[GIM] Curso de Férias - Python

Luiz Eduardo Barros e Victor Matheus Castro

Conteúdo do Dia

- Orientação a Objetos
- Classes
- Instâncias e Métodos
  - Método __init__( )
- Variáveis de Classe
- Polimorfismo e Herança
- Exceções
- Iteradores
- Geradores
- Intermezzo

Orientação a Objetos

- OO: uma forma de conceber seu projeto;
- Objetos em Python apresentam os seguintes atributos, entre outros:

  Tipo: determina os valores que o objeto pode receber e as operações que podem ser executadas nesse objeto;

  Endereço: índice de memória onde se inicia o armazenamento do objeto. Como o valor armazenado nas posições da memória será interpretado, depende do tipo da variável;

  Tempo de vida: intervalo de tempo durante o qual o objeto se encontra vinculado a um endereço de memória.

Classes

- Uma classe corresponde a uma estrutura de dados que representa um conjunto de instâncias definidas por seus atributos e métodos;
- Em Python a classe de um objeto e o tipo de um objeto são sinônimos;

  A classe determina o tipo do objeto e como ele pode ser manipulado;
- Uma classe encapsula dados, operações e semântica;
- A classe é o que faz com que Python seja uma linguagem de programação orientada a objetos;
- O usuário de uma classe manipula objetos instanciados dessa classe somente com os métodos fornecidos por essa classe.

Classes

```
class Fruit(object):
        """Uma classe que faz várias frutas."""
        def __init__(self, name, color, flavor, poisonous):
                self.name = name
                self.color = color
                self.flavor = flavor
                self.poisonous = poisonous
        def description(self):
                print "I'm a %s %s and I taste %s." (self.color, self.name,
self.flavor)
        def is_edible(self):
                if not self.poisonous:
                        print "Sim, eu sou comestível"
                else:
                        print "Não me coma! Sou venenosa"
lemon = Fruit("lemon", "yellow", "sour", False)
lemon.description()
lemon.is_edible()
```

Instâncias e Métodos

- **Objetos** são instanciados pelas classes;
- Cada instância (objeto) em uma programa Python tem seu próprio **namespace**;
- Os nomes no namespace da classe objeto são chamados de **atributos da classe**;
- Funções definidas dentro de uma classe são chamadas de **métodos**.
- O nome em um namespace de instância é chamado de **atributo de instância**;
- Por convenção, o primeiro argumento do método tem sempre o nome **self**. Portanto, os atributos de **self** são atributos de instância da classe.

Métodos

```
class Calculadora(object):
        def __init__(self, a, b):
                self.a = a
                self.b = b

        def soma(self):
                return self.a + self.b

        def subtrai(self):
                return self.a - self.b

        def multiplica(self):
                return self.a * self.b

        def divide(self):
                return self.a / self.b
```

Método __init__:

- O init é um método construtor, ele inicializa o estado de um objeto;
- O método init é invocado sempre que há uma nova instância de uma classe;
- Ele sempre leva pelo menos um argumento, self, que se refere ao objeto que está sendo criado;
- Você pode pensar em init como a função que "dá o boot"em cada objeto criado pela classe;
- Na verdade não estamos apenas definindo o método init mas sobrescrevendo o init da classe base.

Método __init__:

```
>>> class Square(object):
>>>         def __init__(self):
>>>             self.sides = 4

>>> my_shape = Square()
>>> print my_shape.sides
```

Método __init__:

- O método **init** na classe Complex é definido assim: i = Complex(0,1)
- O efeito deste estado é equivalente a isto:

  c = object.__new__(Complex)
  Complex.__init__(c,0,1)
- O método new pertence a classe built-in (interno do Python) que é chamada para criar uma nova instância de object. O método init dentro da classe Complex é chamado para inicializar o estado da nova instância.

Variáveis de classe

- Para quem tem experiência com Java, os atributos de classe em Python, são equivalentes às variáveis 'static';
- Eles podem ser acessados sem precisar criar uma instância dessa classe, e ficam definidas em todo seu escopo.

```
>>> class Idade:
...         i = 18
... >>> Idade.i
18
```

Herança (Derivação)

- As **classes derivadas** são características extremamente úteis;
- A derivação é a definição de uma **classe nova estendendo uma classe existente**;
- O objetivo é de explorar os pontos em comum que existem entre as classes de um programa;
- As classes diferentes podem **compartilhar** valores e operações;
- A classe nova é chamada de **classe derivada** e a classe existente de quem é derivada é chamada de **classe base**;
- Em Python, **deve haver ao menos uma classe base**, mas pode haver mais de uma, formando assim herança múltipla.

Herança (Derivação)

- Python suporta classes clássicas (old-style classes) e classes de novo-estilo (new-style classes);
- Uma **new-style class** é uma classe que é derivada da classe interna do objeto;
- Uma **old-style class** é uma classe que não tem uma classe base ou uma que é derivada somente de outras classes old-style;
- A derivação em Python é indicada incluindo o(s) nome(s) da classe(s) base em parênteses na declaração da classe derivada.

Herança (Derivação)

```
class Pessoa(object):
        FEMALE = 0
        MALE = 1

        def __init__(self, nome, sexo):
                super(Pessoa, self).__init__()
                self._nome = nome
                self._sexo = sexo

        def __str__(self):
                return str(self._nome)

class Pais(Pessoa):

        def __init__(self, nome, sexo, crianca):
                super(Pais, self).__init__(nome, sexo)
                self._crianca = crianca

        def getCrianca(self, i):
                return self._crianca[i]

        def __str__(self):
                pass
```

Herança (Derivação)

- Uma classe derivada **herda todos os atributos** de sua classe base:

  p = Pessoa()
  q = Pais()

- Assim p é uma Pessoa, tem os atributos **_nome** e **_sexo** e o método **str**;
- Além disso, a classe Pais é derivado de Pessoa, então o objeto q também tem os atributos **_nome** e **_sexo**, e o método **str**;
- Uma classe derivada pode **estender** a classe base de diversas maneiras:

  Os novos atributos podem ser usados,
  Os novos métodos podem ser definidos,
  E os métodos existentes podem ser sobrescritos.

Herança (Derivação)

- Se um método for definido em uma classe derivada que tenha o mesmo nome que um método na classe base, o método na classe derivada sobrescreve ele na classe base;

  Por exemplo, o método **str** na classe Pais sobrescreve o método str na classe Pessoa;

  Consequentemente, **str**(p) invoca Pessoa.**str** , visto que o **str**(q) invoca Pais.str;
- Uma instância de uma classe derivada **pode ser usada em qualquer lugar** em um programa onde uma instância da classe base possa ser usado;
- Por exemplo, isto significa que a classe Pais pode ser passada como um parâmetro real a um método que espere receber uma Pessoa.

Polimorfismo

- **Polimorfismo**: "muitas formas". Caracteriza-se quando duas ou mais classes distintas possuem métodos com o mesmo nome.
- O polimorfismo é usado em **classes distintas compartilhando funções em comum**;
- Porque as classes derivadas são distintas, suas execuções podem diferir;
- Entretanto, as classes derivadas compartilham de uma relação comum, instâncias daquelas classes são usadas exatamente na mesma maneira.

Exceções

- Há **situações inesperadas** durante a execução de um programa, isto ocorrerá com todos;
- A solução pode ser codificar **verificações de erros**...
- Entretanto, um algoritmo simples pode tornar-se **ilegível** quando checamos muitos erros;
- As exceções fornecem uma **maneira limpa de detectar e assegurar situações inesperadas**;
- Quando um programa detecta um erro, levanta uma **exceção**.

Exceções

- Em Python, uma **exceção é um objeto** derivado da classe base interna **Exception**;
- Dizemos que um programa gerou ou levantou (raised) uma condição de exceção na forma de um objeto;
- Um programa precisa capturar (catch) tais objetos e tratá-los para que a execução não seja abortada.

Exceções

- Um método levanta uma exceção usando o identificador **raise**: Um identificador raise é similar a um identificador **return**;
- Um identificador **return** representa a terminação normal de um método e o objeto retornado combina o valor do retorno do método;
- Um **identificador raise** representa a terminação anormal de um método e o objeto levantado representa o tipo de erro encontrado.
- Os alimentadores da exceção são definidos usando um bloco **try**: O corpo do bloco **try** será executado até que uma exceção esteja levantada ou até que termine normalmente.

Iteradores

- Nos bastidores, o comando for aplica a função embutida iter() à coleção;
- Essa função devolve um iterador que define o método next(), que acessa os elementos da coleção em sequência, um por vez;
- Quando acabam os elementos, next() levanta uma exceção StopIteration, indicando que o laço for deve encerrar;

Iteradores

```
>>> s = 'abc'
>>> it = iter(s)
>>> it
<iterator object at 0x00A1DB50>
>>> it.next()
'a'
>>> it.next()
'b'
>>> it.next()
'c'
>>> it.next()
Traceback (most recent call last):
File "<stdin>", line 1, in ?
it.next()
StopIteration
```

Geradores

- Funções geradoras (generator) são uma maneira fácil e poderosa de criar um iterador;
- Uma função geradora é escrita como uma função normal, mas usa o comando yield para produzir resultados. (N.d.T. Quando invocada, a função geradora produz um objeto gerador.)
- Cada vez que next() é invocado, o gerador continua a partir de onde parou (ele mantem na memória seus dados internos e a próxima instrução a ser executada).

Geradores

- Como usar um gerador:

```
def inversor(data):
    for index in range(len(data)-1, -1, -1):
        yield data[index]
```

Intermezzo: estilo de codificação

- Para que o código fique mais legível e agradável aos olhos, é comum que haja estilos de codificação, aqui serão mencionados os principais pontos do Intermezzo, utilizado em Python;
- Características:

  Use 4 espaços de recuo, e nenhum tab (4 espaços são um bom meio termo entre indentação estreita);

  Quebre as linhas de modo que não excedam 79 caracteres;

  Deixe linhas em branco para separar as funções e classes, e grandes blocos de código dentro de funções;

  Quando possível, coloque comentários em uma linha própria.

Intermezzo: estilo de codificação

- Características:

  Escreva docstrings;

  Use espaços ao redor de operadores e após vírgulas;

  Nomeie suas classes e funções de modo consistente;

  Não use codificações exóticas se o seu código é feito para ser usado em um contexto internacional.