Aula 04 - POO em Python: Herança

Introdução ao tema

Herança é o mecanismo em POO pelo qual uma classe pode herdar os recursos (atributos e métodos) de outra classe.

Herança significa criar novas classes com base nas existentes.

Introdução ao tema

Uma classe que herda de outra classe pode reutilizar os métodos e campos dessa classe. Além disso, você também pode adicionar novos campos e métodos à sua classe atual.

O que será trabalhado nessa aula:

- Por que usar herança?
- Tipos de Herança
- Função `Super()`

Herança em Python

Por que usar herança em Python?

- **Reutilização de Código**: O código escrito na Superclasse é comum a todas as subclasses. As classes filhas podem usar diretamente o código da classe pai.
- **Substituição de método**: A substituição de método só pode ser btida por meio de herança. É uma das maneiras pelas quais Java atinge o polimorfismo em tempo de execução.
- **Abstração**: O conceito de abstrato onde não temos que fornecer todos os detalhes é alcançado por meio de herança. ***A abstração*** apenas mostra a funcionalidade para o usuário.

Herança em Python

A classe existente é chamada de ***classe base*** ou ***classe pai*** e a nova classe é chamada de ***subclasse*** ou ***classe filha*** ou ***classe derivada***.

Sintaxe

```
class BaseClass:
  Body of base class
class DerivedClass(BaseClass):
  Body of derived class
```

Herança em Python

Na herança, a ***classe filha*** adquire todos os `membros de dados` e `funções` da ***classe pai***. Além disso, uma ***classe filha*** também pode fornecer sua implementação específica para os ***métodos da classe pai***.

Exemplo Real

```
# Classe Pai/Base
class Vehicle:
    def Vehicle_info(self):
        print('Dentro da classe Pai (Vehicle)')

# Classe Filha
class Car(Vehicle):
    def car_info(self):
        print('Dentro da classe Filha (Car)')
```

Herança em Python

Exemplo Real

```
# Criando objeto de Car
carro = Car()

# Acessando informações de Veículo usando o objeto carro
carro.Vehicle_info()
carro.car_info()
```

Saída

```
Dentro da classe Pai (Vehicle)
Dentro da classe Filha (Car)
```

Herança em Python

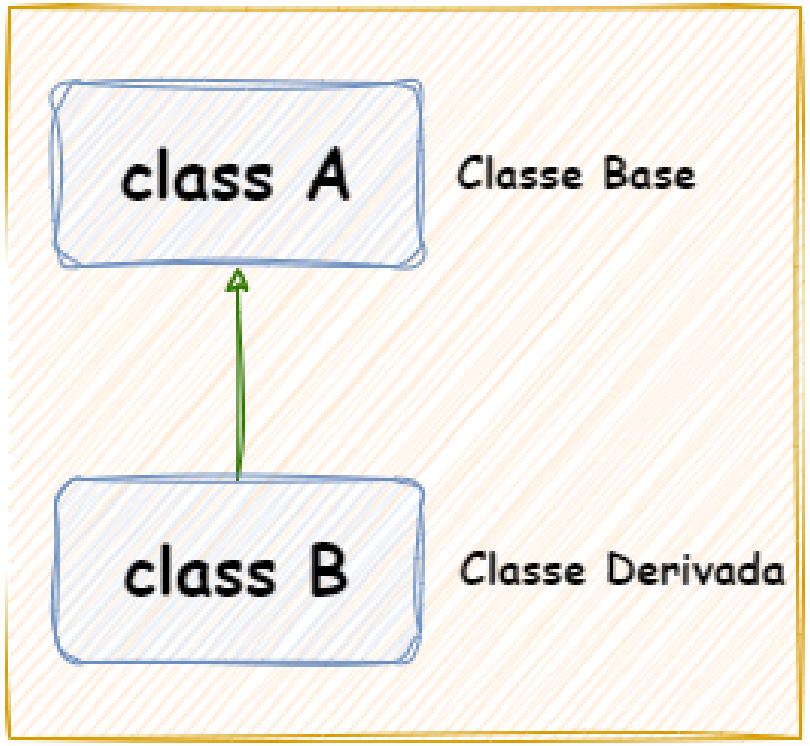
Observe que durante a herança apenas o objeto da subclasse é criado, não a superclasse.

No programa anterior, quando um objeto da classe `Car()` é criado, uma cópia de todos os métodos e campos da ***classe pai*** adquirem um espaço de memória neste objeto.

É por isso que, usando o objeto da ***classe filha***, também podemos acessar os membros de uma ***classe pai***.

Tipos de Herança: Única

Na herança simples/única, uma classe filha herda de uma classe pai simples. Aqui está uma classe filha e uma classe pai.



Exemplo Real

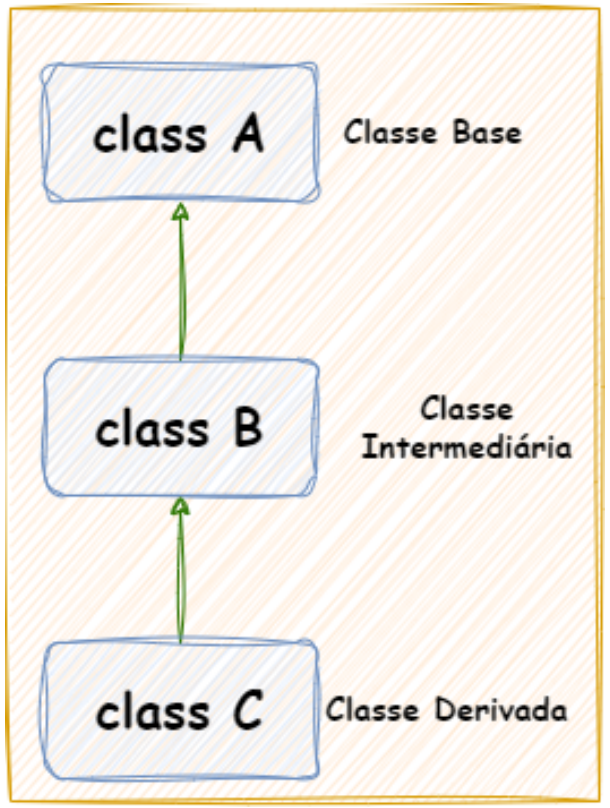
```
class Employee:
    salary = 60000

    def show_salary(self):
        print(f'Salário mensal é de {self.salary}')

class Engineer(Employee):
    def __init__(self):
        self.bonus = 10000

    def show_bonus(self):
        print(f'Bonus é de {self.bonus}')

empregado = Engineer()

empregado.show_salary()
empregado.show_bonus()
```

Saída

```
Salário mensal é de 60000
Bonus é de 10000
```

Tipos de Herança: Multinível

Na ***herança multinível***, uma classe ***derivada*** herdará uma classe ***base*** e, assim como a classe ***derivada***, também atuará como classe ***base*** para outras classes.



Exemplo Real

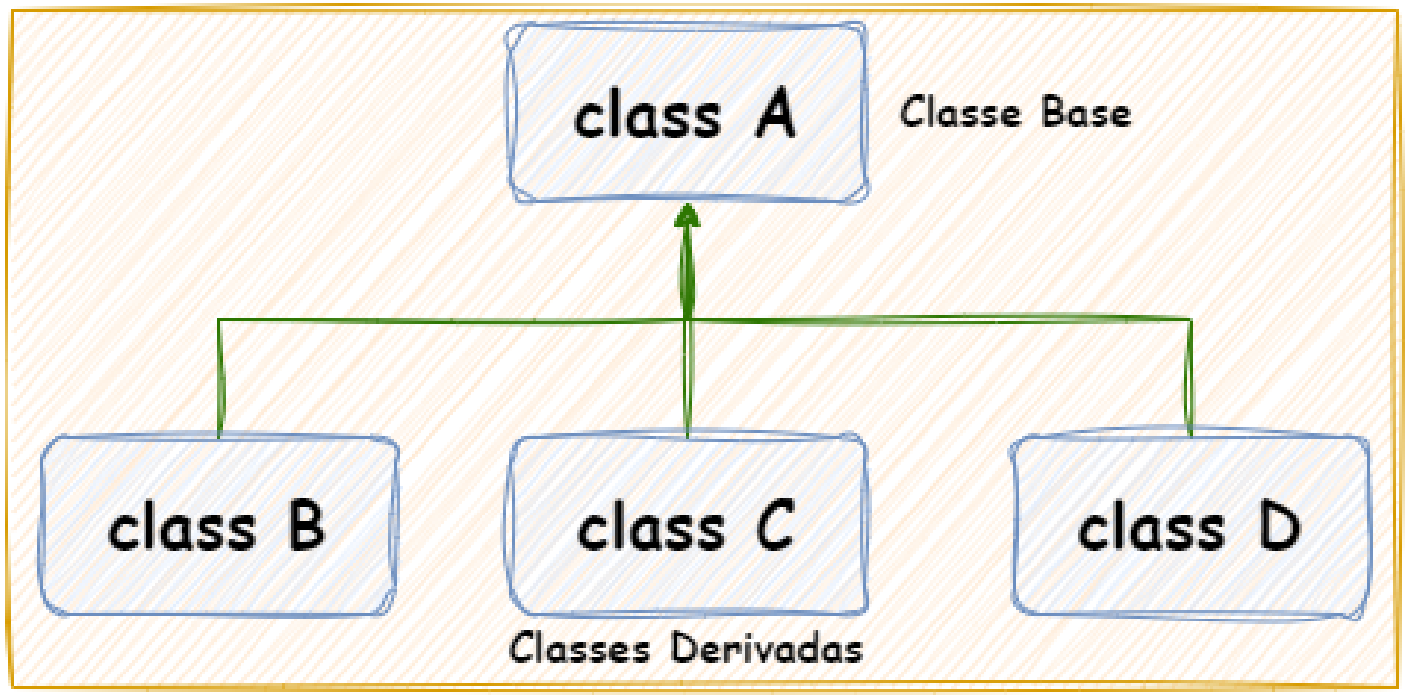
```
class Vehicle:
    def Vehicle_info(self):
        print('Dentro da classe Vehicle')

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

class SportsCar(Car):
    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
```

Tipos de Herança: Hierárquica

Na herança hierárquica, mais de uma classe filha é derivada de uma única classe pai. Em outras palavras, podemos dizer uma classe pai e várias classes filhas.



Exemplo Real

```
class Vehicle:
    def info(self):
        print("Esté é um veículo!")

class Car(Vehicle):
    def car_info(self, name):
        print("A marca do carro é ", name)

class Truck(Vehicle):
    def truck_info(self, name):
        print("O caminhão é um ", name)

objeto_carro = Car()
objeto_carro.info()
objeto_carro.car_info('BMW')

objeto_caminhao = Truck()
objeto_caminhao.info()
objeto_caminhao.truck_info('Ford')
```

Saída

```
Esté é um veículo!
A marca do carro é BMW

Esté é um veículo!
O caminhão é um Ford
```

Método Super()

A função `super()` do Python nos permite referenciar a superclasse implicitamente.

Ao referenciar a superclasse (classe pai) da subclasse (classe filha), não precisamos escrever o nome da superclasse explicitamente.

Exemplo Real

```
class Mammal:
  def __init__(self, mammalName):
    print(mammalName, ' é um animal de sangue quente.')

class Dog(Mammal):
  def __init__(self):
    print('O cachorro tem 4 patas.')
    super().__init__('Cachorro')
```

Exemplo Real

```
obj_cao = Dog()

obj_mamifero = Mammal('Gato')
```

Saída

```
O cachorro tem 4 patas.
Cachorro é um animal de sangue quente.

Gato é um animal de sangue quente.
```

No `obj_cao`, é chamado o construtor `__init__()` da classe `Mammal`, atráves da classe `Dog`, usando o código `super().__init__('Dog)`

No `obj_mamifero`, o construtor `__init__()` da classe `Mammal` é chamado diretamente.