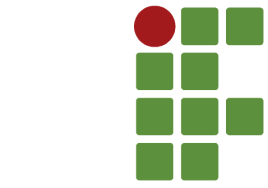
Introdução a Orientação a Objetos - Parte II

Programação Estruturada e Orientada a Objetos



INSTITUTO
FEDERAL
Rio Grande do Norte

Campus
Ceará-Mirim

Prof. Me. Leo Moreira Silva

9 de março de 2020

Agenda

Construtores

Documentação

Encapsulamento

Construtores

- Métodos importantes em classes
- São executados assim que o objeto é instanciado;
- Em Python, possui a seguinte estrutura:

```
def __init__(self):
```

- Geralmente utilizado para a inialização de atributos.

- Classe Carro **sem** construtor:

```
class Carro:
    cor = 'sem cor'
    marca = 'sem marca'
    modelo = 'sem modelo'
    ano = 2010
    km_rodados = 0

    def detalhes(self):
        print('cor:', self.cor)
        print('marca:', self.marca)
        print('modelo:', self.modelo)
        print('ano:', self.ano)
        print('km rodados:', self.km_rodados)
```

```
>>>car_1 = Carro() #Instancia o objeto da classe Carro na variável 'car_1'
>>>car_1.cor = 'Vermelho'
>>>car_1.marca = 'Honda'
>>>car_1.modelo = 'HR-V'
>>>car_1.ano = 2016
>>>car_1.detalhes() #Chama o método 'detalhes' implementado na classe Carro

cor: Vermelho
marca: Honda
modelo: HR-V
ano: 2016
km_rodados: 0
```

- Classe Carro **com** construtor:

```
class Carro:
    def __init__(self, cor, marca, modelo, ano, km_rodados):
        self.cor = cor
        self.marca = marca
        self.modelo = modelo
        self.ano = ano
        self.km_rodados = km_rodados

    def detalhes(self):
        print 'cor:', self.cor
        print 'marca:', self.marca
        print 'modelo:', self.modelo
        print 'ano:', self.ano
        print 'km rodados:', self.km_rodados
```

```
>>> from Carro_construtor import Carro
>>> car = Carro('Azul', 'Honda', 'HR-V', 2016, 2000)
>>> car.detalhes()

cor: Azul
marca: Honda
modelo: HR-V
ano: 2016
km_rodados: 2000
```

Documentação

- Documentar Classes e Métodos
  - Necessário para todos que irão utilizar o código
- Função `help()`:
  - Exibe documentação de um método/classe

```
>>> help(math.cos)
Help on built-in function cos in module math:
cos(...)
    cos(x)
    Return the cosine of x (measured in radians).
```

- Documentação em Python: **docstrings**

```
class Carro:
    '''
    Classe que representa um carro. Cada carro possui:
        -cor
        -marca
        -modelo
        -ano
        -km_rodados
        -statusMotor
        -statusMovimento
    '''
```

```
def andar(self):
    '''Método que coloca o carro em movimento
    Verifica antes se o carro está ligado ou desligado'''
    if(self.statusMotor == True):
        if(self.statusMovimento == True):
            print 'O carro já está em movimento!'
        else:
            self.statusMovimento = True
            print 'Carro em movimento!'
    else:
        print 'Necessário ligar o motor!'
```

```
class Carro
    | Classe que representa um carro.
    | Cada carro possui:
    |   -cor
    |   -marca
    |   -modelo
    |   -ano
    |   -km_rodados
    |   -statusMotor
    |   -statusMovimento
    |
    | Methods defined here:
    |
    | andar(self)
    |   Método que coloca o carro em movimento
    |   Verifica antes se o carro está ligado ou desligado
```

Encapsulamento

- Encapsulamento de dados é a proteção dos atributos e métodos de uma Classe
- Seu objetivo é restringir o acesso direto à informação
- Existem dois tipos de atributos em OO - Python:
  - Público
  - Privado

Encapsulamento

- Atributos Públicos
  - Podem ser acessados diretamente
  - Não existe restrição quanto a escrita e leitura deles
- Atributos privados
  - São acessados via métodos
  - Restrição de leitura e escrita aos dados de forma direta

```
class Carro:
    #Atributo público
    cor = 'azul'

    #Atributo privado
    __nomeProprietario = 'Leo Silva'
```

- Se o atributo é privado, como acessar?
- Via **métodos**

```
class Carro:
    #Atributo público
    cor = 'azul'
    #Atributo privado
    __nome_proprietario = 'Leo Silva'

    def get_nome_proprietario(self):
        return self.__nome_proprietario

    def set_nome_proprietario(self, novo_nome_proprietario):
        self.__nome_proprietario = novo_nome_proprietario
```

Encapsulamento

- Métodos GET/SET:
  - Utilizados para acesso de leitura (GET) e escrita (SET) de atributos privados
- Métodos definidos da seguinte forma:
  - get_nome_do_atributo(self)
  - set_nome_do_atributo(self, novo_nome_do_atributo)

Dúvidas?