08

Mãos na massa

Chegou a hora de você pôr em prática o que foi visto na aula. Para isso, execute os passos listados abaixo.

1) Comece a abstrair mais a ideia da playlist. Não dá pra colocar em uma `list` e somente depois dar nome, por exemplo. Você precisa criar uma classe pra representar as diferentes playlists.

Então, crie a classe logo abaixo da classe `Serie` :

```
class Playlist():
    def __init__(self, nome, programas):
        self.nome = nome
        self.programas = programas

    def tamanho(self):
        return len(self.programas)
`
```

Com esta classe, você consegue representar melhor a playlist.

2) Mude o código de impressão e verifique se vai dar certo:

```
vingadores = Filme('vingadores - guerra infinita', 2018, 160)
atlanta = Serie('atlanta', 2018, 2)
tmep = Filme('todo mundo em panico', 1999, 100)
demolidor = Serie('demolidor', 2016, 2)

vingadores.dar_likes()
vingadores.dar_likes()
vingadores.dar_likes()
atlanta.dar_likes()
atlanta.dar_likes()
tmep.dar_likes()
tmep.dar_likes()
demolidor.dar_likes()
demolidor.dar_likes()

listinha = [atlanta, vingadores, demolidor, tmep]
minha_playlist = Playlist('fim de semana', listinha)

for programa in minha_playlist:
    print(programa)
```

Com essa alteração, o código gerou um erro, sinalizou que a playlist não é um *iterable*, um iterável.

3) Realmente não é. Por agora, corrija corrigir isto apenas pegando a lista interna da `Playlist` :

```
for programa in minha_playlist.programas:
    print(programa)
```

Funcionou! Só que desse jeito, você está acessando uma informação interna da `Playlist`. E se você fizer uma herança da classe `list`?

4) Tente isso para facilitar a iteração. Se você definir que `Playlist` herda de `list`, pode dizer que `Playlist` é um `list`, e como consequência disso, já que list é `Iterable`, `Playlist` também será.

O código fica assim:

```
class Playlist(list):
    def __init__(self, nome, programas):
        self.nome = nome
        super().__init__(programas)
```

5) E a parte da impressão muda também, para percorrer diretamente a `Playist`:

```
for programa in minha_playlist:
    print(programa)
```

6) Perceba que quando você herda de `list`, você reutiliza o `init` da superclasse, que pode receber os itens em forma de `list`. Outro detalhe importante é que não é necessário mais o método `tamanho`, já que `list` suporta o `len` do Python.

Se você quiser verificar o tamanho, pode fazer o seguinte:

```
print(f'Tamanho: {len(minha_playlist)}')
```

7) Com a herança do `list`, como foi falado na aula, você começa a usar muitas funcionalidades desta classe que já estavam prontas. Isso é muito bom, pois você precisa *reinventar a roda*, porém tem alguns problemas nessa abordagem, como acoplamento e aumento da complexidade da sua classe.

Para tentar diminuir esta complexidade, dê alguns passos para trás nesse momento, para tentar aproveitar apenas as vantagens. Deixe a classe `Playlist` assim:

```
class Playlist():
    def __init__(self, nome, programas):
        self.nome = nome
        self._programas = programas

    @property
    def listagem(self):
        return self._programas

    @property
    def tamanho(self):
        return len(self._programas)
```

8) E o código de impressão deste jeito:

```
for programa in minha_playlist.listagem:
    print(programa)
```

```
print(f'Tamanho: {len(minha_playlist.listagem)}')
```

Essas alterações removeram a complexidade da herança, e encapsula a listagem pra ficar claro pro usuário que existe uma propriedade representando os programas listados e também o tamanho (já que você perdeu a vantagem de herdar `list`).

Deu uma sensação que o código andou para trás agora, que você perdeu algumas vantagens de usar a herança. Mas na próxima aula, você dará um jeito *pythonico* nisso!