Examen final 2021-02-01

95.14/75.40 - Algoritmos y Programación I - Curso Essaya

Objetivo

Implementar la clase `IMDB`, una base de datos de cine.

Se dispone de los siguientes archivos:

- `imdb.py`: implementación (inicialmente vacía) de la clase `IMDB`
- `pruebas.py`: Pruebas automáticas

El archivo `pruebas.py` efectúa una serie de pruebas para verificar el correcto funcionamiento de la clase `IMDB`.

La idea es agregar en `imdb.py` todo el código necesario para que las pruebas automáticas pasen. Además de las funciones que se piden en el enunciado se permite crear cualquier cantidad de funciones y clases internas.

Hay 5 ejercicios. Es condición necesaria (pero no suficiente) para aprobar el examen que haya 3 ejercicios OK.

Salida del programa

Al ejecutar el programa (`python3 pruebas.py`), se ejecutan todas las pruebas, y se imprime el resultado de cada ejercicio (OK o FAIL), junto con la cantidad de ejercicios OK. Ejemplo:

```
$ python3 pruebas.py
ejercicio_1: OK
ejercicio_2: OK
ejercicio_3: OK

Traceback (most recent call last):
File "pruebas.py", line 237, in main
    ejercicio()
File "pruebas.py", line 148, in ejercicio_4
    assert ok
AssertionError

ejercicio_4: FAIL
ejercicio_5: OK
Cantidad de ejercicios OK: 4
```

Recordar: es condición necesaria[1] (pero no suficiente[2]) para aprobar el examen que haya al menos 3 ejercicios OK.

[1] Es posible que un ejercicio sea considerado ”bien” aun cuando la prueba informa ”FAIL”; por ejemplo si hay un error trivial en el código que se arreglaría haciendo un pequeño cambio.

[2] Es posible (pero poco probable) que un ejercicio sea considerado ”mal” aun cuando la prueba informa ”OK”; por ejemplo si hay errores conceptuales.

La clase IMDB

La clase `IMDB` modela una base de datos de cine, y permite hacer consultas como búsqueda de actores y actrices, calificaciones de películas, etc.

La base de datos almacena información de **actores** y **films**:

- Un **actor** tiene:
  - Un **nombre**
  - Una **fecha de nacimiento** (año, mes, día)
  - Un listado de films en los que participó (puede ser vacío, un film no puede aparecer 2 veces en el listado)
- Un **film** tiene:
  - Un **nombre**
  - Una **fecha de lanzamiento** (año, mes, día)
  - Un listado de actores que participaron en el film (puede estar vacío, un actor no puede aparecer 2 veces en el listado)
  - Un listado de calificaciones recibidas, que son números entre 1 y 10. A partir de este listado se calculará un promedio.

IDs

Desde "afuera" de la implementación, tanto los actores como los films se identifican mediante un número único llamado ID. Es decir, todas las operaciones reciben y devuelven IDs. La elección de estos IDs al crear usaurios/tuits queda a cargo de la clase `IMDB`.

Los IDs de los actores deben ser únicos, y lo mismo para los films, pero no hay problema en que un actor y un film tengan el mismo ID.

Recomendación: usar números crecientes:

- Creamos el primer actor -> ID = 0
- Creamos el primer film -> ID = 0
- Creamos el segundo actor -> ID = 1
- Creamos el segundo film -> ID = 1
- Creamos el tercer film -> ID = 2

Recomendación: Notar que en la descripción de las entidades **actor** y **film**, se menciona que un actor tiene una lista de films, y al mismo tiempo un film tiene una lista de actores. Si lo tomamos en forma literal llegamos a una definición recursiva que es más complicada de lo necesario. En cambio, pensar que los actores y films pueden estar relacionados mediante listas de IDs (ej: un film puede tener una lista de IDs de actores).

Descripción de las pruebas

`ejercicio_1`: Funcionamiento básico

Prueba el funcionamiento básico de la clase `IMDB`. Sin esta prueba funcionando probablemente no se pueda pasar ninguna de las otras pruebas.

Métodos a implementar:

- `__init__()`: Crea una instancia de `IMDB` con 0 actores y 0 films.
- `actor_agregar(nombre, año, mes, dia)`: Recibe el nombre y la fecha de nacimiento de un actor o actriz, y lo agrega a la base de datos. Devuelve el ID del actor, que debe ser distinto a los IDs de los otros actores existentes.
- `cantidad_actores()`: Devuelve la cantidad de actores existentes en la base de datos.
- `actor_nombre(id_actor)`: recibe el ID de un actor y devuelve su nombre.

- `actor_nacimiento(id_actor)`: recibe el ID de un actor y devuelve su fecha de nacimiento (tupla `(año, mes, día)`).
- `film_agregar(nombre, año, mes, dia, ids_actores)`: Recibe el nombre, la fecha de lanzamiento y una lista de IDs de actores que participaron en un film, y agrega el film a la base de datos. Devuelve el ID del film, que debe ser distinto a los IDs de los otros films existentes.
- `cantidad_films()`: Devuelve la cantidad de films existentes en la base de datos.
- `film_nombre(id_film)`: recibe el ID de un film y devuelve su nombre.
- `film_lanzamiento(id_film)`: recibe el ID de un film y devuelve su fecha de lanzamiento (tupla `(año, mes, día)`).
- `film_actores(id_film)`: recibe el ID de un film y devuelve la lista de IDs de los actores que participaron en el mismo.
- `actor_films(id_actor)`: recibe el ID de un actor y devuelve la lista de IDs de los films en los que participó.

`ejercicio_2:` Archivos CSV

Prueba que podamos exportar la base de datos en formato CSV.

Funciones a implementar:

- `escribir_csv()`: Escribe tres archivos CSV, todos sin encabezado:
  - `actores.csv` con formato `nombre,año,mes,día`
  - `films.csv` con formato `nombre,año,mes,día`
  - `films_actores.csv` con formato `id_film,id_actor`. Cada una de las líneas de este archivo representa una relación "tal actor trabajó en tal film".

`ejercicio_3:` Agrupamiento de films por décadas

Funciones a implementar:

- `films_decadas()`: Devuelve un diccionario `{década: <lista de IDs de films>}`

La década de un año es el mayor múltiplo de 10 que es menor o igual al año. Ejemplos:

- `decada(1983) -> 1980`
- `decada(2015) -> 2010`
- `decada(2010) -> 2010`
- `decada(1999) -> 1990`

`ejercicio_4:` Calificaciones

Funciones a implementar:

- `calificar(id_film, calificación)`: Agrega la calificación (número entre 1 y 10) al film dado.
- `film_promedio(id_film)`: Devuelve la calificación promedio del film. En caso de que el film no haya recibido ninguna calificación, devuelve 0.
- `films_top10()` Devuelve la lista de los IDs de los 10 films con mejor promedio, ordenada de mayor a menor según el promedio.

`ejercicio_5:` Distancia entre actores

Funciones a implementar:

- `distancia(id_actor1, id_actor2)`: Devuelve la distancia entre los actores dados.

Dados dos actores o actrices, podemos intentar trazar un camino según las películas en las que actuaron; y definimos la **distancia** entre dos actores como **la longitud del camino mínimo** entre ellos.

Por ejemplo, si tenemos las siguientes películas:

- *Nueve Reinas* (Leticia Brédice, Gastón Pauls, Ricardo Darín)
- *Relatos salvajes*, (Rita Cortese, Ricardo Darín)

Podemos obtener las siguientes distancias:

- `distancia(Ricardo Darín, Ricardo Darín) -> 0` (por definición)
- `distancia(Ricardo Darín, Gastón Pauls) -> 1` (porque trabajaron juntos en *Nueve Reinas*)
- `distancia(Leticia Brédice, Rita Cortese) -> 2` (porque Brédice trabajó con Darín en *Nueve Reinas*, y Darín con Cortese en *Relatos Salvajes*)

Ayuda: una forma de implementar esta función es mediante el algoritmo de *búsqueda en anchura* (*breadth first search* en inglés), que en pseudocódigo sería así:

```
algoritmo distancia(A, B):
    Sea visitados := conjunto de actores visitados
    Sea Q := cola de tuplas (actor, distancia)

    visitados.agregar(A)  # marcamos A como visitado
    Q.encolar((A, 0))     # encolamos A con distancia 0

    mientras Q no está vacía:
        (V, D) = Q.desencolar()  # El actor V está a distancia D
        si V == B:               # V es el actor que estamos buscando?
            return D
        por cada film en el que trabajó V:
            por cada actor W que trabajó en ese film:
                si W no está en visitados:
                    visitados.agregar(W)  # marcamos W como visitado
                    Q.append((W, D + 1))  # encolamos W con distancia D + 1
```

Nota: no es necesario implementar una clase `Cola`, se permite utilizar una lista de Python o cualquier otra estructura de la biblioteca estándar.