Programação Orientada a Objeto em Python

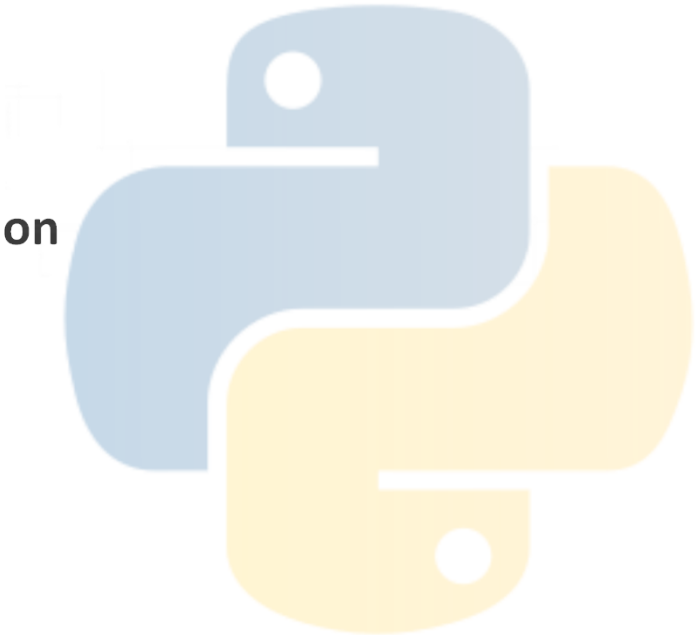
Seminário de LP III

Por: [1]Adriana Nery Ramos, [1]Daniel Andersen Cerqueira Lima,[1]Fabilone Santos da Silva, [1]Janaina C.C. G. Reis, [1]Jonisson S. Santos, [1]Roberta Idelfonso

[1]Graduando em Ciência da Computação- Universidade Estadual de Santa Cruz- UESC

Tópicos Abordados:

Histórico da Linguagem Python
Vantagens da Utilização da Linguagem
Classes em Python
Encapsulamento
Polimorfismo
Herança
Ex: aplicação direta da programação em Python



1.0- Histórico da Linguagem Python

Criada em 1989 pelo holandês Guido Van Rossum no Instituto de Pesquisa Nacional para Matemática e Ciência da Computação (CWI), em Amsterdã, Holanda. Influenciada pela linguagem ABC, desenvolvida no CWI por Guido e outros nas décadas de 70 e 80. ABC tinha um foco bem definido: ser uma linguagem de programação para usuários inteligentes de computadores que não eram programadores: Físicos, Cientistas Sociais e até Linguistas. Também foi levado em consideração que o projeto do sistema operacional distribuído Amoeba precisava de uma linguagem de script.

Com isso surge o Python, uma linguagem interpretada, bastante portável, orientada a objetos (incluindo herança múltipla). Apresenta semântica dinâmica, um moderno mecanismo de tratamento de erros e exceções. Python possui Uma forma eficiente de acesso e reutilização de código com o uso de módulos, coleta de lixo automática, recursos avançados de manipulação de textos, listas e outras estruturas de dados (como dicionários). Python possui ainda uma sintaxe simples e faz muito uso de indentação, o que é uma característica marcante da linguagem.

2.0- Vantagens da Utilização da Linguagem

· **Qualidade do código:** A sintaxe do Python facilita e encoraja a legibilidade do código, o que o torna mais fácil de manter e reutilizar;
· **Produtividade:** é mais rápido desenvolver um mesmo código em Python do que em outras linguagens como o C, C++ ou Java;
· **Código compacto:** requer menos linhas, e não necessita da declaração prévia do tipo das variáveis. Além disso, não requer um ciclo de compilação, podendo o programa ser executado imediatamente;
· **Portabilidade:** muitos programas Python correm sem qualquer alteração em várias plataformas como o Linux e o Windows;
· **Possui uma vasta biblioteca padrão.**
. É uma das linguagens mais usadas por grandes empresas da área de TI.

3.0- Classes em Python

Como nas outras linguagens de programação classes em Python são objetos. Mas em Python todos os tipos de dados são tratados como objetos. Dessa forma uma classe em Python permite que o mecanismo de herança realize múltiplas classes bases, uma classe derivada pode sobrescrever quaisquer métodos de uma classe ancestral, e um método pode invocar outro método homônimo de uma classe ancestral. Python também permite que objetos possam armazenar uma quantidade arbitrária de dados privados e que uma classe seja criada dentro de outra classe.

3.1- Sintaxe de Definição de uma Classe

A forma mais simples de definição de uma classe pode ser descrita dessa maneira:

```
>>> class NomeDaClasse:
        <comando-1>
        .
        .
        .
        <comando-N>
```

3.1- Sintaxe de Definição de uma Classe

EX:

```
>>> class ImprimiDados:                          #definindo a classe
...         def imprimi(self, inf ):              #definindo método da classe
...                 print inf
...         def imprimiMedia(self, inf ):         #definindo método da classe
...                 if inf > 7:
...                         print "Aprovado!"
...                 elif inf < 7:
...                         print "Reprovado!"
...                 else:
...                         print "Aprovado Pelo Conselho!"
>>> x= ImprimiDados()                            #criando uma instância da classe
>>> a= 8                                         #inicializando uma variável
>>> x.imprimi(a)                                 #executando um método da classe
>>> x.imprimiMedia(a)                            #executando um método da classe
8                                                #Saída
Aprovado!
```

3.2- Construtores de uma Classe

Quando uma Classe define um método __init__(), o processo de instanciação acontece automaticamente sobre a Classe.

EX:

```
>>> class ValoresInteiros:                    #definindo a classe
...         valor = int
...         def __init__(self):  #definindo método construtor da classe sem valores
...                 Self.valor  =  0          #instanciando o atributo da classe
-------------------------------------------------------------------------------------------
...         def __init__(self, x): #definindo método construtor com argumentos
...                 Self.valor  =  x          #instanciando o atributo da classe
...        def  imprimi(self):                #instanciando o atributo da classe
...                 print self.valor
>>> x = ValoresInteiros()                     #criando uma instância da classe
>>> a = ValoresInteiros( 2)                   #criando uma instância da classe
>>> x.imprimi()                               #executando um método da classe
>>> a.imprimi()                               #executando um método da classe
0                                             #Saída
2                                             #Saída
```

3.2- Construtores de uma Classe

Além do método construtor __init__(), as classes podem ter os métodos __add__() e __mul__().

EX:

```
>>> class ValoresInteiros:                          #definindo a classe
...     valor = int                                  #atributo da classe
...     def __init__(self, x):                       #método construtor
...         self.valor = x                           #instanciando atributo
...     def __add__(self, parametros):               #método __add__
...         return ValoresInteiros(self.valor + parametros)      #retorno de uma instância
...     def __mul__(self,  parametros):              #método __mul__
...         self.valor = self.valor * parametros     #instanciando atributo
...     def imprimi(self):                           #método imprimi()
...         print self.valor
>>> a = ValoresInteiros(3)                          #criando instâncias da classe com __init__
>>> a.imprimi()                                     #executando método imprimi()
3                                                   #Saída
>>> b = a + 7                                       #criando instâncias da classe com __add__
>>> b.imprimi()                                     #executando método imprimi()
 10                                                 #Saída
>>> a*3                                             #instanciando atributos da classe com __mul__
>>> a.imprimi()                                     #executando método imprimi()
9                                                   #Saída
```

3.3- Sobrecarga de operadores

Esses mesmos métodos podem ser usados para realizar sobrecarga de operadores, ou seja:

EX:

```
>>> class ValoresInteiros:                          #definindo a classe
...     valor = int                                 #atributo da classe
...     soma = int                                  #atributo da classe
...     def __init__(self, x, y):                   #método construtor
...         self.valor = x                          #instanciando atributo
...         self.soma = y                           #instanciando atributo
...     def __add__(self, pr):                      #método __add__
...         return ValoresInteiros(self.valor + pr.valor, self.soma + pr.soma) #ret. de uma
instância
...     def __mul__(self,  pr):                     #método __mul__
...         self.valor = self.valor * pr.valor * pr.soma        #instanciando atributo
...         self.soma = self.valor * pr.valor - pr.soma         #instanciando atributo
```

3.4- Métodos Estáticos

Um método estático em Python é uma atribuição a classe que não precisa do primeiro argumento para ser instanciado na classe.

EX:

```
>>> class ValoresInteiros:                    #definindo a classe
...     valor = int                           #atributo da classe
...     def __init__(self, x):                #método construtor
...         self.valor = x                    #instanciando atributo
...     def imprimi(self):                    #método imprimi()
...         print self.valor
... @staticmethod                             #método estático
...     def main(*argumentos):                #método main
...         c = ValoresInteiros( 7 )          #instanciando
...         c.imprimi()                       #executando método imprimi()
...          return 0                         #final método estático
>>> a = ValoresInteiros( 1 )                  #criando instâncias da classe
>>> a.main( “teste”)                          #executando o método main
7                                             #Saída
```

4- Encapsulamento

A proteção dos atributos ou métodos de uma classe em Python existem somente de duas formas o public e o private e eles são definidos no próprio nome do atributo ou método. Atributos ou métodos iniciados por no máximo dois sublinhados e terminados por um sublinhado são privados e todas as outras formas são públicas.

EX:

```
>>> class A:
...         a = 1                # atributo público
...         __b = 2              # atributo privado a classe A
>>> class B(A):
...          __c = 3             # atributo privado a B
...         def __init__(self):
...         print self.a
...         print self.__c
>>> a = A()
>>> print a.a                    # imprime 1
1                                #saída
>>> b = B()
>>> print b.__b                  # Erro, pois __b é privado a classe A.
>>> print b.__c                  # Erro, __c é um atributo privado, somente chamado pela classe.
>>> print b._B__c                # Imprime __c = 3, muito pouco utilizada, mas existe.
```

5- Polimorfismo

O polimorfismo em Python é muito similar ao polimorfismo em Java, ou seja, eu posso implementar métodos em uma hierarquia de classes.

EX:

```
>>> class Corredor(Atleta):
...     def correr(self):
...         print("Corredor correndo!")

...     def aquecer(self):
...         print("Corredor aquecido!")
```

```
>>> class Nadador(Atleta):
...     def nadar(self):
...         print("Nadador nadando!")

...     def aquecer(self):
...         print("Nadador aquecido!")
```

```
>>> class Ciclista(Atleta):
...     def pedalar(self):
...         print("Ciclista pedalando!")

...     def aquecer(self):
...         print("Ciclista aquecido!")
```

6- Herança

Uma nova classe em Python pode herdar métodos, atributos , etc. de outras classes, assim como implementar novos métodos e criar novos dentro de si mesma. Sendo assim os mecanismos de heranças podem ser:

Herança simples: a classe herda somente uma outra classe

Ex:

```
>>> class Terrestre(object):                    #criando classe
...     moveTerra = True                        #atributos da classe
...     inf = str
...     def __init__(self, objeto):             #construtor
...         self.inf = objeto
...     def imprimiTerrestre(self):             #método
...                 print self.inf
...                 print “Move em Terra: ", self.moveTerra
```

6.1- Herança Simples

Ex:

```
... class Carro(Terrestre):                    #nova classe herdeira
...     def __init__(self, objeto):            #construto
...       super(Carro, self).__init__(objeto)


>>> a= Carro( “Chevrolet”)                     #instanciando
>>> a.imprimiTerrestre()                       #executando método da classe
Chevrolet                                      #saída
Move em Terra:  True
```

6.2- Herança Múltipla

Herança Múltipla: a classe herda mais de uma classe.

Ex:

```
>>> class Terrestre(object):                    #criando classe
...     moveTerra = True                        #atributos da classe
...     inf = str
...     def __init__(self, objeto):             #construtor
...         self.inf = objeto
...     def imprimiTerrestre(self):             #método
...             print self.inf
...             print "Move em Terra: ", self.moveTerra

>>> class Aquatico(object):                     #criando outra classe
...     moveAgua = True                         #atributos da classe
...     infA = str
...     def __init__(self, objeto):             #construtor
...         self.infA = objeto
...     def imprimiAquatico(self):              #método
...             print self.infA
...             print "Move em Água: ", self.moveAgua
```

6.2- Herança Múltipla

Ex:

```
>>>  class Anfibio(Terrestre, Aquatico):                    #definindo classe herdeira
...    def __int__(self, objeto, objetoA):                  #método construtor
...       Terrestre.__init__(self, objeto)                  #método construtor da classe pai
...       Aquatico.__init__(self, objetoA)                  #método construtor da classe pai
...     def imprimi(self):                                  #definindo novo método
...        Terrestre.imprimiTerrestre(self)                 #método da classe pai
...        Aquatico.imprimiAquatico(self)                   #método da classe pai

>>> b = Anfibio("Chevrolet", "Boat 230 Cabinada")          #nova instância da classe Anfibio
>>> b.imprimi()                                            #executando método da classe

Chevrolet                                                  #saída
Move em Terra:  True
Boat 230 Cabinada
Move em Água:  True
```

Obrigado!

Referências Bibliográficas:

LUTZ, Mark. ASCHER, David. **Learning Python**. United States of America.O' Reilly Network, 2003.

KIUSALAAS, Jaan. **Numerical Methods In Engineering With Python.** New York. Cambridge University Press, 2005.

ROSSUM, Guido Van. DRAKE JR., Fred L. Tutorial Python. Disponível em: <http://pythonbrasil.com.br>. Acesso em: 20/10/2010.

LABAKI, Josué. Introdução a Python- Módulo A, grupo de Python- Universidade Estadual Paulista. Disponível em: <http://labaki.tk>. Acesso em: 20/10/2010.

BORGES, L. E. *Python para Desenvolvedores*. Disponível em: <http://ark4n.files.wordpress.com/2009/05/python_para_desenvolvedores.pdf >. Sob licença *Creative Commons*. ROSSUM, G. v. *Tutorial de Python*. Release 2.1. Acesso em: 20/10/2010.

SOUZA, Francisco A. S. Mini Curso de Python, Conceitos Básicos, Estruturas de Dados e Orientação a objetos. Disponível em: <http://www.slideshare.net/franciscosouza/minicurso-de-python>. Acesso em: 25/11/2010.

Python. Disponível em: <http://pt.wikipedia.org/wiki/Python>. Acesso em: 01/11/2010.

Programação Orientada a Objeto. Disponível em: <http://www.python.org.br/wiki/ProgramacaoOrientadaObjetoPython>. Acesso em: 01/11/2010.

BARBIERI, Gustavo. Python. Disponível em: <http://www.gustavobarbieri.com.br/palestras/python-5hs/aula-01.pdf>. Acesso em: 01/11/2010.