



Algoritmos Computacionais

Conteúdo: Introdução à programação orientada a objetos

Prof. Dsc. Giomar Sequeiros
giomar@eng.uerj.br

Classes

Definição de classes em Python

- Regras sintáticas para definição de uma **classe**

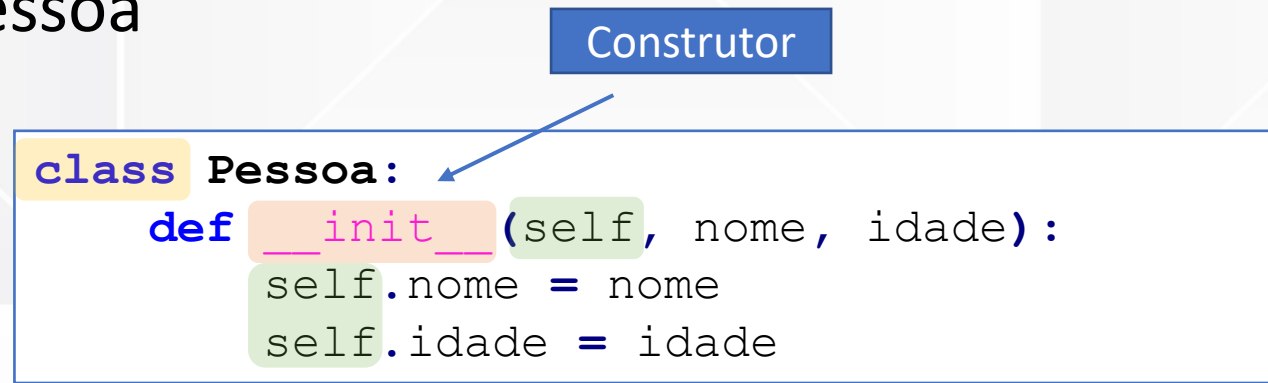
1. **cabeçalho**: palavra reservada **class** , seguida do nome da classe;
2. o **método especial (construtor)** **__init__** é utilizado para **inicializar objetos** da classe; é chamado automaticamente quando um novo objeto da classe é criado;
3. **parâmetro** **self** : utilizado para nos referirmos ao objeto que acabou de ser criado

Definição de uma Classe em Python

- Exemplo: Classe Pessoa

Construtor

```
class Pessoa:
    def __init__(self, nome, idade):
        self.nome = nome
        self.idade = idade
```

A diagram showing a Python class definition for 'Pessoa'. The class name 'Pessoa' is highlighted in yellow. The method name '__init__' is highlighted in pink. The parameters 'self', 'nome', and 'idade' are highlighted in green. A blue box labeled 'Construtor' has an arrow pointing to the '__init__' method.

- No corpo de `__init__()`, existem duas instruções usando a variável `self`:
- `self.nome = nome`** cria um **atributo** chamado `nome` e atribui a ele o valor do parâmetro `nome`.
- `self.idade = idade`** cria um **atributo** chamado `idade` e atribui a ele o valor do parâmetro `idade`.

Criando instâncias em Python

- Considerando a classe Pessoa

```
class Pessoa:  
    def __init__(self, nome, idade):  
        self.nome = nome  
        self.idade = idade
```

- Criamos as instâncias p1 e p2

```
# Criando instâncias de Pessoa  
p1 = Pessoa("João", 25)  
p2 = Pessoa("Maria", 25)
```

- Podemos acessar os atributos usando ponto “.”:

```
>>> p1.nome  
'João'  
>>> p1.idade  
25
```

Conceitos básicos: Métodos em classes

- Definir um método em uma classe, basta incluir a **definição** da **função** seguindo o escopo de bloco da classe.
- Em todos métodos associados à instância definido dentro de uma classe devem ter o **argumento self** definido como primeiro argumento.
- Há geralmente um **método especial __init__** definido na maioria das classes.

Métodos em classes em Python

- Consideremos a classe Pessoa e adicionamos o método **aniversariar()**

```
class Pessoa:  
    def __init__(self, nome, idade):  
        self.nome = nome  
        self.idade = idade  
  
    def aniversariar(self):  
        self.idade += 1
```

- Criamos as instâncias p1 e acessamos o atributo e método usando ponto “.”:

```
>>> p1 = Pessoa("João", 25)  
>>> p1.idade  
25  
>>> p1.aniversariar()  
>>> p1.idade  
26
```


Métodos em classes: Exemplo 1

- Considere a classe **Retangulo** e **Quadrado**

```
class Retangulo:
    def __init__(self, ladoA, ladoB):
        self.ladoA = ladoA
        self.ladoB = ladoB

    def area(self):
        return self.ladoA * self.ladoB

    def perimetro(self):
        return 2*self.ladoA + 2*self.ladoB
```

Saída:

```
>>> r = Retangulo(2,3)
>>> r.area()
6
>>> r.perimetro()
10
```

```
class Quadrado:
    def __init__(self, lado):
        self.lado = lado

    def area(self):
        return self.lado * self.lado

    def perimetro(self):
        return 4 * self.lado
```

Saída:

```
>>> c = Quadrado(5)
>>> c.area()
25
>>> c.perimetro()
20
```


Métodos em classes: Exemplo 2

- Conta bancária

atributos

```
class Conta:
```

```
    def __init__(self, cliente, numero, saldo = 0):  
        self.saldo = 0  
        self.cliente = cliente  
        self.numero = numero  
        self.operacoes = []  
        self.deposito(saldo)
```

métodos

```
    def saque(self, valor):  
        if self.saldo >= valor:  
            self.saldo -= valor  
            self.operacoes.append(["Saque", valor])  
  
    def deposito(self, valor):  
        self.saldo += valor  
        self.operacoes.append(["Deposito", valor])  
  
    def extrato(self):  
        print("Extrato Conta N° %s\n" % self.numero)  
  
        for op in self.operacoes:  
            print("%10s %10.2f" % (op[0], op[1]))  
        print("\n Saldo: %10.2f\n" % self.saldo)
```

Métodos em classes: Exemplo 2 (cont.)

- Conta bancária

```
class Cliente:  
    def __init__(self, nome, telefone):  
        self.nome = nome  
        self.telefone = telefone
```

- Teste

```
joao = Cliente("João da Silva", "(21) 99123-4567")  
conta1 = Conta(joao, 1, 1000)  
conta1.saque(80)  
conta1.extrato()  
conta1.saque(240)  
conta1.deposito(50)  
conta1.extrato()
```

Saída:

```
Extrato Conta N° 1  
Deposito      1000.00  
Saque         80.00  
Saldo:        920.00
```

```
Extrato Conta N° 1  
Deposito      1000.00  
Saque         80.00  
Saque        240.00  
Deposito       50.00  
Saldo:       730.00
```

Exercício 1

Escreva uma **classe** em Python denominada **Elevador** para armazenar as informações de um elevador dentro de um prédio. A classe deve armazenar o andar atual (térreo = 0), total de andares no prédio, excluindo o térreo, capacidade do elevador, e quantas pessoas estão presentes nele. A classe deve também disponibilizar os seguintes métodos:

- a) Um **construtor** que deve receber como parâmetros a capacidade do elevador e o total de andares no prédio (os elevadores sempre começam no térreo e vazio);
- b) Método **entrar()**: para acrescentar uma pessoa no elevador (só deve acrescentar se ainda houver espaço);
- c) Método **sair()**: para remover uma pessoa do elevador (só deve remover se houver alguém dentro dele);
- d) Método **sobe()**: para subir um andar (não deve subir se já estiver no último andar);
- e) Método **desce()**: para descer um andar (não deve descer se já estiver no térreo);
- f) Crie uma função **main()** (fora da classe) para testar a funcionalidade da classe Elevador

Exercício 1: solução

- Criamos a classe Elevador

```
class Elevador:
    def __init__(self, capacidade, totalAndares):
        self.capacidade = capacidade
        self.totalAndares = totalAndares
        self.andarAtual = 0
        self.pessoasPresentes = 0
```

Exercício 1: solução

- Acrescentamos os métodos entrar e sair na classe Elevador

```
def entrar(self):  
    if self.pessoasPresentes < self.capacidade:  
        self.pessoasPresentes += 1  
    else:  
        print('Erro...capacidade insuficiente')  
  
def sair(self):  
    if self.pessoasPresentes > 0:  
        self.pessoasPresentes -= 1  
    else:  
        print('Erro...elevador vazio')
```

Exercício 1: solução

- Acrescentamos os métodos sobe e desce na classe Elevador

```
def sobe(self):  
    if self.andarAtual < self.totalAndares:  
        self.andarAtual += 1  
    else:  
        print('Erro...já está no último andar')  
  
def desce(self):  
    if self.andarAtual > 0:  
        self.andarAtual -= 1  
    else:  
        print('Erro...já está no térreo')
```

Exercício 1: solução

- Acrescentamos o método imprimir dentro da classe Elevador

```
def imprimir(self):  
    print('-----Dados do elevador-----')  
    print('Capacidade:',self.capacidade)  
    print('totalAndares:',self.totalAndares)  
    print('Andar atual:',self.andarAtual)  
    print('Pessoal presentes no elevador:',self.pessoasPresentes)
```


Exercício 1: solução

- Fora da classe Elevador criamos o teste main

```
if __name__ == '__main__':  
    # Criamos uma instância de elevador  
    meuElevador = Elevador(4, 10)  
    # testamos os métodos  
    meuElevador.imprimir()  
    meuElevador.entrar()  
    meuElevador.sobe()  
    meuElevador.imprimir()  
    meuElevador.sair()  
    meuElevador.desce()  
    meuElevador.imprimir()
```

Exercício 2

Escreva uma classe que represente uma **Moto**, com atributos marca, modelo, cor, marcha (que indica em que marcha a Moto se encontra no momento, sendo representado de forma inteira, onde 0 - neutro, 1 – primeira, 2 – segunda, etc), menorMarcha (indica qual será a menor marcha possível para a moto), maiorMarcha, (indica qual será a maior marcha possível) e o atributo ligada (que terá a função de indicar se a moto está ligada ou não. Este atributo deverá ser do tipo booleano).

- a) Adicione um método **construtor** que permita a definição de todos os atributos no momento da instanciação do objeto.
- b) Adicione os métodos **marchaAcima** e **marchaAbaixo** que deverão efetuar a troca de marchas, onde o método marchaAcima deverá subir uma marcha, ou seja, se a moto estiver em primeira marcha, deverá ser trocada para segunda marcha e assim por diante. O método marchaAbaixo deverá realizar o oposto, ou seja, descer a marcha. É importante que estes métodos não permitirem a troca de marchas para valores abaixo da menorMarcha e acima da maiorMarcha.
- c) Adicione os métodos **ligar** e **desligar** que deverão mudar o conteúdo do atributo ligada conforme o caso.
- d) Crie uma função **main()** (fora da classe) para testar a funcionalidade da classe Elevador