Curso de Python em 5 Horas

Classes e Trabalhando com Arquivo

Gustavo Sverzut Barbieri

GPSL - UNICAMP

12 de maio de 2005

Resumo

Esta aula apresenta o uso de classes e também como trabalhar com arquivos e entrada e saída em geral.

O material de apoio a ser utilizado se encontra em:

- Python para já programadores: `http://www.gustavobarbieri.com.br/python/aulas_python/aula-01.pdf`
- Resumo: `http://www.gustavobarbieri.com.br/python/aulas_python/resumo.pdf`

Panorama

Panorama

Classes: Sintaxe

```
class NOME_DA_CLASSE( CLASSE_PAI_1, CLASSE_PAI_2 ):
    """Documentacao da classe.
    """

    ATRIBUTOS_DE_CLASSE

    def __init__( self, PARAMETROS ):
        CODIGO_DO_CONSTRUTOR

    def METODO( self, PARAMETROS ):
        CODIGO_DO_METODO
```

Panorama

Classes "new-style"

- Desde a unificação dos tipos na versão 2.2 do Python, todos as classes devem herdar de `object` e são chamadas de *"new-style-classes"*.
- Por compatibilidade, ainda pode-se criar classes à moda antiga (clássicas).
- Porém classes clássicas não se beneficiam dos recursos providos:
  - Sem `super()`
  - Herança múltipla é precária, sem `obj.__mro__`
  - Sem conceito de propriedades: `property()`

Panorama

Classes: Exemplo Básico

```
import datetime

class Pessoa( object ):
    def __init__( self, nome, nascimento ):
        self.nome = nome
        self.nascimento = nascimento
    def idade( self ):
        delta = datetime.date.today() - self.nascimento
        return delta.days / 365
    def __str__( self ):
        return '%s, %d anos' % ( self.nome, self.idade() )

gustavo = Pessoa( 'Gustavo Sverzut Barbieri',
                  datetime.date( 1982, 6, 19 ) )
print gustavo.idade()
print gustavo
```

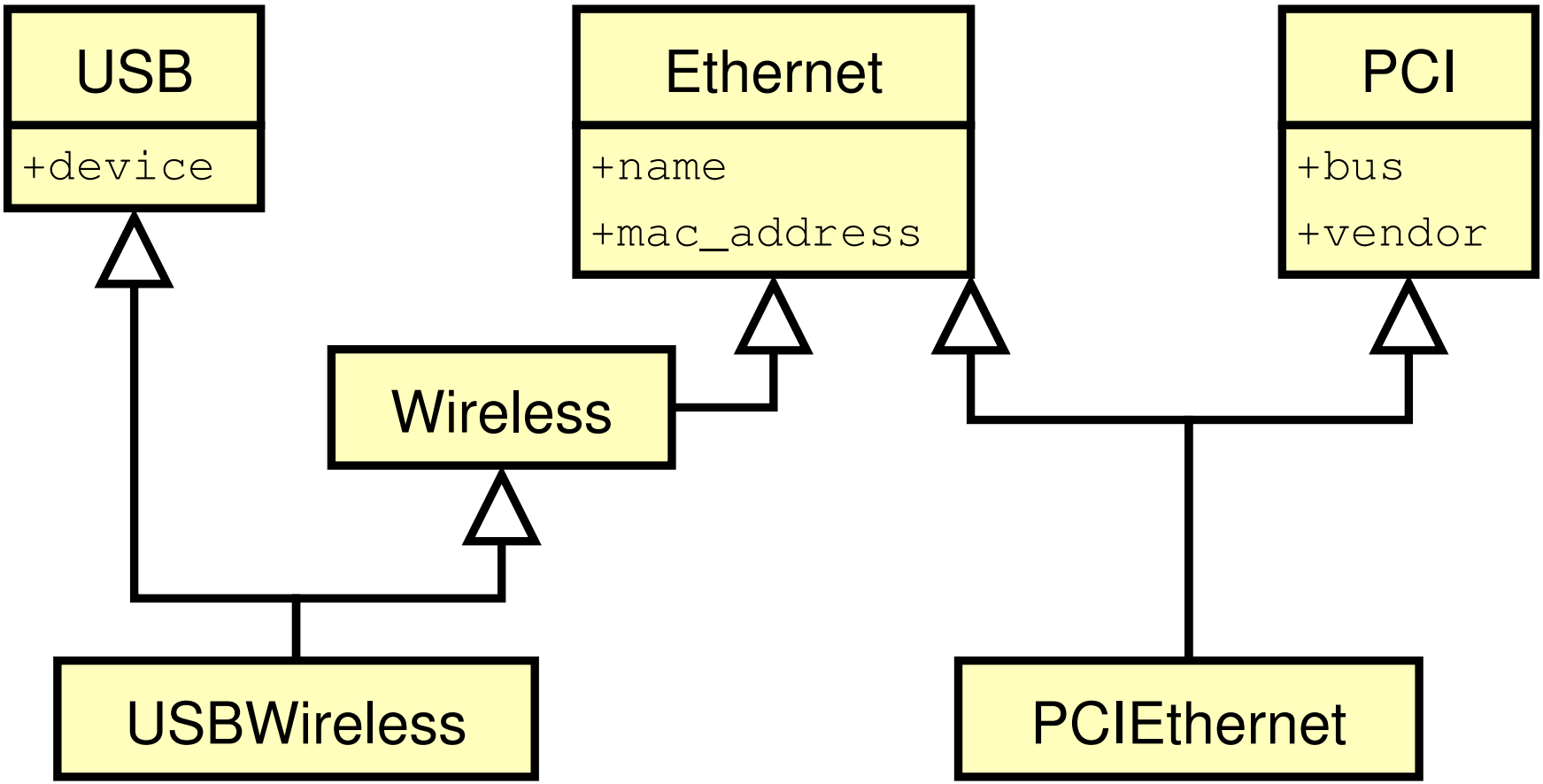
Classes: Exemplo Avançado

```
class Ethernet( object ):
    def __init__( self, name, mac_address ):
        self.name
        self.mac_address

class Wireless( Ethernet ):
    def __init__( self, name, mac_address ):
        Ethernet.__init__( self, name, mac_address )

class PCI( object ):
    def __init__( self, bus, vendor ):
        self.bus = bus
        self.vendor = vendor

class USB( object ):
    def __init__( self, device ):
        self.device = device
```

Classes: Exemplo Avançado (Continuação)

```
  class PCIEthernet( PCI, Ethernet ):
   def __init__( self, bus, vendor, name, mac_address ):
      PCI.__init__( self, bus, vendor )
      Ethernet.__init__( self, name, mac_address )

class USBWireless( USB, Wireless ):
   def __init__( self, device, name, mac_address ):
      USB.__init__( self, device )
      Wireless.__init__( self, name, mac_address )

wlan0 = USBWireless( 'usb0', 'wlan0', '00:33:44:55:66' )
eth0  = PCIEthernet( 'pci:0:0:1', 'realtek',
                     'eth0', '00:11:22:33:44' )

print isinstance( wlan0, Ethernet ) # True
print isinstance( eth0, PCI )       # True
print isinstance( eth0, USB )       # False
```

Classes: Exemplo Avançado (Diagrama)



Panorama

Atributos de Classe

- São atributos que estão na classe, não na instância
- São compartilhados entre todas as instâncias (economia de memória)
- Os valores são instanciados/atribuídos ao ler a definição de classe
- Úteis para casos como Jogos, onde uma imagem deve ser compartilhada por todos os personagens idênticos, economizando memória
- Úteis para fazer *"lock"* em regiões críticas, evitar condições de corrida, etc...
- **Perigoso para programadores descuidados!**

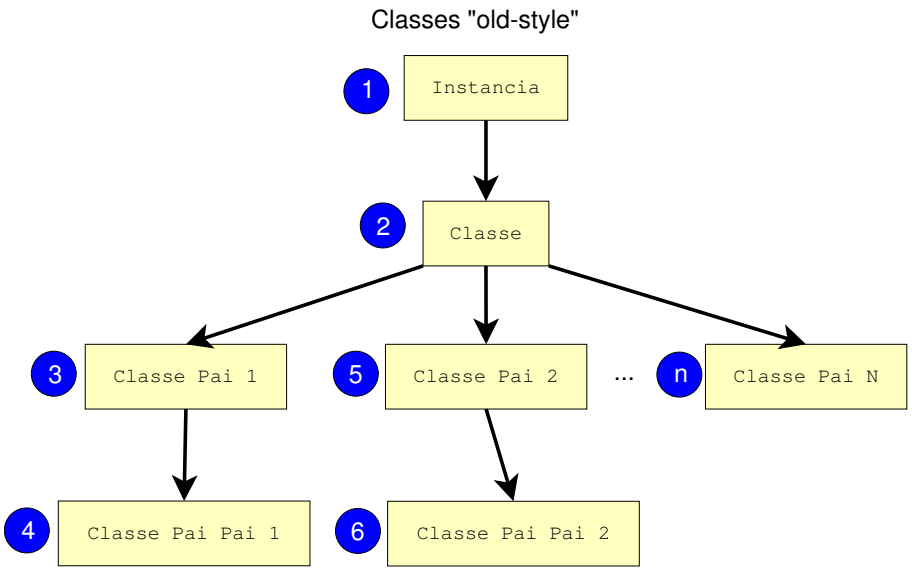
Atributos de Classe: Exemplos

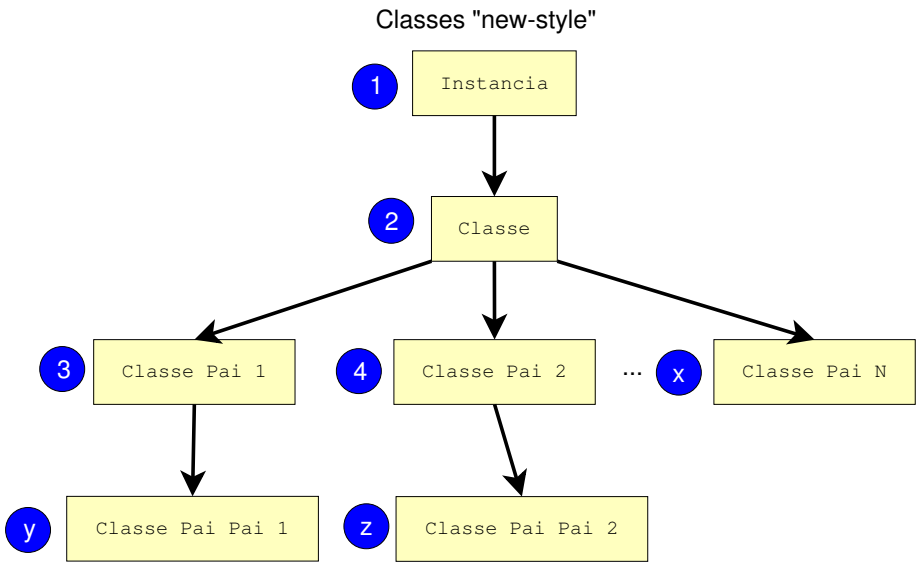
```
class C( object ):
    l = []

c1 = C()
c2 = C()
c1.l.append( 1 )
print c1.l # imprime [1]
print c2.l # imprime [1]
print C.l  # imprime [1]

c2.l.append( 2 )
print c1.l # imprime [1, 2]
print c2.l # imprime [1, 2]
print C.l  # imprime [1, 2]

C.l.append( 3 )
print c1.l # imprime [1, 2, 3]
print c2.l # imprime [1, 2, 3]
print C.l  # imprime [1, 2, 3]
```

Veja o próximo slide, sobre resolução de atributos para entender melhor.

Panorama

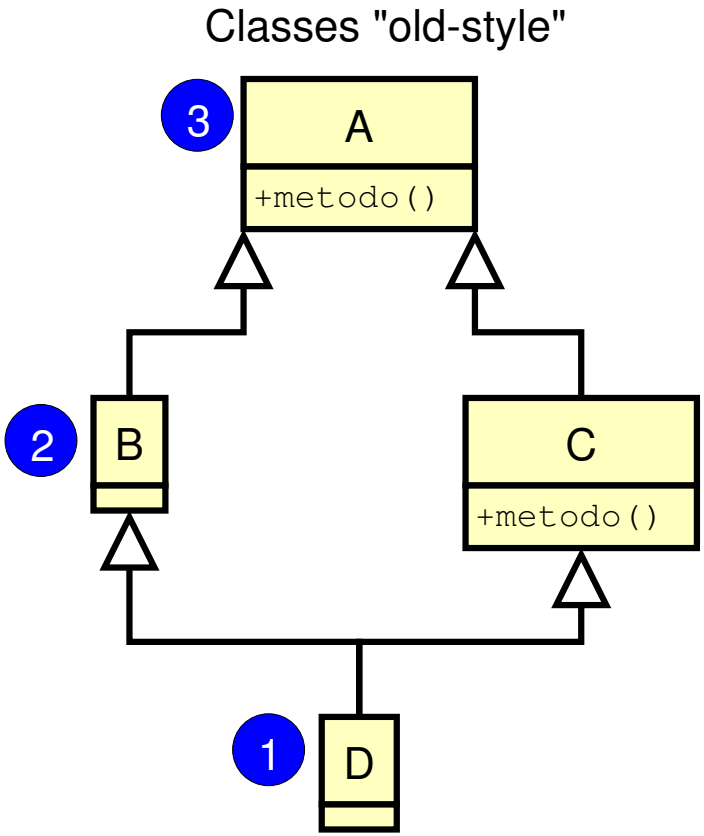
Resolução de Atributos



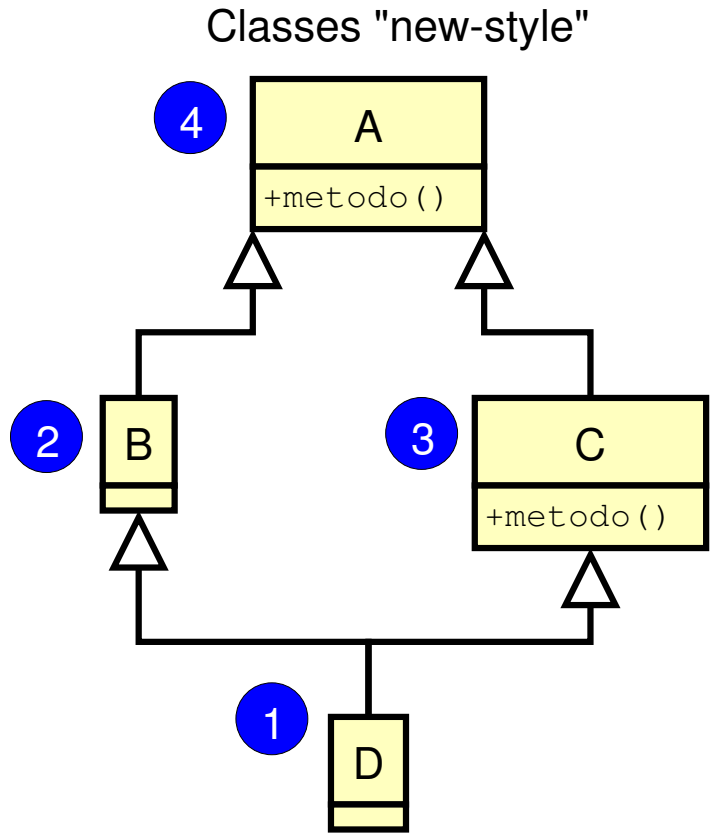


- Em objetos de *"old-style classes"* a pesquisa é recursiva.
- Em objetos de *"new-style classes"* a pesquisa segue uma ordem mais lógica, que pode ser vista em `obj.__mro__`.

Resolução de Atributos: Herança em Diamante




```
d = D()
d.metodo()
```




```
d = D()
d.metodo()
```

- **Old-Style:** Chama `A.metodo()`, o que é um problema, sendo que as funcionalidades de C devem depender de `C.metodo()`.
- **New-Style:** Chama `C.metodo()`, o que é o esperado.

Panorama

Métodos Estáticos

São métodos que não precisam de uma instância ou classe para serem chamados, parece com o conceito de métodos estáticos em C++ e Java. Exemplo:

```
class C( object ):
    def f():
        print 'metodo estatico'
    f = staticmethod( f )

C.f() # imprime 'metodo estatico'
c = C()
c.f() # imprime 'metodo estatico'
```

Atenção

A linha essencial é: `f = staticmethod( f )`, ela redefine o método previamente declarado como um método estático.

Métodos de Classe

São métodos que são chamados sobre a classe, ao invés da instância.
Exemplo:

```
class C( object ):
    def f( classe ):
        print 'metodo da classe:', classe
    f = classmethod( f )

C.f() # imprime 'metodo da classe: <class '__main__.C'>'
c = C()
c.f() # imprime 'metodo da classe: <class '__main__.C'>'
```

Atenção

A linha essencial é: `f = classmethod( f )`, ela redefine o método previamente declarado como um método da classe, ao invés de método da instância.

Panorama

Atributos Públicos e Privados

- Nomenclatura define atributos/métodos públicos e privados
  - **Privados:** nomes que se iniciam com __ e não terminam com __.
  - **Públicos:** os outros nomes possíveis.
  - Convenciona-se que atributos/métodos que se iniciam e terminam com __ (portanto públicos) são de uso interno da classe, apesar de poderem ser utilizados pelo mundo externo.
- A proteção é feita por mistura de nomes e pode ser "burlada" (flexibilidade para o programador):

```
class C( object ):
    __privado_classe = 1
    def __init__( self ):
        self.__privado_instancia = 2

c = C()
print c._C__privado_classe
print c._C__privado_instancia
```

Propriedades: Acesso de Escrita, Leitura dos Atributos

- *"new-style classes"* provê o conceito de propriedade, o qual pode ter os acessos de escrita e leitura controlados por métodos:

```
class C( object ):
    def __init__( self ):
        self.__x = None
    def getX( self ):
        print 'getX()'
        return self.__x
    def setX( self, valor ):
        print 'setX(', valor, ')'
        if isinstance( valor, int ):
            self.__x = valor
        else:
            raise TypeError( 'x precisa ser inteiro!' )
    x = property( getX, setX )

c = C()
c.x = 1
c.x = 'abc' # TypeError: x precisa ser inteiro
print c.x
```

Propriedades: Exemplo mais real

```
def gera_propriedade_tipo( nome, tipo, valida, errmsg ):
    attr = '__%s' % nome
    def get( self ):
        return getattr( self, attr )
    # get()

    def set( self, valor ):
        if isinstance( valor, tipo ):
            if valida( valor ):
                setattr( self, attr, valor )
            else:
                raise ValueError( errmsg )
        else:
            raise TypeError( errmsg )
    # set()

    return property( get, set )
# gera_propriedade_tipo()
```

Propriedades: Exemplo mais real (2)

```
def gera_propriedade_str( nome, tam_max ):
    def valida( valor ):
        return len( valor ) <= tam_max
    # valida()

    msg = ( '%s precisa ser "str" e ter ' \
            'menos que %s letras' ) % ( nome, tam_max )

    return gera_propriedade_tipo( nome, str, valida, msg )
# gera_propriedade_str()


def gera_propriedade_int( nome, min, max ):
    def valida( valor ):
        return min <= valor <= max
    # valida()

    msg = '%s precisa ser "int" entre %s e %s' % \
          ( nome, min, max )

    return gera_propriedade_tipo( nome, int, valida, msg )
# gera_propriedade_str()
```

Propriedades: Exemplo mais real (3)

```
class Pessoa( object ):
    nome  = gera_propriedade_str( "nome", 20 )
    rg    = gera_propriedade_str( "rg", 9 )
    idade = gera_propriedade_int( "idade", 1, 150 )


p = Pessoa()

p.nome = 10      # TypeError
p.nome = "Este texto tem mais de 20 letras" # ValueError
p.idade = "200" # TypeError
p.idade = 200   # ValueError

p.nome  = "Gustavo" # Ok!
p.idade = 22
```

CONTINUE CLASSES!!!

continue-me!!!!

Panorama

Panorama

Lendo um Arquivo

- Lendo todo o conteúdo para a memória:

```
arquivo  = open( 'nome_arq.txt' )
conteudo = arquivo.read()
arquivo.close()
```

- Lendo todas as linhas para a memória (lista):

```
arquivo  = open( 'nome_arq.txt' )
linhas   = arquivo.readlines()
arquivo.close()
```

- Lendo uma quantidade específica de bytes:

```
arquivo  = open( 'nome_arq.txt' )
comeco   = arquivo.read( 20 ) # 20 primeiros bytes
arquivo.close()
```

Lendo um Arquivo (2)

- Lendo com laços:

```
arquivo = open( 'nome_arq.txt' )
for linha in arquivo: # identico a arquivo.readlines()
    print linha,
```

```
arquivo = open( 'nome_arq.txt' )
while True:
    linha = arquivo.readline()
    if linha == '':
        break
    print linha,
```

Panorama

Panorama

Referências utilizadas nesta aula

- Python Tutorial `http://docs.python.org/tut/tut.html`
- Python Library Reference `http://docs.python.org/lib/lib.html`
- Python Language Reference `http://docs.python.org/ref/ref.html`
- Python para já Programadores `http://www.gustavobarbieri.com.br/python/aulas_python/aula-01.pdf`

Panorama

Contato

Gustavo Sverzut Barbieri

Email:	barbieri@gmail.com
Website:	`http://www.gustavobarbieri.com.br`
ICQ:	17249123
MSN:	barbieri@gmail.com
Jabber:	gsbarbieri@jabber.org

Obtenha esta palestra em:
`http://palestras.gustavobarbieri.com.br/python-5hs/`