Programação I
Aula 18 — Programação com objetos

Pedro Vasconcelos
DCC/FCUP

Nesta aula

Programação com objetos

Programação procedimental e funcional:

- decompomos problemas em sub-problemas
- compomos programas definimos funções e procedimentos

Programação com objetos:

- agregamos dados e operações em objetos
- compomos programas pela interação entre vários objectos

O que é um objeto?

- Entidade que agrega dados e operações associadas
- Permite agrupar valores e funcionalidades relacionados
- Propriedades dos objetos:

  identidade podemos distinguir um objeto de qualquer outro

  estado dados internos ao objeto

  comportamento operações que o objeto expõe ao exterior (métodos)

Módulo *turtle*

- Até agora usamos o *turtle* de forma procedimental
- Este módulo também pode ser usado com objetos

Cada tartaruga é um objeto com:

estado a posição e orientação, côr do traço, etc.;

comportamento métodos para mover e desenhar (`forward`, `left`, `right`, ...)

identidade podemos criar múltiplas tartarugas distintas numa mesma janela.

Módulo *turtle* (cont.)

```
>>> import turtle
>>> alice = turtle.Turtle()   # criar a 1ª tartaruga
>>> bob = turtle.Turtle()     # criar a 2ª tararuga
>>> alice.forward(200)        # mover a Alice
>>> bob.left(90)              # rodar Bob...
>>> bob.pencolor('blue')      # ...e mudar a côr
>>> bob.forward(100)          # mover o Bob
```

Estado

Cada tartaruga guarda estado:

- a sua posição e orientação;
- a cor da caneta e espessura do traço;
- ...

```
>>> alice.position()
(200.00,0.00)
>>> bob.position()
(0.00,100.00)
>>> alice.heading()
0.0
>>> bob.heading()
90.0
```


Identidade

Podemos distinguir as duas tartarugas.

```
>>> alice
<turtle.Turtle object at 0xfba3d0>
>>> bob
<turtle.Turtle object at 0x7f2dac171390>
>>> alice == bob
False
```

As duas tartarugas são diferentes mesmo se estiverem na mesma posição e com a mesma orientação!

Construtores

- `Turtle` é um construtor de objetos
  - não confundir com o nome do módulo `turtle`
- Usamos como uma função sem argumentos:

```
>>> import turtle
>>> alice = turtle.Turtle()
```

ou ainda

```
>>> from turtle import *
>>> alice = Turtle()
```

- `Turtle` é também o nome de uma classe:
  - contém as definições das operações comuns a todas as tartarugas
  - um *molde* para construir novos objetos (instâncias)

Tipos de dados

Os tipos pré-definidos em Python são objetos:

tipos básicos `int`, `float`, `bool`

tipos estruturados `str`, `list`, `tuple`, `dict`

Podemos definir novas classes para representar tipos de dados novos.

Classes

Objetos dum mesmo tipo pertem à mesma classe:

- todos os objetos duma classe têm os mesmos métodos
- cada objeto tem o seu estado e identidade próprios

Exemplo:

- os objetos tartaruga pertencem à classe `Turtle`
- todos suportam os mesmo comandos
- a posição, orientação, etc. são específicos de cada objeto

Definição de classes

```
class nome:
    ⋮
    definições
    ⋮
```

Pontos no plano

Vamos definir uma classe para representar pontos geométricos no plano 2-D.

Cada ponto terá dois atributos: as coordenadas x, y.

Pontos no plano (cont.)

Primeira versão: declaramos uma nova classe sem qualquer método.

```
class Point:
    '''Classe para representar pontos no plano;
    cada ponto terá dois atributos: x, y.'''
```

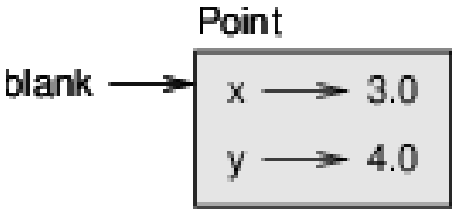
Pontos no plano (cont.)

Criar um ponto:

```
>>> blank = Point()
>>> blank
<__main__.Point object at 0xb7e9d3ac>
```

Definir os atributos:

```
>>> blank.x = 3.0
>>> blank.y = 4.0
```



Atributos

Podemos usar os atributos tal como outras variáveis.

```
>>> '(%g, %g)' % (blank.x, blank.y)
'(3.0, 4.0)'
>>> distance = math.sqrt(blank.x**2 + blank.y**2)
>>> distance
5.0
```

Passagem de objetos

Também podemos definir funções funções cujos argumentos são objetos.

```
def print_point(p):
    "Imprimir as coordenadas dum ponto."
    print('(%g, %g)' % (p.x, p.y))
```

Passamos um objeto a uma função pelo nome:

```
>>> print_point(blank)
(3.0, 4.0)
```


Exercício

Definir uma função `distance(p1,p2)` que calcule a distância entre dois pontos.

Rectângulos

Vamos agora definir uma classe para representar *rectângulos*.

Atributos:

width	largura
height	altura
corner	canto inferior esquerdo (um objeto `Point`)

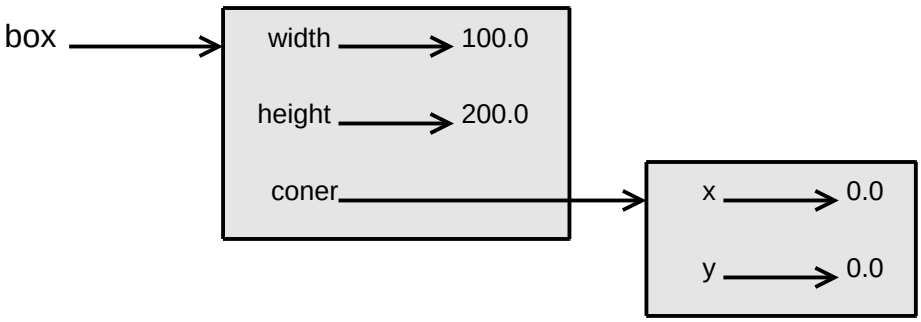
(Para simplificar: apenas rectângulos com lados paralelos aos eixos.)

Rectângulos (cont.)

```
class Rectangle:
    '''Classe para representar rectângulos;
    atributos: width, height, corner.'''
```

Criar um rectângulo

```
>>> box = Rectangle()
>>> box.width = 100.0
>>> box.height = 200.0
>>> box.corner = Point()
>>> box.corner.x = 0.0
>>> box.corner.y = 0.0
```



Métodos

Podemos associar *operações* aos objetos acresentando métodos às classes.

Exemplo: calcular a área

```
class Rectangle:
    ⋮
    def area(self):
        return self.width * self.height
```

O primeiro argumento de um método (chamado *self*) é sempre o objeto sobre o qual invocamos o método:

```
>>> box.area()
20000.0
```


Inicialização

- Vamos definir métodos `__init__` para efetuar a inicialização dos objetos
- Os argumentos do método `__init__` são passados quando construimos um novo objeto
- Desta forma podemos garantir que os objetos têm sempre os atributos necessários

Inicialização (cont.)

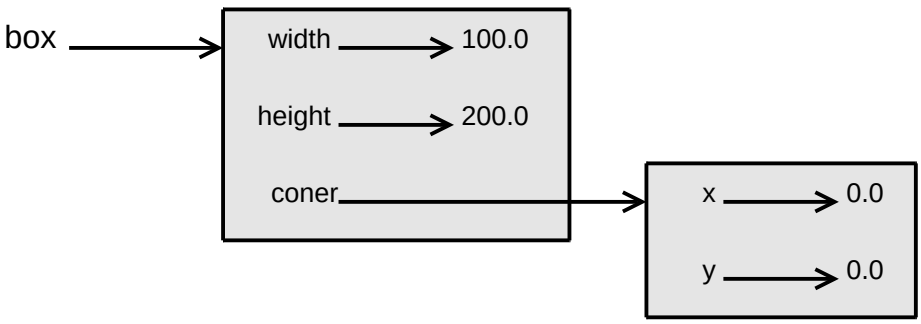
```
class Point:
    ⋮
    def __init__(self, x, y):
        "Ponto com coordenadas x, y."
        self.x = x
        self.y = y

class Rectangle:
    ⋮
    def __init__(self, pt, w, h):
        "Rectângulo com canto pt e dimensões w, h."
        self.width = w
        self.height = h
        self.corner = pt
```


Inicialização (cont.)

Criar o rectângulo (exemplo anterior)

```
>>> box = Rectangle(Point(0.0,0.0), 100.0, 200.0)
```



Conversão em texto

Por omissão, um objeto é mostrado com o nome da classe e um endereço de memória:

```
>>> box = Rectangle(Point(0.0,0.0), 100.0, 200.0)
>>> print(box)
<__main__.Rectangle object at 0x7fd15f7ffb10>
```

Vamos definir métodos `__str__` para mostrar de forma mais inteligível.

Conversão em texto (cont.)

```
class Point:
    ⋮
    def __str__(self):
        return "Point(%g,%g)" % (self.x, self.y)

class Rectangle:
    ⋮
    def __str__(self):
        return ("Rectangle(%s, %g, %g)" %
                (str(self.corner),
                 self.width, self.height))
```

Conversão em texto (cont.)

```
>>> box = Rectangle(Point(0.0,0.0), 100.0, 200.0)
>>> str(box)
'Rectangle(Point(0.0, 0.0), 100.0, 200.0)'
>>> print(box)
Rectangle(Point(0.0, 0.0), 100.0, 200.0)
```