) + pow(self._end.x, 2))

    def show(self):
        print(f"({self._start.x}, {self._start.y})-({self._end.x}, {self._end.y}):\
        length({self._length})")

l = Line(w5e5.Point(0.0, 1.0), w5e5.Point(3.0, 3.0))
l.show()
```

3　パッケージ

自作モジュールをパッケージ化することによってモジュールの整理や名前の衝突を避けることができます．パッケージ化は以下のディレクトリ構造で実現することができます．パッケージはサブパッケージを持つことができます．フォルダをパッケージとして認識させるためには「__init__.py」を作成する必要があります．このファイルは空でも問題ありません．



それぞれのモジュールをパッケージから import することができます．from ～ import ～を使うことで関数やクラスを使用時にモジュール名を省略することができます．

```
import package_name.sub_package_name1.module1-1
```

```
from package_name.sub_package_name1 import module1-2
```