Armazenando e Acessando: Estruturas de Dados em Ação

Objetivos da aula

- Armazenamento e acesso de objetos Filme em estruturas de dados
- Nesta aula vamos organizar os objetos Filme em listas, dicionários e conjuntos para análises eficientes. Ao final da aula você será capaz de:
  - Usar **listas** para armazenamento principal de filmes
  - Agrupar filmes por gênero usando **dicionários**
  - Criar conjuntos (**set**) para categorias únicas
  - Filtrar, buscar e agregar dados de filmes
  - Aplicar **TDD** para validar operações de acesso e manipulação

Conceito de Lista

- Uma lista é uma coleção ordenada de elementos que permite adicionar, acessar e iterar sobre itens de forma sequencial.

```
filmes  =  [
    Filme("Matrix", 63000000, 465000000, "Ficção"),
    Filme("Titanic", 200000000, 2200000000, "Drama")
]
for filme in filmes:
    print(filme.titulo, filme.genero)
```

- No exemplo acima, percorremos todos os filmes, mostrando títulos e gêneros.

Conceito de Dicionário

- Um dicionário é uma coleção de pares chave-valor, onde cada chave é única e mapeia para um valor.
- Permite acesso rápido aos valores por meio de suas chaves.

```
# Criando filmes
filme1 = Filme("Matrix", 63000000, 465000000, "Ficção")
filme2 = Filme("Avatar", 237000000, 2847000000, "Ficção")
filme3 = Filme("Titanic", 200000000, 2200000000, "Drama")

# Criando dicionário por gênero
dicionario_genero  =  {
    "Ficção": [filme1, filme2],
    "Drama": [filme3]
}
```

Conceito de Conjunto

- Um conjunto é uma coleção não ordenada e sem elementos duplicados. Ideal para armazenar valores únicos, como categorias ou tags.
- A operação de verificação de pertencimento é rápida.

```
generos = {"Ficção", "Drama", "Ação"}
```

Lista vs Dicionário vs Conjunto

- Lista: ordenada, permite duplicatas, acessível por índice
- Dicionário: coleção de pares chave-valor, acesso rápido por chave única
- Conjunto (set): não ordenado, elementos únicos, ideal para categorias sem repetição

Filtrando filmes por gênero

- Cria uma sublista contendo apenas filmes de ficção.

```
ficcao  =  [f for f in filmes if f.genero == "Ficção"]
```

Criando dicionário por gênero

- Agrupa filmes por gênero em listas dentro de um dicionário.

```
from filme import Filme

lista_filmes  =  [
    Filme("Matrix", 63000000, 465000000, "Ficção"),
    Filme("Titanic", 200000000, 2200000000, "Drama")
]

dicionario_genero = {}
for f in lista_filmes:
    if f.genero not in dicionario_genero:
        dicionario_genero[f.genero] = []
        dicionario_genero[f.genero].append(f)
```

Por que criar lista vazia no dicionário?

- dicionario_genero[f.genero] = [] # Cria lista vazia para novos gêneros
  - Quando encontramos um gênero pela primeira vez, ele ainda não existe como chave no dicionário.
  - Precisamos criar uma lista vazia para armazenar os filmes desse gênero.
  - Depois disso, podemos adicionar filmes à lista usando .append(f).
  - Sem essa linha, tentar adicionar diretamente causaria um KeyError.

Teste de dicionário por gênero

- Valida se o agrupamento por gênero está correto.

```python
def test_dicionario_por_genero():
    assert "Ficção" in dicionario_genero
    assert all(isinstance(f, Filme) for f in dicionario_genero["Ficção"])
```

Criando conjuntos de gêneros

- Cria um conjunto com gêneros únicos presentes na lista.

```
generos = set(f.genero for f in filmes)
```

Teste de conjunto de gêneros

- Garante que os gêneros únicos foram capturados.

```
def test_conjunto_generos():
    assert isinstance(generos, set)
    assert "Drama" in generos
```

Calculando receita e orçamento

- A função sum agrega todos os valores numéricos de uma coleção, retornando a soma total dos elementos.

```python
# Soma a receita de todos os filmes na lista
total_receita = sum(f.receita for f in filmes)

# Calcula a média de orçamento dos filmes.
media_orcamento = sum(f.orcamento for f in filmes) / len(filmes)
```

Buscando filme por título

- Procura o filme "Matrix" usando expressão geradora.
- A função next retorna o **próximo item de um iterador**. Quando usada com expressões geradoras, permite acessar o **primeiro elemento que satisfaz uma condição**, ou um valor padrão caso não exista.

```
matrix = next((f for f in filmes if f.titulo == "Matrix"), None)
```

Filtragem por lucro

- Cria sublista com filmes que tiveram lucro acima de 100 milhões.

```
filmes_lucrativos  =  [f for f in filmes if f.lucro() > 100000000]
```

Agrupando filmes

- Organiza filmes em categorias de orçamento.

```
orcamento_grupos  =  {"baixo": [], "medio": [], "alto": []}
for f in filmes:
    if f.orcamento < 50000000:
        orcamento_grupos["baixo"].append(f)
    elif f.orcamento < 200000000:
        orcamento_grupos["medio"].append(f)
    else:
        orcamento_grupos["alto"].append(f)
```

Criando um set de títulos

- Captura todos os títulos sem repetição.

```
titulos_unicos = set(f.titulo for f in filmes)
```

Contagem de filmes por gênero

- Calcula quantos filmes existem em cada gênero.

```
contagem_genero = { g: len(dicionario_genero[g]) for g in dicionario_genero}
```

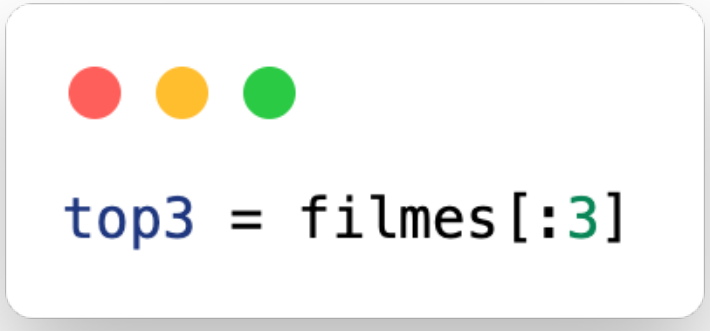
Ordenando filmes por receita

- Organiza a lista do filme mais lucrativo para o menos lucrativo.
- O parâmetro reverse=True em funções de ordenação (sort ou sorted) ordena os elementos em ordem decrescente em vez da ordem crescente padrão.

```
filmes.sort(key=lambda f: f.receita, reverse=True)
```

Pegando top 3 filmes

- Seleciona os três primeiros filmes da lista já ordenada.



Set: União e Interseção

- Sets permitem operações matemáticas como união (todos os elementos) e interseção (elementos comuns).

```
generos_filmes1  = {"Ficção", "Drama", "Ação"}
generos_filmes2  = {"Drama", "Comédia", "Ação"}

# União
todos_generos = generos_filmes1 | generos_filmes2
# Interseção
generos_comuns = generos_filmes1 & generos_filmes2
```

Set: Diferença

- A operação de diferença retorna elementos presentes em um set e ausentes no outro.

```
generos_exclusivos =  generos_filmes1 - generos_filmes2
```

Dicionário: Atualizando valores

- Dicionários permitem atualizar valores de chaves existentes e criar novas chaves dinamicamente.

```
contagem_genero =  "Ficção": 2, "Drama": 1}
contagem_genero{"Drama"] += 1  # Adiciona mais um filme de Drama
contagem_genero["Comédia"] = 1  # Cria nova chave
```

Dicionário: Iterando sobre chave/valor

- .items() permite iterar simultaneamente sobre chaves e valores de um dicionário.

```
for genero, lista_filmes in dicionario_genero.items():
    print(f"Gênero: {genero}, Quantidade: {len(lista_filmes)}")
```

Dicionário: Obtendo valor com get

- get retorna o valor da chave se existir, ou um valor padrão caso a chave não esteja presente, evitando erros.

```
num_filmes_comedia = contagem_genero.get("Comédia", 0)
print(num_filmes_comedia)
```

Lista: Compreensão com condição

- Listas suportam compreensões, permitindo criar novas listas aplicando filtros ou transformações de forma concisa.

```
filmes_lucro_alto =  [f for f in filmes if f.lucro() > 500000000]
```

Lista: Ordenação personalizada

- sorted cria uma nova lista ordenada com base em qualquer critério, sem modificar a lista original. Aqui, ordenamos pelo orçamento dos filmes.

```
filmes_ordenados_orcamento = sorted(filmes, key=lambda f: f.orcamento)
```

Lista: Ordenação avançada

- É possível usar tuplas como chave de ordenação para combinar múltiplos critérios.
  - Primeiro ordena por gênero (alfabético)
  - Depois, dentro de cada gênero, ordena pela receita em ordem decrescente (-f.receita).

```python
# Ordena primeiro por gênero e depois por receita decrescente
filmes_ordenados = sorted(filmes, key=lambda f: (f.genero, -f.receita))
```

Encerramento

- Você aprendeu:
  - Armazenar objetos Filme em listas, dicionários e conjuntos
  - Filtrar, buscar e agregar informações
  - Agrupar filmes por gênero ou faixa de orçamento
  - Aplicar TDD para validar as operações
  - Preparar os dados para análises e visualizações futuras

PUC Minas