Programação com Interfaces Gráfica

Mario Benevides e Paulo Roma

Universidade Federal do Rio de Janeiro
Rio de Janeiro, Brasil

Introdução

Apresentação

- Mario Benevides e Paulo Roma
  - Professores Titulares
  - Dept. Ciência da Computação - IM
  - COPPE/Sistemas
- Aulas fortemente baseadas nas notas de aula do Professores Claudio Esperança (COPPE/Sistemas) e Paulo Roma;
- `http://orion.lcg.ufrj.br/python/`
- Agradecimento Especial ao Prof. Claudio Esperança pela suas excelentes notas de aula.

Agenda

Primeira Parte:

- Introdução à Orientação a Objeto
- Classes
- Exceções
- Modulos
- Arquivos

Segunda Parte:

- Interfaces Gráficas

Orientação a Objetos

- É uma disciplina de programação assim como a Programação Estruturada
- Tenta unificar as idéias de algoritmos e estruturas de dados através do conceito de Objeto
  - Um objeto é uma unidade de software que encapsula algoritmos e os dados sobre o qual os algoritmos atuam
  - Um objeto é um tipo abstrato de dados na prática
- Os seguintes conceitos são importantes quando falamos de orientação a objetos:
  - Polimorfismo
  - Abstração
  - Herança

Polimorfismo

- É o que permite que dois objetos diferentes possam ser usados de forma semelhante
- Por exemplo, tanto listas quanto tuplas ou strings podem ser indexadas por um número entre colchetes e suportam o método len
- Assim, se escrevemos

```
for i in range(len(X)): print (i, X[i])
```

- Não é possível saber de antemão se X é uma tupla, uma lista ou uma string
- Desta forma, se escrevemos um algoritmo para ser aplicado um objeto X, então também pode ser aplicado a um objeto Y desde que Y seja suficientemente polimórfico a X

Abstração e Encapsulamento

- É o que permite que um objeto seja utilizado sabendo-se sobre ele apenas a sua interface
- Um objeto é uma unidade de software que encapsula algoritmos e os dados sobre o qual os algoritmos atuam
- Em OO a abstração tem mais alcance pois um objeto encapsula tanto dados como algoritmos
- Assim, podemos atribuir objetos ou passar objetos como argumentos, sem necessariamente saber como o objeto está implementado

Herança

- É o que permite construir objetos que são especializações de outro objeto
  - Isso permite o re-uso de software já que objetos especializados herdam dos objetos genéricos uma série de atributos comuns
- Por exemplo, considere um objeto que representa uma forma geométrica. Então, ele pode ter características tais como área, perímetro, centróide, etc.
- Um polígono é uma forma geométrica
  - Portanto, herda todas as características de formas geométricas
  - Deve suportar também características específicas como número de lados e comprimento de arestas

Objetos em Python

- Python suporta OO através de classes
- Uma classe pode ser entendida como uma fábrica de objetos, todos com as mesmas características
  - Diz-se que objeto fabricado por uma classe é uma instância da classe
- A rigor, uma classe é também um objeto
  - Encapsula dados e algoritmos
  - Entretanto, não é normalmente um objeto fabricado por uma classe, mas um objeto criado pela construção class
- Um objeto encapsula dados e algoritmos sob a forma de variáveis e métodos
  - É comum chamar esses elementos constituintes dos objetos de atributos

Motivação p/ Usar OO

- Exemplo: Circulo: centro (x,y) e um raio
- Implementar: lista: circulo (x , y , raio)
- Exemplo: circulo=("3", "5", "7")
  - circulo[2] = "-5", Este erro só seria percebido quando formos usar o circulo
  - circulo.sort(), Este erro é difícil de perceber
- Usando OO
  - tratar exceções
  - encapsular métodos e dados
  - re-utilzar código

Declarando Classes

- A maneira mais simples é:

```
class nome (object):
    var = valor
    ...
    var = valor
    def metodo (self, ... arg):
        ...
    def metodo (self, ... arg):
        ...
```

- As variáveis e os métodos são escritos precedidos pelo nome da classe e por um ponto (.)
  - Assim, uma variável v definida numa classe c é escrita c.v
- Os métodos sempre têm self como primeiro argumento
- Uma nova instância da classe é criada usando nome ()

Exemplo 1

- Uma classe C com um método f:

```
class C (object):
    a = 2
    b = 3
    def f(self, x):
        return C.a*x+C.b
exemplo = C()
print ( exemplo.f(7))
```

- Executando o programa exemplo:

```
>>>
17
```

Atributos e Instâncias

- No exemplo anterior, a e b eram atributos da classe C e portanto usáveis por qualquer instância de C
- Um atributo attr associado a uma instância obj tem nome obj.attr

```
class C(object):
    a = 2
    b = 3
    def f(self, x):
        return C.a*x+C.b
exemplo = C()
print ( exemplo.f(7))

>>> C.a = 9
>>> exemplo.f(7)
66
```

Atributos e Instâncias

- Freqüentemente, atributos associados a instâncias individuais
- Referir atributo at de objeto dentro de seus métodos: self.at

```
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()
>>> obj1.init(2,3)
>>> obj2 = C()
>>> obj2.init(8,1)
>>> obj1.f(7)
17
>>> obj2.f(7)
57
```

Exemplo 2: Classe C time de futebol

```
class Time(object):
    ## Classe dos times de futebol
    def __init__(self, nome, campeao, divisao1):
        self.nome = nome
        self.campeao = campeao
        self.divisao1 = divisao1

    def descricao(self):
        ff = "Eu sou %s. Somos %s" % (self.nome, self.campeao)
        print ( ff )

    def div(self):
        if  self.divisao1:
             print ( "Somos da Primeira Divisão." )
        else:
             print ( "Estamos na Segunda Divisão." )
```

Exemplo 2: Classe C time de futebol

- Instanciando a classe Time para o objeto Fluminense

```
>>>Fluminense = Time("tricolor", "tetra-campeões", True)
>>>Fluminense.descricao()

Eu sou tricolor. Somos tetra-campeões

>>>Fluminense.div()

Somos da Primeira Divisão.
```

Resumo

Nesta Aula : Introdução a OO e Classes

Aula Seguinte: Classes (Continuação)

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