Computadores e Programação

2009–2010 1º semestre — aula 8

Helmut Wolters

helmut@coimbra.lip.pt

2009-11-20

Classes

- Todos os tipos de dados são objectos em Python.
- É o facto de um objecto "saber" o que é e que funções se lhe podem aplicar que confere robustez ao Python e o torna tão simples.
- É possível adicionar classes de objectos à linguagem através da instrução `class`.
- Por exemplo, para criar uma classe de vectores:

```
>>> class Vec:
...   def __init__(self,x,y,z):
...     self.x=x
...     self.y=y
...     self.z=z
...   def norm(self):
...     return (self.x**2+self.y**2+self.z**2)**0.5
```

Classes

- A instrução `class` apenas define a classe, não cria nenhum objecto. Para criar um objecto do tipo `Vec` (em terminologia de programação orientada por objectos, diz-se criar uma instância da classe) invoca-se a classe como se de uma função se tratasse:

```
>>> u=Vec(1,1,1)
```

- Neste exemplo, `u` será uma instância da classe `Vec`:

```
>>> type(u)
<type 'instance'>
>>> type(Vec)
<type 'classobj'>
```

Classes

- A função definida no interior da classe (designa-se por método da classe) é uma função especial (usa o nome especial `__init__`) que dá valores iniciais aos atributos do objecto (`x`, `y` e `z`, neste caso). O seu primeiro argumento (`self`) representa a própria instância da classe e é sempre omitido quando se invoca um método.
- Uma classe é um espaço de nomes, tal como um módulo.
- Os métodos são invocados qualificando o objecto (`objecto.metodo()`)

```
>>> u.x
1
>>> u.norm()
1.7320508075688772
>>> u.__init__(1,1,1,1)
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: __init__() takes exactly 4 arguments (5 given)
```

Classes

- Ao contrário dos módulos, é possível adicionar nomes a uma classe. Para tal basta fazer uma atribuição:

```
>>> dir(u)
['__doc__', '__init__', '__module__', 'x', 'y', 'z']
>>> u.novo_atributo=100
>>> dir(u)
['__doc__', '__init__', '__module__', 'novo_atributo', 'x'
>>> u.novo_atributo
100
```

- Embora as classes e os módulos sejam ambos espaços de nomes, são estruturas bastante diferentes: as classes são fábricas de objectos que herdam propriedades umas das outras e permitem fazer a sobrecarga de operadores.

Classes

- Podem-se definir subclasses de uma classe existente. Neste caso a subclasse herda todos os atributos e métodos da classe mãe:

```
>>> class UnitVec(Vec):
...   def __init__(self,x,y,z):
...     norm=(x**2+y**2+z**2)**0.5
...     self.x=x/norm
...     self.y=y/norm
...     self.z=z/norm
>>> v=UnitVec(3,1,2)
>>> dir(v)
['__doc__', '__init__', '__module__', 'norm', 'x', 'y', 'z
>>> v.x
0.80178372573727319
>>> v.norm()
1.0
```

Classes

- Há um conjunto de nomes especiais para os métodos que permite fazer a sobrecarga de operadores. Por exemplo, se se definir um método usando o nome especial `__or__` esse método passa a poder ser invocado usando o operador `|`:

Classes

```
>>> class Vec:
...   def prod(self,other):
...     return Vec(self.y*other.z-self.z*other.y,\
...                self.z*other.x-self.x*other.z,\
...                self.x*other.y-self.y*other.x)
...   def __or__(self,other):
...     return self.prod(other)
...
>>> u=Vec(1,0,0)
>>> v=Vec(0,1,0)
>>> a=u.prod(v)
>>> print a.x, a.y, a.z
0 0 1
>>> b=u|v
>>> print b.x, b.y, b.z
0 0 1
```

Classes

- Alguns nomes especiais para sobrecarga de operadores: `__init__`, `__del__`, `__add__`, `__or__`, `__repr__`, `__len__`, `__cmp__`, etc.

  http://www.python.org/doc/2.4.4/ref/customization.html

  http://www.python.org/doc/2.4.4/ref/numeric-types.html

- Resumindo: as classes permitem incorporar numa estrutura única os atributos de um objecto e as funções que é possível efectuar sobre esse objecto. A herança é um mecanismo que permite definir subclasses especializadas que herdam todos os atributos e métodos da classe mãe. As subclasses podem conter novos métodos ou versões diferentes dos métodos da classe mãe.
- As interfaces gráficas são construídas com classes...