POO em Python

Rafael Vieira Coelho

Programação Orientada a Objetos

rafaelvc2@gmail.com

Classes

- A linguagem Python é bastante popular para criar scripts, mas ela também suporta programação orientada a objetos.
- Classes descrevem dados (atributos) e métodos que manipulam estes dados (comportamento) que são encapsulados em objetos independentes.
- Códigos que usam este tipo de abstração favorece a manutenção e entendimento.

Definindo a Classe Person

Ex1Classe.py

```
"""A simple class."""
class Person(object):
    species = "Homo Sapiens"

    """This is the initializer. It's a special method (see below)."""
    def __init__(self, name):
        self.name = name

    """This method is run when Python tries to cast the object to a string.
     Return this string when using print(), etc. """
    def __str__(self):
        return self.name

    """Reassign and print the name attribute."""
    def rename(self, renamed):
        self.name = renamed
        print("Now my name is {}".format(self.name))

```

self

- Todos métodos de uma classe deve receber como parâmetro o próprio objeto (self).

Ex1Classe.py

```
1  """A simple class."""
2  class Person(object):
3      species = "Homo Sapiens"
4
5      """This is the initializer. It's a special method (see below)."""
6      def __init__(self, name):
7          self.name = name
8
9      """This method is run when Python tries to cast the object to a string.
10      Return this string when using print(), etc. """
11     def __str__(self):
12         return self.name
13
14     """Reassign and print the name attribute."""
15     def rename(self, renamed):
16         self.name = renamed
17         print("Now my name is {}".format(self.name))
18
```

__init__

- Este método é usado para inicializar os atributos (construtor).

Ex1Classe.py

```
1  """A simple class."""
2  class Person(object):
3      species = "Homo Sapiens"
4
5      """This is the initializer. It's a special method (see below)."""
6      def __init__(self, name):
7          self.name = name
8
9      """This method is run when Python tries to cast the object to a string.
10      Return this string when using print(), etc. """
11     def __str__(self):
12         return self.name
13
14     """Reassign and print the name attribute."""
15     def rename(self, renamed):
16         self.name = renamed
17         print("Now my name is {}".format(self.name))
18
```

__str__

- Este método é usado criar uma representação textual do objeto.

Ex1Classe.py

```
1  """A simple class."""
2  class Person(object):
3      species = "Homo Sapiens"
4
5      """This is the initializer. It's a special method (see below)."""
6      def __init__(self, name):
7          self.name = name
8
9      """This method is run when Python tries to cast the object to a string.
10     Return this string when using print(), etc. """
11     def __str__(self):
12         return self.name
13
14     """Reassign and print the name attribute."""
15     def rename(self, renamed):
16         self.name = renamed
17         print("Now my name is {}".format(self.name))
18
```

Instanciando Objetos da Classe Person

```
20  if __name__ == '__main__':
21      # Instances
22      kelly = Person("Kelly")
23      joseph = Person("Joseph")
24      john_doe = Person("John Doe")
25
26      # Attributes
27      print(kelly.species)
28      print(john_doe.species)
29      print(joseph.name)
30
31      # Methods
32      print(john_doe.__str__())
33      print(john_doe)
34      john_doe.rename("John")
35
```

```
Homo Sapiens
Homo Sapiens
Joseph
John Doe
John Doe
Now my name is John
```

Section 38.2: Bound, unbound, and static methods

The idea of bound and unbound methods was removed in Python 3. In Python 3 when you declare a method within a class, you are using a `def` keyword, thus creating a function object. This is a regular function, and the surrounding class works as its namespace. In the following example we declare method `f` within class `A`, and it becomes a function `A.f`:

Python 3.x `Version ≥ 3.0`

```
class A(object):
    def f(self, x):
        return 2 * x
A.f
# <function A.f at ...>  (in Python 3.x)
```

Métodos Estáticos

- Apenas precisamos mudar a anotação.

Ex2Metodos.py

```
1  class D(object):
2      multiplier = 2
3
4      @classmethod
5      def f(cls, x):
6          return cls.multiplier * x
7
8      @staticmethod
9      def g(name):
10         print("Hello, %s" % name)
```

```
12 if __name__ == '__main__':
13     print(D.f) # <bound method type.f of <class '__main__.D'>> D.f(12)
14     D.g # <function D.g at ...>
15     D.g("world") # Hello, world
16
17     d = D()
18     d.multiplier = 1337
19     print((D.multiplier, d.multiplier))
20     # (2, 1337)
21     print(d.f) # <bound method D.f of <class '__main__.D'>> d.f(10)
```

```
<bound method D.f of <class '__main__.D'>>
Hello, world
(2, 1337)
<bound method D.f of <class '__main__.D'>>
```

Herança entre Classes

- A superclasse se chama BaseClass neste exemplo e a classe filha que herda as características da classe mãe é DerivedClass.

Ex3Heranca.py

```
1  class BaseClass(object):
2      pass
3
4
5  class DerivedClass(BaseClass):
6      pass
```

Herança Simples

- A classe Square é filha de Rectangle.

```
 9  class Rectangle:
10      def __init__(self, w, h):
11          self.w = w
12          self.h = h
13
14      def area(self):
15          return self.w * self.h
16
17      def perimeter(self):
18          return 2 * (self.w + self.h)
19
20
21  class Square(Rectangle):
22      def __init__(self, s):
23          # call parent constructor, w and h are both s
24          super(Square, self).__init__(s, s)
25          self.s = s
```

```
28  if __name__ == '__main__':
29      r = Rectangle(10, 20)
30      print(r.area())  # Output: 12 r.perimeter() # Output: 14
31
32      s = Square(5)
33      print(s.area())  # Output: 4 s.perimeter() # Output: 8
```

```
200
25
```

Herança Múltipla

- A classe FooBar é classe filha das classes Foo e Bar.

Ex4HerancaMultipla.py

```
1  class Foo(object):
2      foo = 'attr foo of Foo'
3
4
5  class Bar(object):
6      foo = 'attr foo of Bar'  # we won't see this. bar = 'attr bar of Bar'
7
8
9  class FooBar(Foo, Bar):
10     foobar = 'attr foobar of FooBar'
11
12
13 if __name__ == '__main__':
14     # Now if we instantiate FooBar, if we look up the foo attribute,
15     # we see that Foo's attribute is found first
16     fb = FooBar()
17     print(fb.foo)
18     print(FooBar.mro())
```

```
attr foo of Foo
[<class '__main__.FooBar'>, <class '__main__.Foo'>, <class '__main__.Bar'>, <class 'object'>]
```

super

- É usado para acessar a superclasse.

```
class Foo(object):
    def foo_method(self):
        print "foo Method"

class Bar(object):
    def bar_method(self):
        print "bar Method"

class FooBar(Foo, Bar):
    def foo_method(self):
        super(FooBar, self).foo_method()
```

super

```
class Foo(object):
    def __init__(self):
        print "foo init"

class Bar(object):
    def __init__(self):
        print "bar init"

class FooBar(Foo, Bar):
    def __init__(self):
        print "foobar init"
        super(FooBar, self).__init__()

a = FooBar()
```

Output:

```
foobar init
foo init
```

Composição

- Podemos ter objetos como atributos de classes.

Ex5Composicao.py

```python
class Person(object):
    def __init__(self, ID, name):
        self.ID = ID
        self.name = name


class City(object):
    def __init__(self, numPeople):
        self.people = []
        self.numPeople = numPeople

    def add_person(self, person):
        self.people.append(person)
        self.numPeople += 1


class Country(object):
    def __init__(self):
        self.cities = []

    def add_city(self, city):
        self.cities.append(city)

    def people_in_my_country(self):
        x = 0
        for c in self.cities:
            x += c.numPeople
        return x
```

```python
if __name__ == '__main__':
    p1 = Person(1, "Joe")
    p2 = Person(2, "Mary")
    NYC = City(998)
    NYC.add_person(p1)
    NYC.add_person(p2)
    US = Country()
    US.add_city(NYC)
    print(US.people_in_my_country())
```

```
1000
```

Exercícios

- Que tal criar uma classe para representar jogos de vídeo games?
- Quais atributos devemos ter?
- Quais métodos?
- Criem dois objetos da classe criada com os seus vídeo games prediletos em um programa principal.