



Paradigma

Orientado a Objetos

Classes, Objetos, Métodos e Construtores



A decorative pattern of hexagons in various shades of blue and cyan, some solid and some outlined, arranged in a cluster on the left side of the slide. A small network icon is also visible near the top left.

1

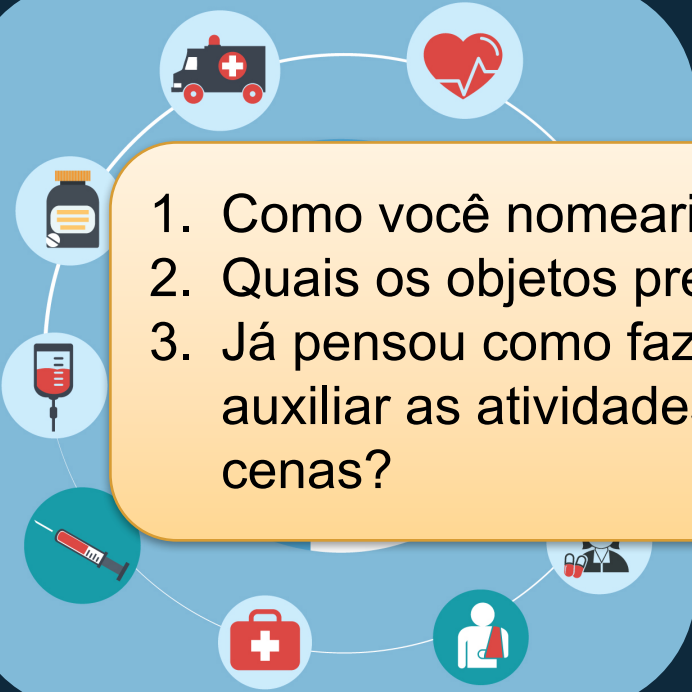

Introdução

Orientação a objetos como abstração do mundo real

Observe as duas cenas




Identifique

- 
- 
1. Como você nomearia as duas cenas?
 2. Quais os objetos presentes nas cenas?
 3. Já pensou como fazer um programa de computador para auxiliar as atividades que as pessoas desempenham nas cenas?



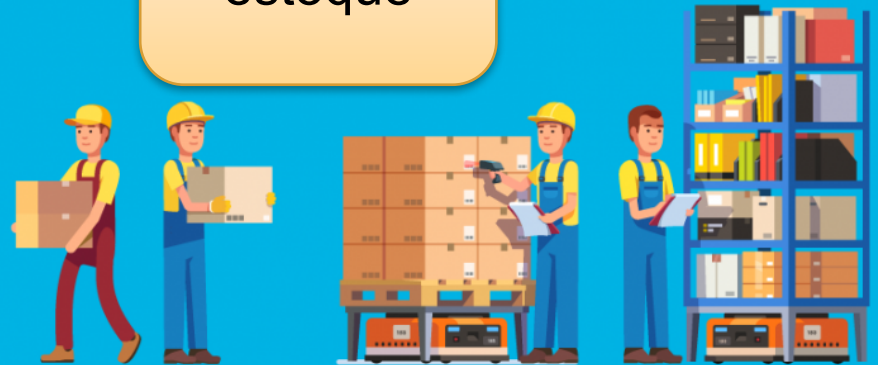
Sistemas



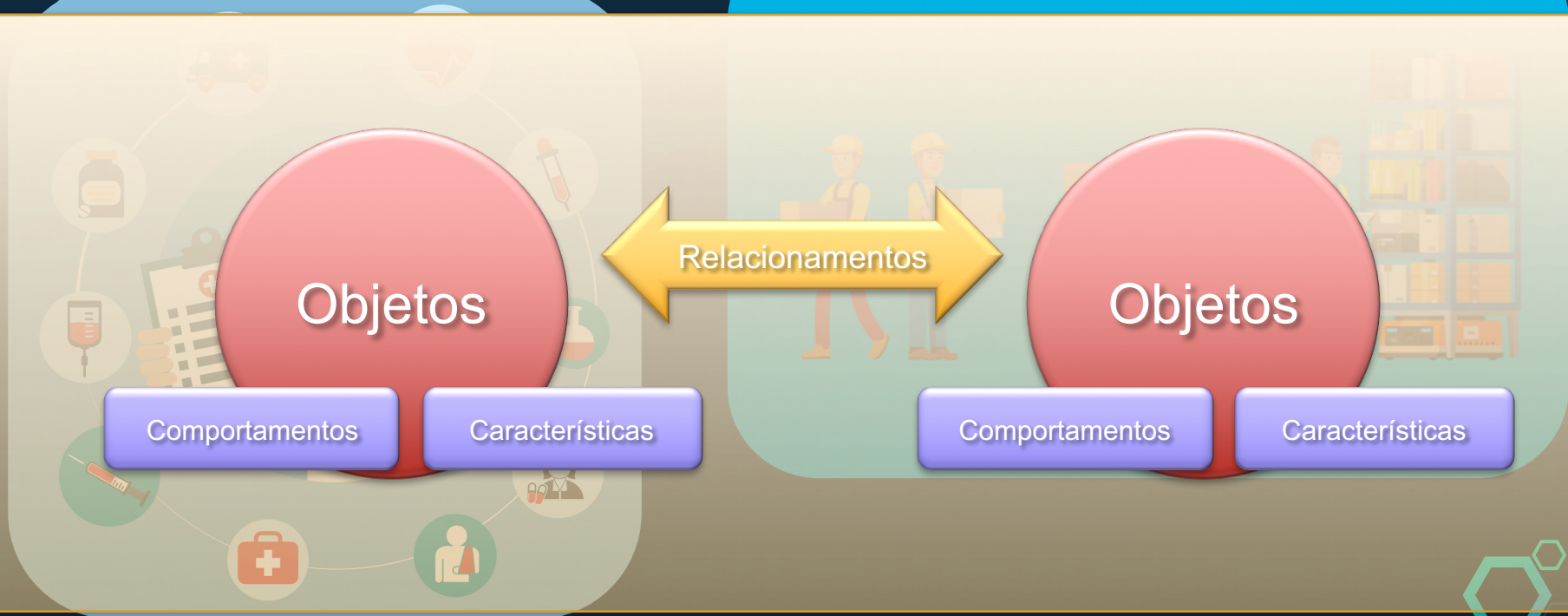
A central illustration of a person's head is surrounded by a circular flow of medical icons: an ambulance, a heart with a pulse line, a syringe, a pill bottle, a blood bag, and a first aid kit. A clipboard with a red cross is also shown.

Sistema de gerenciamento de pacientes em um hospital

Controle de estoque



Objetos do mundo real





Paradigma Orientado a Objetos

- Conjunto de princípios
 - orientam a criação de sistemas computacionais, objetos que interagem entre si.
- Em termos de LPs, conceitos formais surgem com Simula 67, sendo consolidados com Smalltalk





Paradigma Orientado a Objetos

- Popularizado com a difusão de interfaces gráficas de usuários (GUIs)
 - surgimento de ferramentas com suporte para desenvolvimento de aplicações gráficas (C++, FoxPro, Delphi).
- Suportado por várias linguagens (ex: Python, Ruby, C#)
 - atualmente sua maior expressão comercial é dada pelo Java





Pilares da OO

- Conceitos fundamentais (pilares) que norteiam o desenvolvimento OO:

- Abstração;
- Encapsulamento;
- Herança;
- Polimorfismo.





Abstração

- Representação de uma entidade do mundo real, com seu comportamento e características.
- “Modelos Mentais”
 - ❑ Classes;
 - ❑ Objetos;
 - ❑ Métodos;
 - ❑ Atributos.





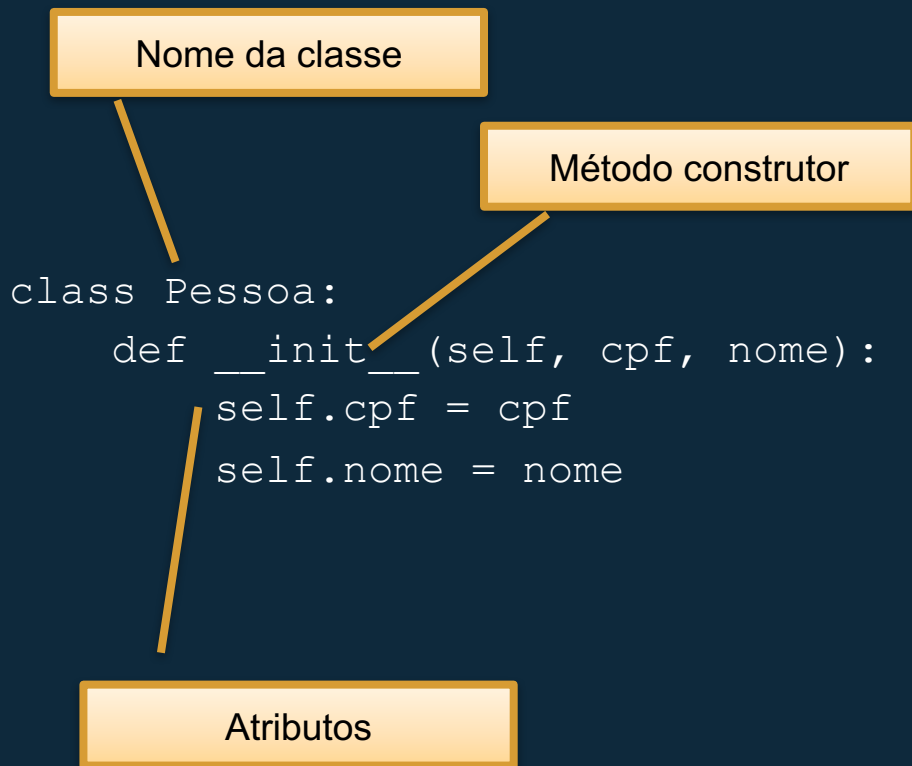
Classes

- Uma classe pode ser entendida como um módulo ou uma estrutura de dados abstrata.
- Uma visão mais ampla pode levar à seguinte definição:
 - Uma classe é um tipo abstrato de dados, que reúne objetos com características similares.
 - O comportamento destes objetos é descrito pelo conjunto de métodos disponíveis.
 - O conjunto de atributos da classe descrevem as características de um objeto.



Classes

Considere a definição de uma classe Pessoa. Existem diversas pessoas, e cada um deles se diferencia pelo nome e cpf.





Objetos

Um objeto pode ser entendido como um ser, lugar, evento, coisa ou conceito do mundo real que possa ser aplicável a um sistema.

- ◇ É comum que haja objetos diferentes com características semelhantes. Esses objetos são agrupados em *classes*.
- ◇ **Classes são um agrupamento de objetos com características similares!**
- ◇ **Objetos são entidades (*instâncias*) únicas de uma classe!**



Objetos

- ◆ Considere a classe `Pessoa` podemos ter um objeto `p1`, com cpf `123.456.789-10` e nome `João da Silva`.

variável que armazena
objeto criado

Instanciação (criação) de um objeto da
classe `Pessoa`

```
p1 = Pessoa('123.456.789-10', 'João da  
Silva')
```





Atributos

- Um atributo é uma característica de um grupo de entidades do mundo real, agrupados em uma classe.
- Uma atributo pode ser um valor simples (um inteiro, por exemplo) ou estruturas complexas (um outro objeto, por exemplo).





Atributos

Considere a classe Pessoa.
Seus atributos são o cpf e o nome.

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



Atributos





Atributos de classe

- ◇ Em geral, os atributos pertencem a cada objeto instanciado, ou seja, a cada nova instanciação de uma mesma classe, cada instância pode ter valores distintos para cada atributo.
- ◇ Atributos de classe são definidos para terem o mesmo valor para todas as instâncias de uma classe.



Exemplo: Atributos de classe

```
class Pessoa:
```

```
    __total_pessoas = 0
```

```
    def __init__(self, cpf, nome):
```

```
        self.cpf = cpf
```

```
        self.nome = nome
```

```
        Pessoa.__total_pessoas += 1
```

```
    def get_total_pessoas(self):
```

```
        return Pessoa.__total_pessoas
```

Em Python, atributos de classe devem ser definidos com dois *'underlines'* como prefixo do nome da variável

Atributos de instância. Esses atributos receberão valores distintos a cada instanciação

Forma de acessar atributo de classe

```
p1 = Pessoa('123.456.789-10', 'Fulano Ciclano', '01/02/1995')  
print(p1.get_total_pessoas()) # erro  
print(Pessoa.get_total_pessoas(p1)) # OK
```

Exemplo: Atributos de classe

```
class Pessoa:
```

```
    __total_pessoas = 0
```

Em Python, atributos de classe devem ser definidos com dois *'underlines'* como prefixo do nome da variável

Atributo de classe
instância de classe
atributo de instância
receber valores
a cada instância

Pensem na utilidade de atributos de classe!

cessar
classe

```
p1 = Pessoa('123.456.789-10', 'Fulano Ciclano', '01/02/1995')  
print(p1.get_total_pessoas()) # erro  
print(Pessoa.get_total_pessoas(p1)) # OK
```



Métodos

- Semelhante a uma função, é a implementação de uma ação da entidade representada pela classe;
- Conjunto de métodos define o comportamento dos objetos de uma classe.



Métodos

```
from datetime import datetime
```

```
class Pessoa:
```

```
    def __init__(self, cpf, nome, data_nascimento):  
        d, m, a = data_nascimento.split("/")
```

```
        self.cpf = cpf
```

```
        self.nome = nome
```

```
        self.data_nascimento = datetime(a, m, d)
```

```
    def get_data_nascimento(self):
```

```
        return self.data_nascimento.strftime("%x")
```

Método que
retorna a data de
nascimento de
uma pessoa. O
self. indica que o
atributo pertence
ao objeto.
Semelhante ao
this. do Java

Exercício

Quais classes, atributos e métodos ?





Construtores

É um método especial para a criação e inicialização de uma nova instância de uma classe.

Um construtor inicializa um objeto e suas variáveis, cria quaisquer outros objetos de que ele precise, garantindo que ele seja configurado corretamente quando criado.

Na maioria das LPs, o construtor é um método que tem o mesmo nome da classe, que geralmente é chamado quando um objeto da classe é declarado ou instanciado.



Exemplo: Construtores

```
from datetime import datetime
```

```
class Pessoa:
```

```
    def __init__(self, cpf, nome, data_nascimento):
```

```
        d, m, a = map(int, data_nascimento.split("/"))
```

```
        self.cpf = cpf
```

```
        self.nome = nome
```

```
        self.data_nascimento = datetime(a, m, d)
```


Esse construtor recebe 3 parâmetros reais, o primeiro (não contado aqui), indicado como *self*, serve como referência para o próprio objeto.





Exemplo: Construtores

```
p1 = Pessoa('123.456.789-10', 'Fulano Ciclano', '01/02/1995')  
p2 = Pessoa('123.456.789-11', 'Ciclano Fulano', '31/12/1996')
```



A cada instanciação de um objeto Pessoa, o método `__init__` é invocado, com os respectivos parâmetros



Exemplo: Construtores

Imagine o caso de um prontuário médico, poderíamos ter, adicionalmente, as seguintes classes:

```
class Medicamento:
```

```
    def __init__(self, nome):  
        self.nome = nome
```

```
class Prontuario:
```

```
    def __init__(obj, paciente):  
        obj.paciente = paciente  
        obj.medicamentos = []
```

Observe que, ao invés de usar *self*, foi utilizado *obj* como referência ao objeto. Não se trata de uma palavra reservada, mas do primeiro argumento de um método do objeto. Costuma-se utilizar *self*.





Destrutores

De forma similar aos construtores, os destrutores são métodos fundamentais das classes, sendo geralmente chamados quando termina o tempo de vida do objeto.

Em algumas linguagens, como C++, ocupam papel tão importante quanto os construtores, por conta da necessidade de desalocação de memória.

Em outras linguagens, como Java, o **Garbage Collector** (Coletor Automático de Lixo) faz esse papel, desalocando aquilo que não é mais utilizado. Há o método `finalize()`, mas raramente é utilizado (há dúvidas se sempre funciona, inclusive).





Destrutores

Tanto os construtores, quanto os destrutores são métodos que não precisam ser definidos em Orientação a Objetos em Python, caso o comportamento esperado seja o padrão.

Geralmente, define-se o construtor para a passagem de argumentos na criação do objeto. Já o destrutor não se costuma definir.

Caso seja necessário realizar algum procedimento na destruição do objeto, define-se o método destrutor, como será exemplificado.





Destrutores

```
p1 = Pessoa('123.456.789-10', 'Fulano Ciclano',  
            '01/02/1995')
```

```
p2 = Pessoa('123.456.789-11', 'Ciclano Fulano',  
            '31/12/1996')
```

```
del p1      # nesse caso, o objeto instanciado é desalocado da memória (destruído)
```

```
del p2      # nesse caso, o objeto instanciado é desalocado da memória (destruído)
```





Destrutores

```
p1 = Pessoa('123.456.789-10', 'Fulano Ciclano',  
            '01/02/1995')
```

```
p2 = Pessoa('123.456.789-11', 'Ciclano Fulano',  
            '31/12/1996')
```

```
del p1      # nesse caso, o objeto instanciado é desalocado da memória (destruído)
```

```
del p2      # nesse caso, o objeto instanciado é desalocado da memória (destruído)
```





Destrutores

```
class A:
```

```
    def __init__(self):  
        print ("A has been created")
```

```
    def __del__(self):  
        print ("A has been destroyed")
```

Método destrutor dos objetos da classe A.

```
a = A()  
del a
```

Quando o objeto a for destruído, a mensagem definida no destrutor será exibida





Garbage Collection em Java

- ◇ Em C++ a memória é alocada e desalocada explicitamente
- ◇ Java possui gerenciamento automático de memória, realizado pela JVM
 - Evita vazamento de memória e bugs de ponteiros
 - Consome recursos computacionais quanto à decisão de desalocação
 - É um processo "não determinístico"





Referência Bibliográfica

Sebesta, R. W. (2011). *Conceitos de Linguagens de Programação*. 9 ed. Bookman.

Capítulo 12

Perkovic, L. *Introdução à Computação Usando Python - Um Foco no Desenvolvimento de Aplicações*. Editora LTC, 1. ed., 2016.

Capítulos 8 e 9.

