

# Estrutura de dados

## Orientação a Objetos

### Classes Abstratas

Prof<sup>a</sup>. Dr<sup>a</sup>. Alana Morais

# Finalizaram a lista?

Praticar faz a diferença ...

# Herança em OO

## Conceito:

- Herança é a capacidade de especializar tipos de objetos (classes), de forma que os tipos especializados contenham, além de características estruturais e comportamentais já definidas pelos seus “ancestrais”, outras definidas para eles próprios.

## Forma:



# Herança em OO

A novidade de hoje é:

Existe uma hierarquia entre as classes que se relacionam por Herança

A superclasse pode definir comportamentos (como um protocolo de funcionamento)

Como assim?

# Classes Abstratas

- Queremos que uma subclasse de alguma classe qualquer seja obrigada a implementar uma estrutura de métodos.
- Por exemplo, se tivermos que criar uma classe que é uma lista mutável, queremos garantir que todas as suas filhas sejam obrigadas a implementar o `metodo1()`, pois caso contrário não será uma lista mutável.

# Classes Abstratas

- Conceito relacionado aos princípios de HERANÇA
- Servem como “modelo” para outras classes que dela herdem.
- Deve ser formada por pelo menos um método abstrato.
- método abstrato: é uma assinatura de método

```
@abstractmethod
```

```
def metodoAbstratoExemplo(self):
```

```
    pass
```

# Classes Abstratas

- Conceito relacionado aos princípios de HERANÇA
- Servem como “modelo” para outras classes que dela herdem.
- Deve ser formada por **pelo menos um método abstrato**.
- Método abstrato: é uma assinatura de método

**@abstractmethod**

```
def metodoAbstratoExemplo(self):  
    pass
```

MÉTODO VAZIO

# Classes Abstratas

- Mas só isso ainda não é suficiente, é preciso alguma estrutura específica do Python, pois trata-se de uma linguagem dinâmica.
- Existem algumas classes abstratas denominadas ABC, acrônimo para Abstract Base Classes.

```
from abc import ABC, abstractmethod
```

```
class Teste(ABC):
```

```
...
```

```
@abstractmethod
```

```
def metodoAbstratoExemplo(self):
```

```
    pass
```



# Classes Abstratas

- Mas só isso ainda não é suficiente, é preciso alguma estrutura específica do Python, pois trata-se de uma linguagem dinâmica.
- Existem algumas classes abstratas denominadas ABC, acrônimo para Abstract Base Classes.

```
from abc import ABC, abstractmethod
```

```
class Teste(ABC):
```

```
...
```

```
@abstractmethod
```

```
def metodoAbstratoExemplo(self):
```

```
    pass
```

# Exemplo

Implemente uma estrutura de classes que representam tipos de Programas em stream. De um lado temos Filmes (com nome, ano, duração e numeroLikes) e do outro temos as Séries ( com nome, ano, temporadas, numeroLikes).

Utilize herança, organize o projeto em pacotes e teste seu projeto.

# Exemplo

Agora torne a classe Programa uma classe Abstrata.

Implemente o método `exibirInformacoesExclusivas( )`

# Vamos fazer uns testes ...

Tente instanciar um objeto da classe Programa.

O que aconteceu?



# Classes Abstratas em Python

No Python, não é muito aconselhável criar nossas próprias classes abstratas, por ser um aspecto mais avançado da linguagem.

O Python recomenda que, caso queiramos criar uma classe que dependa de outra, precisamos checar se já não existe uma destas classes base prontas para uso.

<https://www.python.org/dev/peps/pep-0544/#protocol-members>

## Exemplo - voltando para nosso exemplo

Se importarmos um pacote **collections.abc**, por exemplo, que contém diversas classes que podem ser utilizadas, inclusive o **MutableSequence**, uma sequência mutável cujos valores podem ser alterados.

Considere a classe Playlist. Se queremos herdar de **MutableSequence**, que não possui apenas métodos abstratos, mas também comportamentos, os quais queremos absorver, teremos algo como:

```
__delitem__, __getitem__, __len__, __setitem__, insert
```

# Vantagens

Verificamos que é possível usar classes abstratas do sistema e obter alguns benefícios, por exemplo:

- Tenho um erro que me diz, em tempo de instanciação, se eu esqueci de implementar algum método da superclasse
- E também sou impedido de instanciar um objeto do tipo da superclasse, pra não ter problema com os métodos abstratos.
- Ainda posso aproveitar código dos meus métodos abstratos (que podem ter implementação na classe mãe)

# Duck Typing





# Exercício Casa

Implemente uma classe abstrata `Conta Bancaria` que contém como atributos o número da conta e o saldo, e como métodos abstratos `sacar` e `depositar` que recebem um parâmetro do tipo `double`.

Crie as classes `Conta Corrente` e `Conta Poupança` que herdam da `Conta Bancaria`. A primeira possui um atributo `taxaDeOperação` que é descontado sempre que um saque e um depósito são feitos. A segunda possui um atributo `limite` que dá crédito a mais para o correntista caso ele precise sacar mais que o saldo. Neste caso, o saldo pode ficar negativo desde que não ultrapasse o limite. Contudo isso não pode acontecer na classe `Conta Corrente`.

Teste essa estrutura.

Dúvidas?