Programação com Interfaces Gráfica

Mario Benevides

Universidade Federal do Rio de Janeiro
Rio de Janeiro, Brasil

Classes

Agenda

Aula Passada: Introdução a OO e Classes

Nesta Aula: Classes (Continuação)

- Atributos herdados da classe
- Método init
- Especialização de classes
- Herança Múltipla
- Métodos Mágicos
- Getters, Setters e Propriedades

Atributos herdados da classe

- Se uma classe define atributos de classe, as instâncias herdam esses atributos da classe como atributos de instância
- Exemplo:

```
>>> class C(object):
 a = 1
 def f(self,x):
          self.a += x

>>> c = C()
>>> c.f(2)
>>> c.a
3
>>> C.a
1
```

Método init

- Um método como init do exemplo da classe $Time$ é bastante útil para inicializar atributos da instância e é conhecido como construtor da classe
- Na verdade, Python suporta construtores que podem ser chamados automaticamente na criação de instâncias
  - Basta definir na classe um método chamado __init__
  - Este método é chamado automaticamente durante a criação de um nova instância da classe, sendo que os argumentos são passados entre parênteses após o nome da classe
- O método __init__ é apenas um exemplo de "método mágico" que é invocado de maneira não padrão

Exemplo

```
>>> class C(object):
        def __init__(self,a=2,b=3):
            self.a = a
            self.b = b
        def f(self,x):
            return self.a*x+self.b


>>> obj1 = C()
>>> obj2 = C(8,1)
>>> obj1.f(7)
17
>>> obj2.f(7)
57
```

Especialização de classes

- Fazer uma classe C herdar de outra B, basta declarar C como:

  ```
  class C(B):
  ```

- Diz-se que C é sub-classe (ou derivada) de B ou que B é super-classe (ou base) de C
- C herda todos os atributos de B
- A especialização de C se dá acrescentando-se novos atributos (variáveis e métodos) ou alterando-se métodos
- Se, um método de C, precisa invocar um método m de B, pode-se utilizar a notação B.m para diferenciar do m de C, referido como C.m

Exemplo: Especialização de classes

```
>>> class B(object):
n = 2
def f(self,x): return B.n*x
>>> class C(B):
def f(self,x): return B.f(self,x)**2
def g(self,x): return self.f(x)+1
>>> b = B()
>>> c = C()
>>> b.f(3)
6
>>> c.f(3)
36
>>> c.g(3)
37
>>> B.n = 5
>>> c.f(3)
225
```

Observação: Especialização de classes

- O parâmetro self não pode ser removido da chamada da função f de B, na classe C, do exemplo anterior:

```
>>> class C(B):
def f(self,x): return B.f(x)**2
def g(self,x): return self.f(x)+1

>>> c=C()
>>> print (c.f(3))
```

- ERRO!!!

Observação: Especialização de classes

- Para chamar um classe D derivada de uma classe C é preciso chamar C de maneira a inicializá-la.
- Isto não é feito automaticamente (por default).
- Permite inicializar os elementos de C que não são específicos de D
- Usa-se a notação C.__init__(self, ...)
- Exemplo:

```
>>> class C(object):
      def __init__(self):
        print ("Construtor de C")
        self.x = 1

>>> class D(C):
     def __init__(self):
         print ("Construtor de D")
         C.__init__(self)
        self.y = 2
```

Exemplo: Especialização de classes

```
>>> class C(object):
      def __init__(self):
        print ("Construtor de C")
        self.x = 1

>>> class D(C):
     def __init__(self):
         print ("Construtor de D")
         C.__init__(self)
         self.y = 2

 >>> d=D()
Construtor de D
Construtor de C
>>> d.x
1
>>> d.y
2
```

Observação:

- A partir do Python 2.2, classes podem também ser declaradas no chamado "novo estilo":
- Se uma classe não é derivada de nenhuma outra, ela deve ser declarada como derivada da classe especial chamada object.
- Exemplo:

```
class C(object):
```

Herança Múltipla

- É possível construir uma classe que herda de duas ou mais outras. Ex.: class C(A,B): ...
- Nesse caso, a classe derivada herda todos os atributos de ambas as classes-base
- Se ambas as classes base possuem um atributo com mesmo nome, aquela citada primeiro prevalece
- No exemplo acima, se A e B possuem um atributo x, então C.x se refere ao que foi herdado de A

Exemplo: Herança Múltipla

```
class C(object):
        def __init__(self,a,b):
                self.a, self.b = a, b
        def f(self,x):
            return self.a*x+self.b
class D(object):
        def __init__(self,legenda):
                self.legenda = legenda
        def escreve(self,valor):
                print (self.legenda,'=',valor)
class E(C,D):
        def __init__(self,legenda,a,b):
                C.__init__(self,a,b)
                D.__init__(self,legenda)
        def escreve(self,x):
                D.escreve(self,self.f(x))
```

Exemplo: Herança Múltipla

```
class C(object):
        def __init__(self,a,b):
                self.a = a
                self.b = b
        def f(self,x):
            return self.a*x+self.b
class D(object):
        def __init__(self,legenda):
                self.legenda = legenda
        def escreve(self,valor):
                print (self.legenda,'=',valor)
class E(C,D):
        def __init__(self,legenda,a,b):
                C.__init__(self,a,b)
                D.__init__(self,legenda)
        def escreve(self,x):
                D.escreve(self,self.f(x))
e = E("f",10,3)
print (e.escreve(4))
>>>
f = 43
```

Atributos Privados

- Em princípio, todos os atributos de um objeto podem ser acessados tanto dentro de métodos da classe como de fora.
- Quando um determinado atributo deve ser acessado apenas para implementação da classe, ele não deveria ser acessível de fora.
- Em princípio tais atributos não fazem parte da interface "pública" da classe.
- Atributos assim são ditos privados.
- Em Python, atributos privados têm nomes iniciados por dois caracteres "traço-embaixo", isto é, __.

Exemplo: Atributos Privados

```
class C(object):
        def __init__(self,x): self.__x = x
        def incr(self):
             self.__x += 1
             return self.__x
a = C(5)
print (a.incr())

>>>
6

>>> a.__x
```

ERRO!!!

Métodos Mágicos

- São métodos que são invocados usando operadores sobre o objeto ao invés de por nome.
- Exemplo: o construtor __init__
- Outros:
- Adição: __add__
  - Chamado usando '+'
- Adição: __sub__
  - Chamado usando '-'
- Adição: __repr__
  - Chamado quando objeto é impresso

Exemplo: Métodos Mágicos

```
class vetor:
def __init__(self,x,y):
   self.x, self.y = x,y
def __add__(self,v):
   return vetor(self.x+v.x, self.y+v.y)
def __sub__(self,v):
   return vetor(self.x-v.x, self.y-v.y)
def __repr__(self):
  return "vetor("+str(self.x)+","+str(self.y)")"
>>> a=vetor(1,2)
>>> a += vetor(3,5)
>>> a-vetor(2,2)
vetor(2,5)
>>> print (a)
vetor(4,7)
```

Getters, Setters e Propriedades

- Muitas vezes queremos que determinados atributos possam ser acessados de forma controlada, isto é, vigiados por métodos
- Os métodos que controlam o acesso a tais atributos são conhecidos como getters e setters , referindo-se a métodos de leitura e escrita, respectivamente
- Os atributos controlados são chamados de propriedades
- Úteis para Encapsulamento e proteger atributos

Exemplo 1:Getters, Setters e Propriedades

```
class C:

    def __init__(self,x):
        self.setX(x)

    def getX(self):
        return self.__x

    def setX(self, x):
        if x < 0: print ("Valor menor que 0")
        elif x > 1000: print ("Valor maior que 1000")
        else:
            self.__x = x
```

Exemplo 1:Getters, Setters e Propriedades

```
class C:

    def __init__(self,x):
        self.setX(x)

    def getX(self):
        return self.__x

    def setX(self, x):
        if x < 0: print ("Valor menor que 0")
        elif x > 1000: print ("Valor maior que 1000")
        else:
            self.__x = x

>>> c1= C(100)
>>> c2= C(150)
>>> c1.setX(c1.getX()+c2.getX())
>>> c1.getX()
250
>>>c1.setX(-10)
Valor menor que 0
```

Exemplo 2: Getters, Setters e Propriedades

```
class Retangulo:
  def __init__(self,tamanho):
     self.setTamanho(tamanho)
  def setTamanho(self,tamanho):
     if min(tamanho)<0: print ("Erro!")
     self.__tamx,self.__tamy = tamanho
  def getTamanho(self):
     return (self.__tamx,self.__tamy)

>>> r = Retangulo((20,30))
>>> r.getTamanho()
(20, 30)
>>> r.setTamanho((-1,0))
Erro!
```

Getters, Setters e Propriedades

- A função property pode ser usada para consubstanciar uma propriedade implementada por métodos de tal maneira que ela pareça um atributo da classe
- Ela é usada no corpo de uma declaração de classe com a forma: atributo = property(fget, fset, fdel, doc)
- fget, fset, fdel são métodos para ler, escrever e remover o atributo
- doc é uma docstring para o atributo

Exemplo: Propriedades

```
class Retangulo:
  def __init__(self,tamanho):
     self.setTamanho(tamanho)
  def setTamanho(self,tamanho):
     if min(tamanho)<0:  print ("Erro!")
     self.__tamx,self.__tamy = tamanho
  def getTamanho(self):
     return (self.__tamx,self.__tamy)
  tamanho = property(getTamanho,setTamanho)

>>> r = Retangulo((20,30))
>>> r.tamanho
(20, 30)
>>> r.tamanho = (30,30)
>>> r.tamanho
(30, 30)
>>> r.tamanho = (-1,0)
Erro!
```

Dicas OO

- Agrupe funções e dados que se referem a um mesmo problema
- Por exemplo, se uma função manipula uma variável global, é melhor que ambas sejam definidas numa classe como atributo e método
- Não permita promiscuidade entre classes e instâncias de classe
  - Por exemplo, se há necessidade de um objeto manipular um atributo de outro, escreva um método com essa manipulação e chame-o
  - Não escreva métodos extensos
  - Em geral, um método deve ser o mais simples possível

Agenda

Nesta Aula: Classes

Próxima Aula: Exceções

- Objeto de Exceção
- Avisos
- Comando $raise$
- Classes de Exceção
- Mais de um $except$
- Comando $else$
- Comando $finally$
- Iteradores e Geradores