



# Introdução a Orientação a Objetos em Python

**Disciplina:** Estrutura de Dados

**JOSÉ TRAJANO MENDES NETO**

Bacharel em Computação

Esp. em Informática na Saúde




# Conteúdo desta Aula

1. Por que utilizar Python?
2. Introdução a OO em Python
3. Classe e Objetos
4. Classe Abstrata
5. Métodos especiais
6. Composição
7. Associação
8. Agregação
9. Polimorfismo




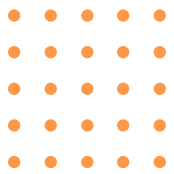



# Por que utilizar Python?

- Simplicidade e legibilidade
  - Ampla comunidade e suporte
  - Versatilidade
  - Grande ecossistema de bibliotecas
  - Produtividade
  - Facilidade de aprendizado
- 
- 
- 



# Características Importantes da Linguagem de Programação Python

- Não é necessário declarar as variáveis;
  - A linguagem Python é *case sensitive*;
  - A indentação faz parte da sintaxe da linguagem;
  - Uso de parênteses é opcional nas operações condicionais.
- 
- 
- 



# Introdução: Conceitos Básicos de Orientação a Objetos (OO)

A Orientação a Objetos é um paradigma de programação que se baseia no conceito de "objetos".

**Classes**

**Objetos**

**Encapsulamento**

**Polimorfismo**

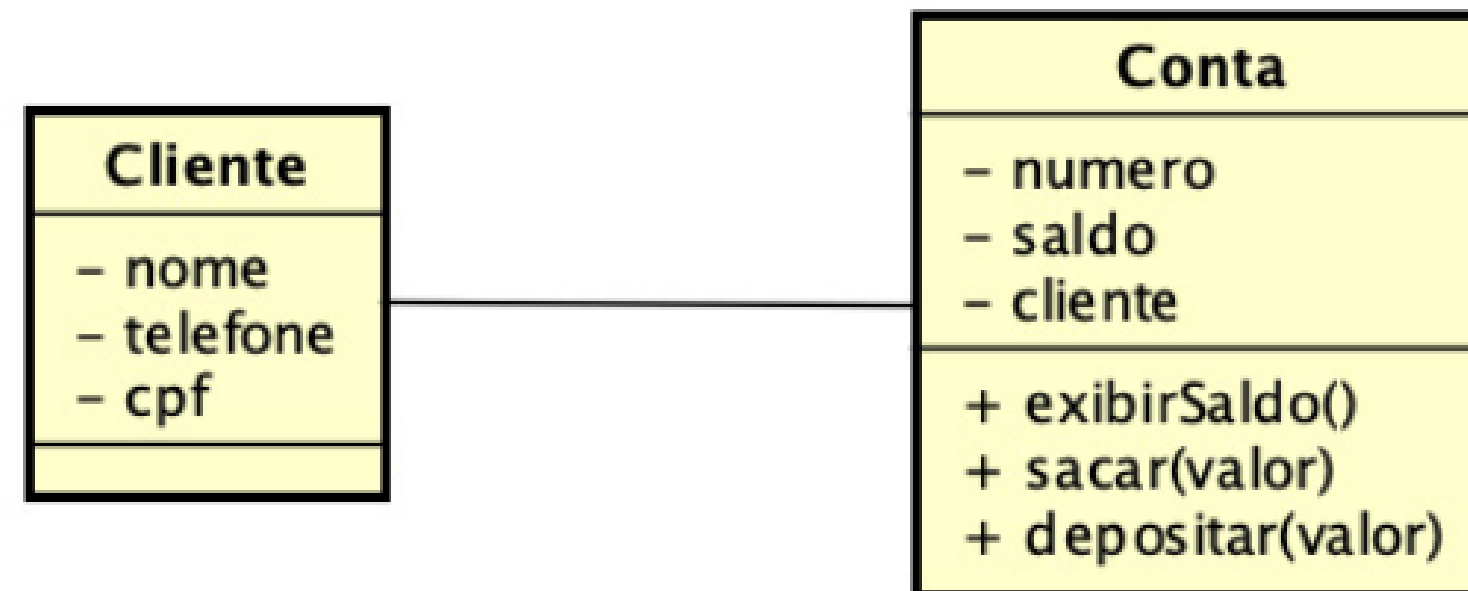
**Herança**



# Classe e Objetos

Embora o nome do paradigma seja Programação Orientada a Objetos, o conceito central é o de classe. É necessário que se defina uma classe para então se criar objetos com base nela.

Dizemos que um objeto é a instância de uma classe. Em uma agência bancária, por exemplo, temos os clientes e suas respectivas contas.





# Classe Abstrata

É uma classe que não pode ser instanciada diretamente, ou seja, você **não pode criar objetos diretamente a partir dela**. Em vez disso, ela serve como uma classe base para outras classes mais específicas, que devem implementar os métodos abstratos definidos na classe abstrata.

As classes abstratas **são usadas para definir uma interface comum** para um grupo de subclasses relacionadas, garantindo que todas as subclasses implementem determinados métodos.

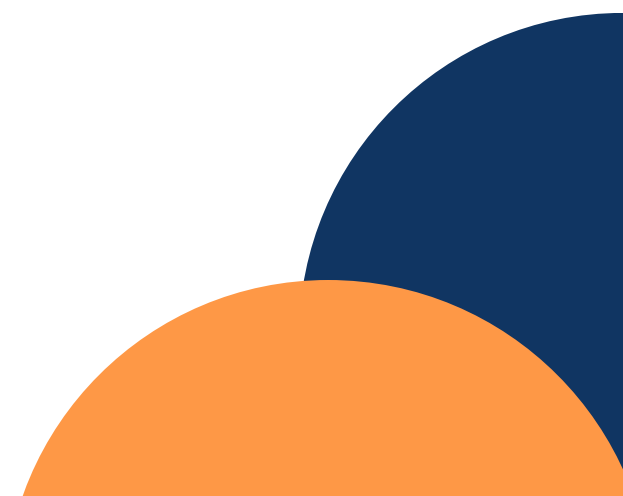
Em Python, a classe abstrata é geralmente implementada usando o módulo **abc (Abstract Base Classes)**, que fornece a funcionalidade para definir classes abstratas e métodos abstratos.





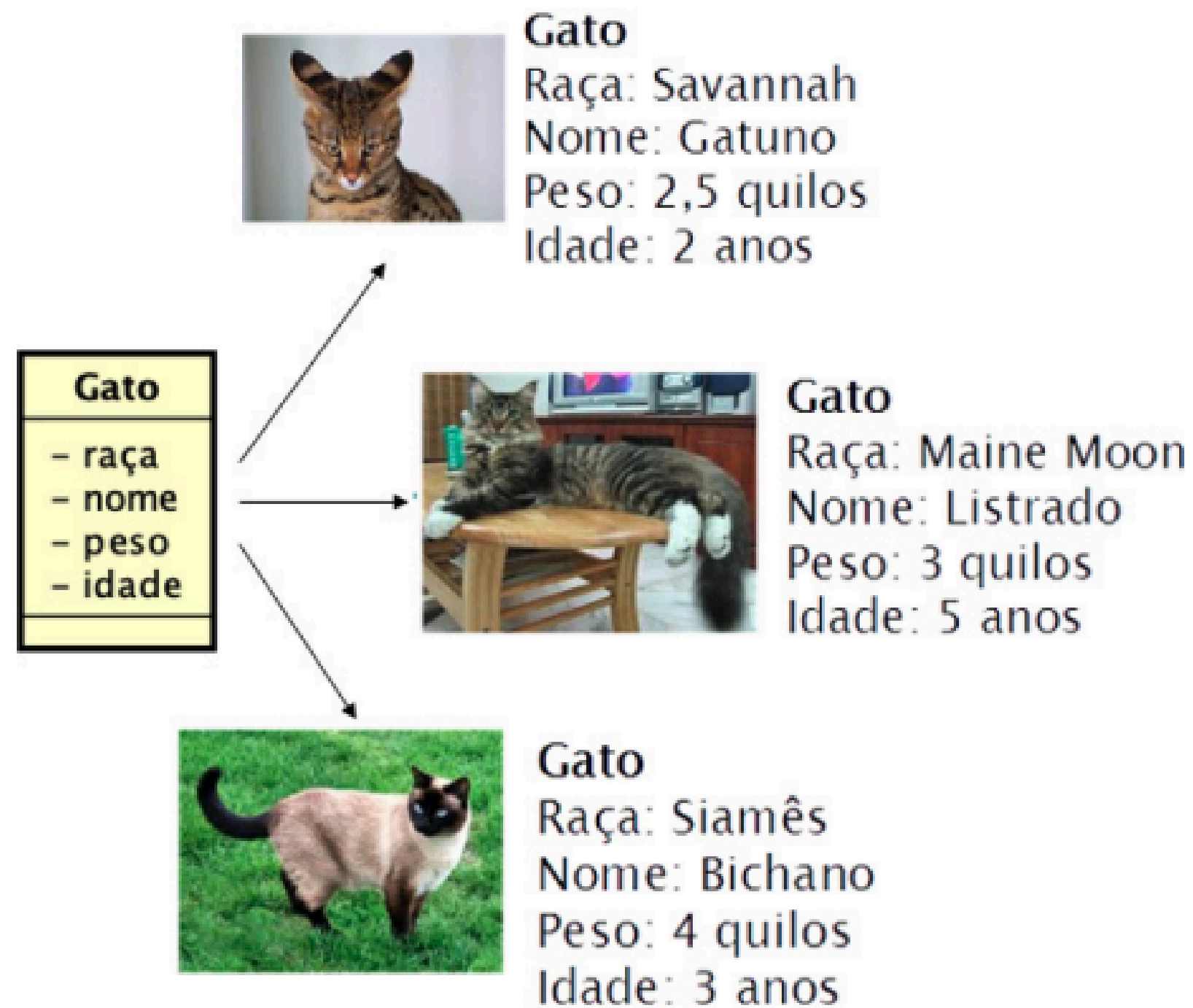
# Atributos

São a identidade do objeto e os comportamentos como sendo os métodos, algo que ele executa. Um atributo é uma característica de um objeto que o descreve. Eles são semelhantes a variáveis e armazenam dados específicos para cada objeto. Por exemplo, em uma classe "Cliente", os atributos podem incluir nome, telefone e cpf.





# Exemplo





# Método `__init__`

O método `__init__` em Python é um método especial que é chamado automaticamente quando uma nova instância de uma classe é criada. Ele é usado para inicializar os atributos de um objeto, ou seja, para atribuir valores aos atributos da classe.

```
class Livro:
    def __init__(self, titulo, autor, ano):
        self.titulo = titulo
        self.autor = autor
        self.ano = ano
```



# Método `__str__`

É um método especial, como `__init__`, usado para retornar uma representação de string de um objeto.

A principal finalidade do método `__str__` é fornecer uma representação legível em string do objeto, para que seja **mais fácil de entender e depurar o código**. Isso é especialmente útil ao trabalhar com classes personalizadas, onde você pode querer fornecer uma forma personalizada de exibir as informações do objeto.

```
def __str__(self):  
    return f"{self.titulo} ({self.autor}, {self.ano})"
```



# Implementando em Python: Classes

No exemplo de uso da classe, criamos dois objetos **pessoa1** e **pessoa2**, passando um nome e uma idade para cada um. Em seguida, chamamos o método `imprimir_informacoes` para cada objeto, que imprime as informações da pessoa correspondente.

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

    def imprimir_informacoes(self):
        print(f"Nome: {self.nome}, Idade: {self.idade}")

# Exemplo de uso da classe Pessoa
pessoa1 = Pessoa("João", 30)
pessoa1.imprimir_informacoes()

pessoa2 = Pessoa("Maria", 25)
pessoa2.imprimir_informacoes()
```



# Composição

Descreve a capacidade de construir classes complexas combinando objetos de outras classes como seus atributos. Em outras palavras, a composição permite que uma classe seja composta por outras classes como parte de sua estrutura interna, sem herança direta.

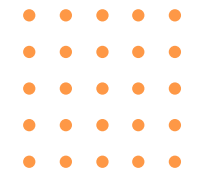
Um exemplo comum de composição é o relacionamento entre um carro e um motor. Um carro contém um motor como um de seus atributos, mas o motor pode existir separadamente do carro e ser compartilhado entre vários carros.



# Associação

Descreve o relacionamento entre dois objetos de classes diferentes. Esses objetos estão conectados de alguma forma, mas cada um mantém sua própria independência e ciclo de vida.

1. **Associação Simples:** um objeto de uma classe é associado a um objeto de outra classe. Por exemplo, uma pessoa tem um carro.
2. **Associação Agregação:** um objeto de uma classe é composto por objetos de outra classe, mas esses objetos podem existir independentemente do objeto principal. Por exemplo, uma escola tem vários alunos.
3. **Associação Composição:** um objeto de uma classe é composto por objetos de outra classe e a vida útil dos objetos internos é controlada pelo objeto principal. Por exemplo, um computador tem uma placa-mãe.



# Agregação

É um tipo de associação na programação orientada a objetos onde um objeto é composto por outros objetos, mas os objetos internos podem existir independentemente do objeto principal. Em outras palavras, a agregação representa uma relação de "tem-um" ou "tem-variados" entre duas classes, onde uma classe contém uma coleção de objetos de outra classe como parte de sua estrutura.

Na agregação, os objetos contidos podem ser compartilhados entre vários objetos principais e sua vida útil não é controlada pelo objeto principal. Isso significa que os objetos contidos podem ser criados, modificados ou destruídos independentemente do objeto principal.



# Encapsulamento

Encapsulamento é o conceito de esconder os detalhes de implementação de um objeto e expor apenas uma interface pública para interagir com ele.

Isso é geralmente alcançado usando modificadores de acesso, como público, protegido e privado, para controlar o acesso aos atributos e métodos de uma classe.

Os principais princípios do encapsulamento são:

1. Modificadores de Acesso
2. Métodos de Acesso
3. Esconder a Complexidade
4. Proteger a Integridade dos Dados





# Encapsulamento

## Modificadores de Acesso

Em Python, os modificadores de acesso são implementados usando convenções de nomenclatura e algumas funcionalidades da linguagem. A linguagem Python não possui modificadores de acesso como em outras linguagens orientadas a objetos (como Java ou C++), onde você pode especificar explicitamente se um membro de uma classe é público, protegido ou privado.

## Métodos de Acesso

Os métodos de acesso, também conhecidos como getters e setters, são métodos públicos de uma classe que permitem acessar e modificar os atributos privados de um objeto de forma controlada. Eles são usados para garantir o encapsulamento dos atributos de uma classe, permitindo que o acesso aos dados seja feito por meio de métodos, em vez de acessar diretamente os atributos.



# Herança

Permite a criação de uma nova classe baseada em uma classe existente. Essa nova classe herda atributos e métodos da classe existente, o que significa que ela pode reutilizar funcionalidades já implementadas e também adicionar novas funcionalidades ou modificar as existentes.

Os principais benefícios da herança incluem:

1. **Reutilização de código:** Permite reutilizar código já implementado em uma classe base, evitando duplicação e promovendo a modularidade e a manutenibilidade do código.
2. **Extensibilidade:** Permite adicionar novos comportamentos e atributos às classes derivadas, estendendo assim a funcionalidade da classe base.
3. **Polimorfismo:** Permite que objetos de diferentes classes derivadas respondam ao mesmo método de maneiras diferentes, proporcionando flexibilidade e facilidade de uso.



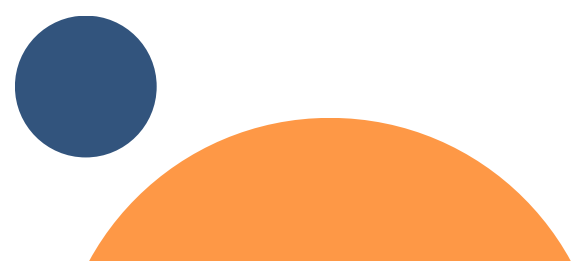
# Polimorfismo

É um conceito chave na programação orientada a objetos que se refere à capacidade de objetos de diferentes classes responderem ao mesmo método de maneiras diferentes. Em outras palavras, o polimorfismo permite que um único nome de método seja usado para realizar diferentes ações com base no contexto em que é chamado.

1. **Polimorfismo de Sobrescrita (Override):** Ocorre quando uma classe filha redefine (sobrescreve) um método de sua classe pai.
2. **Polimorfismo de Sobrecarga (Overload):** Ocorre quando uma classe possui vários métodos com o mesmo nome, mas com diferentes parâmetros.



# Revisão

- Por que utilizar Python?
    - Sintaxe simples e legível.
    - Vasta biblioteca padrão.
    - Comunidade ativa de desenvolvedores.
  - Classe e Objetos
    - Classes definem estrutura e comportamento.
    - Objetos são instâncias específicas de classes.
  - Classe Abstrata
    - Fornecem estrutura para definição de métodos a serem implementados por subclasses.
  - Métodos Especiais
    - `__init__` para inicialização de objetos.
    - `__str__` para representação de objetos como strings.
- 



# Revisão

- Composição
    - Relacionamento onde um objeto é composto por outros objetos.
    - Objetos internos são essenciais para o objeto principal.
  - Associação
    - Relacionamento entre objetos de classes diferentes.
    - Permite interação entre objetos.
  - Agregação
    - Relacionamento onde um objeto é composto por outros objetos.
    - Objetos internos podem existir independentemente do objeto principal.
  - Polimorfismo
    - Trata objetos de diferentes classes de forma uniforme.
    - Simplifica a escrita de código e aumenta a reutilização.
- 