Aula 05 - POO em Python: Herança e Polimorfismo

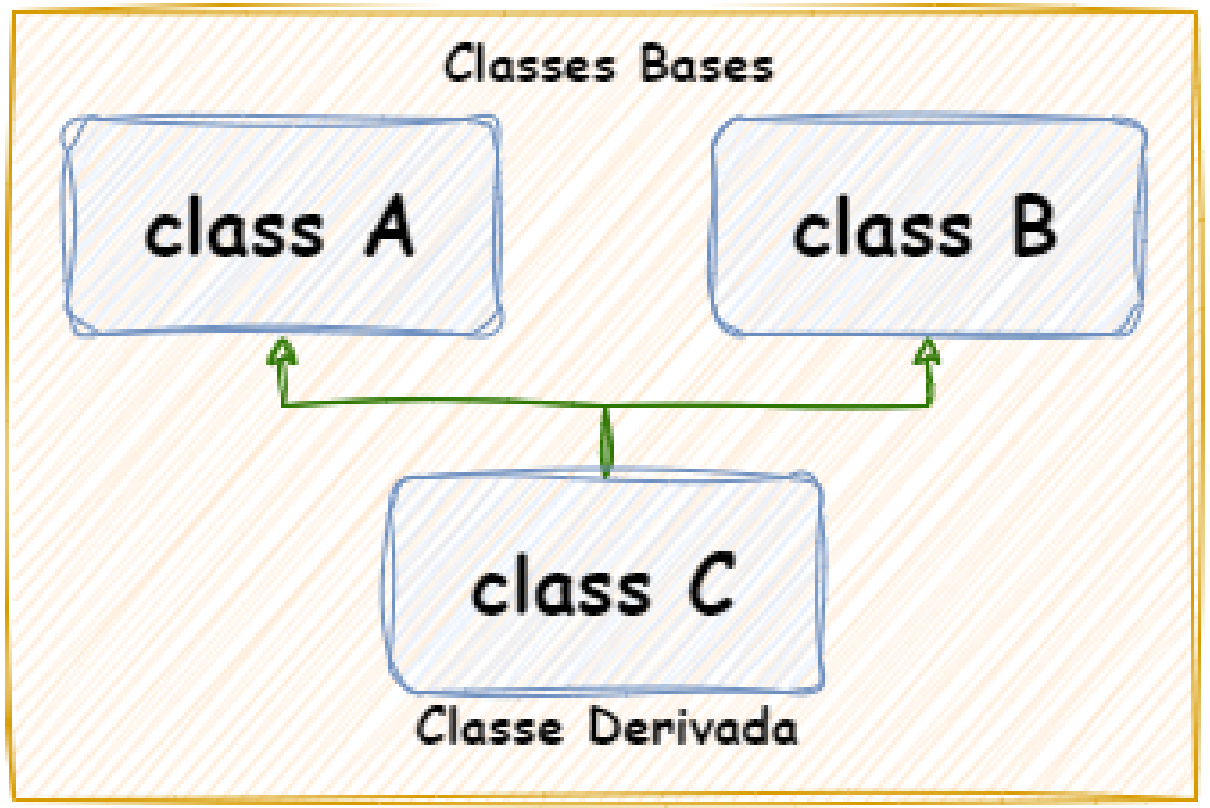
Herança em Python

O que será trabalhado nessa aula:

- Tipos de Herança - Continuação
- Substituição de métodos
- Polimorfismo

Tipos de Herança: Multipla

Em herança múltipla, uma classe filha pode herdar de várias classes pai.



Exemplo Real

```python
# Classe Pai 1
class Person:
    def person_info(self, name, age):
        print('Executando dentro da classe Person')
        print('Nome:', name, 'Idade:', age)

# Classe Pai 2
class Company:
    def company_info(self, company_name, location):
        print('Executando dentro da classe Company')
        print('Nome:', company_name, 'Local:', location)

# Classe Filha
class Employee(Person, Company):
    def Employee_info(self, salary, skill):
        print('Executando dentro da classe Employee')
        print('Salário:', salary, 'Skill:', skill)
```

```
objeto_empregado = Employee()

# Acessando Dados
objeto_empregado.person_info('Miguel Antônio', 42)
objeto_empregado.company_info('Google', 'Belo Horizonte')
objeto_empregado.Employee_info(12000, 'Machine Learning')
```

Saída

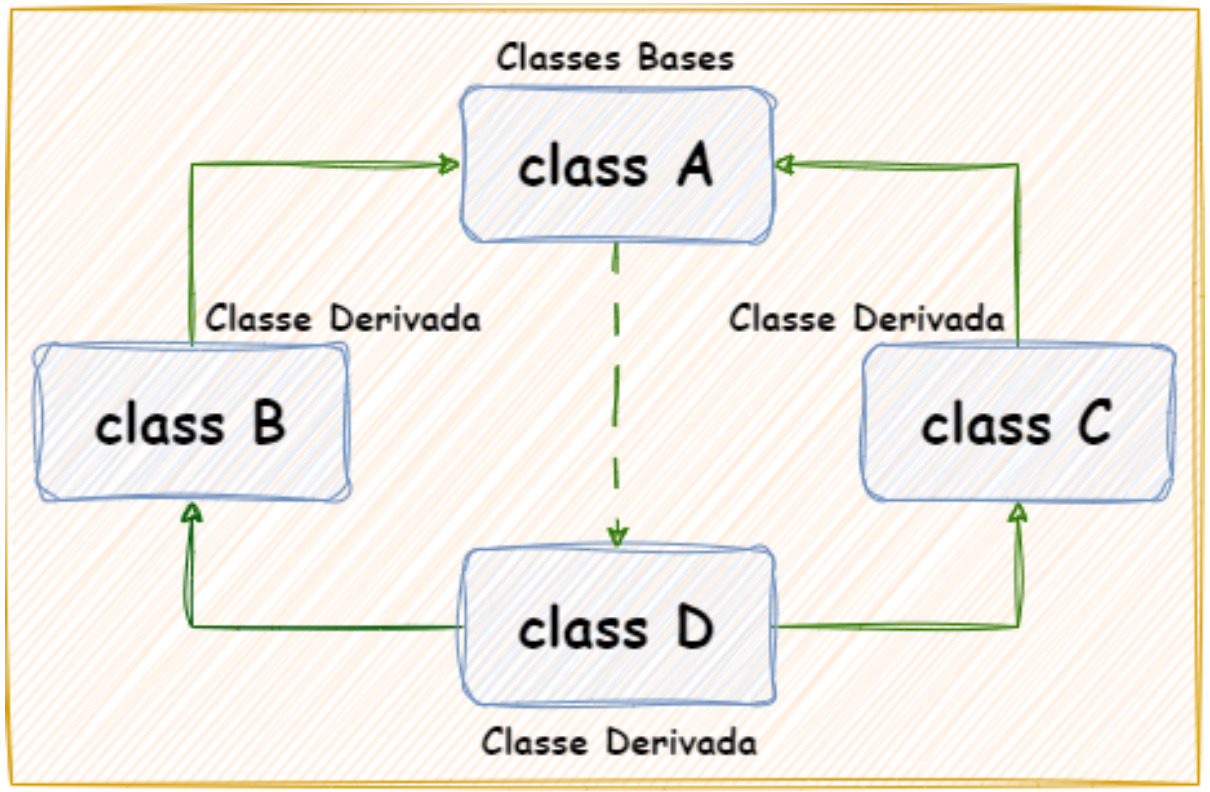
```
Executando dentro da classe Person
Nome: Miguel Antônio Idade: 42

Executando dentro da classe Company
Nome: Google Local: Belo Horizonte

Executando dentro da classe Employee
Salário: 12000 Skill: Machine Learning
```

Tipos de Herança: Híbrida

Quando a herança consiste em vários tipos ou uma combinação de diferentes heranças, é chamada de herança híbrida.



Exemplo Real

```
class Vehicle:
    def vehicle_info(self):
        print('Dentro da classe "Vehicle!"')

class Car(Vehicle):
    def car_info(self):
        print('Dentro da classe "Car"')

class Truck(Vehicle):
    def truck_info(self):
        print('Dentro da classe "Truck"')

# Sports Car can inherits properties of Vehicle and Car
class SportsCar(Car, Vehicle):
    def sports_car_info(self):
        print('Dentro da classe "SportsCar"')
```

```
s_carro = SportsCar()

s_carro.vehicle_info()
s_carro.car_info()
s_carro.sports_car_info()
```

Saída

```
Dentro da classe "Vehicle!"
Dentro da classe "Car"
Dentro da classe "SportsCar"
```

Nesse exemplo, existe herança ***hierárquica*** e ***múltipla***. Aqui criamos uma classe pai `Vehiclee` duas classes filhas nomeadas `Car` e `Truck` isso é ***herança hierárquica***.

Outra é `SportsCar` herdar de duas classes pai chamadas `Car` e `Vehicle`. Esta é uma ***herança múltipla***.

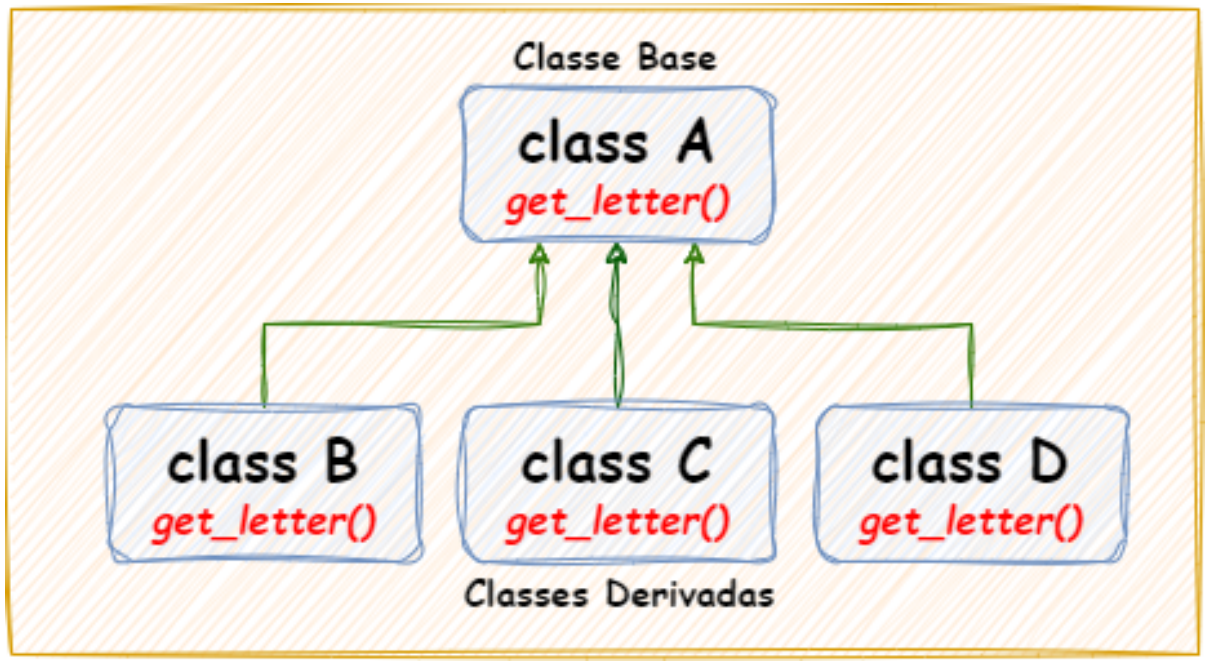
Override (Substituição) de Método

Na herança, todos os membros disponíveis na ***classe pai*** estão, por padrão, estão também disponíveis na ***classe filho***.

Se a ***classe filho*** não se satisfizer com a implementação da ***classe pai***, então a ***classe filho*** tem permissão para ***redefinir*** esse método estendendo ***funções adicionais*** na classe filho. Esse conceito é chamado de ***Override (substituição) de método***.

Override (Substituição) de Método

Quando um método de ***classe filha*** tem o mesmo ***nome***, os mesmos ***parâmetros*** e o mesmo ***tipo de retorno*** de um método em sua ***superclasse***, diz-se que o ***método na classe filha*** substitui o ***método na classe pai***.



Exemplo Real

```
class Vehicle:
    def max_speed(self):
        print("Velocidade maxíma 100 Km/Hora")

class Car(Vehicle):
    # overridden da implementção da classe Vehicle
    def max_speed(self):
        print("Velocidade maxíma 200 Km/Hora")

carro = Car()
carro.max_speed()
```

Saída

```
Velocidade maxíma 200 Km/Hora
```

Overloading (Sobrecarga) de Método

O processo de chamar o mesmo ***método*** com ***parâmetros diferentes*** é conhecido como ***sobrecarga de método***.

Python ***não suporta sobrecarga de método***. Python considera apenas o ***último método definido***, mesmo se você sobrecarregar o método.

Exemplo Real

```
class Vehicle:
    def vehicle_info(self):
        print('Informações da classe "Vehicle!"')

    def vehicle_info(self, marca_veiculo):
        print(f'Informações do carro {marca_veiculo}')
```

Exemplo Real

```
s_carro = Vehicle()
s_carro.vehicle_info()
```

Saída

```
TypeError: Vehicle.vehicle_info() missing 1 required positional argument: 'marca_veiculo'
```

Exemplo Real

```
s_carro = Vehicle()
s_carro.vehicle_info('Ford')
```

Saída

```
Informações do carro Ford
```

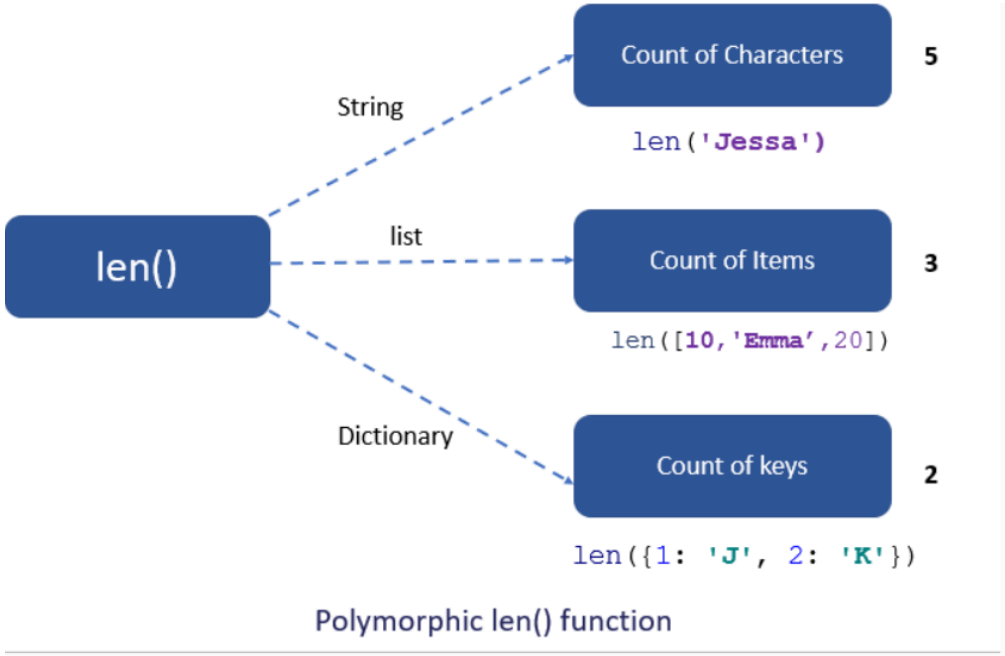
Polimorfismo

Polimorfismo em Python é a habilidade de um objeto assumir muitas formas. Em palavras simples, polimorfismo nos permite executar a mesma ação de muitas maneiras diferentes.

No polimorfismo, um método pode processar objetos de forma diferente dependendo do tipo de classe ou tipo de dado.

Polimorfismo

A função interna `len()` calcula o comprimento de um objeto dependendo do seu tipo. Se um objeto for uma `string`, ele retorna a contagem de caracteres, e se um objeto for um `list`, ele retorna a contagem de itens em uma lista.



Polymorphic len() function