7. Objetos

May 5, 2014

1 Programación orientada a objetos

Todo en Python es un objeto.

1.1 ¿Qué es un objeto?

Por ejemplo, supongamos que queremos simular una partícula:

```
In [1]: x = 3
        v = 1
        m = 10
        q = -1
```

Hemos representado las propiedades de una partícula en nuestro programa.

```
In [2]: x2 = 7
        v2 = 1
        m2 = 1
        q2 = -1
```

Para 10 partículas: usar un arreglo

```
In [3]: x = [3, 7, 1]
```

¿Dónde están las partículas? Están en todos lados / ningún lado. No es una representación explícita / fiel de mi modelo.

Quisiera poder decir "esta cosa representa la partícula número 1"

Para esto necesitamos *objetos*: vamos a *crear* nuevos tipo de objeto que contengan propiedades internas propias, y que pueden hacer cosas.

Un *objeto* es una cosa que contiene cosas y que hace cosas.

Un *objeto* es una caja que contiene atributos (variables) y que tiene funciones (métodos / funciones computacionales).

¿Cómo se puede hacer esto en Python

1.2 Clases

Una *clase* (*class* en inglés) es una especificación de la estructura de un conjunto de objetos con un tipo nuevo dado

E.g. queremos un tipo de objeto de nuevo para representar una partícula

```
In [7]: class Particula:
            pass  # pass hace nada
```

Por lo pronto, tenemos una caja vacía, que no contiene nada

Vamos a *crear* un objeto de este tipo:

```
In [9]: p = Particula()
        # Usamos el nombre de la clase como si fuera una función

In [11]: p  # caja vacia por el momento

Out[11]: <__main__.Particula instance at 0x10c8a4a70>
```

Preguntarle al objeto `p` qué contiene:
`p.<TAB>`

```
In [12]: p.x = 1

In [13]: p.v = 2
```

Crear otra partícula:

```
In [14]: p2 = Particula()

In [15]: p2.x = 10

In [16]: p.x

Out[16]: 1
```

1.3 Métodos

Los métodos son *funciones* propias de los objetos.

Son funciones que se declaran *adentro* de la declaración de la clase – por lo tanto, están *adentro* de la clase.

```
In [19]: class Particula:
             x = 1
             v = 2

In [20]: p = Particula()

In [22]: p.x, p.v

Out[22]: (1, 2)

In [23]: p2 = Particula()

In [24]: p2.x, p2.v

Out[24]: (1, 2)

In [25]: p2.x = 10

In [29]: p.x

Out[29]: 1

In [38]: class Particula:
             x = 1
             v = 2

             def mover(self, dt):
                 """mover por un tiempo dt"""

                 self.x += dt*self.v
```

`self` quiere decir `auto-`, `sí mismo`

`self` es una *referencia* (como un puntero) al objeto a quien movemos. Es algo que Python agrega automáticamente al momento de llamar la función.

```
In [47]: p = Particula()

In [48]: p.x, p.v

Out[48]: (1, 2)

In [49]: p.mover(0.1)  # le estoy diciendo a la particula 'p' "muevete"

In [50]: p.x, p.v

Out[50]: (1.2, 2)
```

1.4 Cómo inicializar objetos

Está horrible meter a mano valores iniciales: debería poder darle argumentos *en el momento de crear una instancia* (un objeto de un tipo dado) como sus valores iniciales: una función constructora / un *constructor*.

En Python, esto es una función especial con un nombre especial, igual para cualquier clase: `__init__`

```
In [53]: class Particula:

             def __init__(self, x0, v0):
                 self.x = x0
                 self.v = v0

             def mover(self, dt):
                 """mover por un tiempo dt"""

                 self.x += dt*self.v

In [55]: p = Particula(10, -1)

In [56]: p.x, p.v

Out[56]: (10, -1)
```

`__init__` es una función especial que se llama automáticamente al momento de crear un objeto, con el único fin de inicializar los atributos internos del objeto.

```
In [57]: p

Out[57]: <__main__.Particula instance at 0x10d773050>

In [58]: class Particula:

             def __init__(self, x0, v0):
                 self.x = x0
                 self.v = v0

             def mover(self, dt):
                 """mover por un tiempo dt"""

                 self.x += dt*self.v

             def __repr__(self):
                 # representacion visual del objeto
                 return "Particula(%s, %s)" % (self.x, self.v)
```

```
In [59]: p = Particula(10, -3)

In [60]: p

Out[60]: Particula(10, -3)
```

1.5 ¿Cómo hacer un gas de partículas?

Un gas consiste en N partículas.

```
In [71]: class Gas:
             def __init__(self, N):
                 self.N = N

                 # lista de particulas:
                 self.particulas = []

                 self.particulas.append(Particula(1, 1))

             def mover(self):
                 print("Estoy a punto de mover tu gas")


In [72]: g = Gas(3)

In [73]: g

Out[73]: <__main__.Gas instance at 0x10c895ea8>

In [74]: g.N

Out[74]: 3

In [75]: g.particulas

Out[75]: [Particula(1, 1)]

In [76]: g.mover()

Estoy a punto de mover tu gas

In [77]: g2 = Gas(10)

In [78]: g

Out[78]: <__main__.Gas instance at 0x10c895ea8>

In [79]: g2

Out[79]: <__main__.Gas instance at 0x10d77a908>
```

Las clases nos permiten *organizar* al código

```
In []:
```