Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

IIC2233 Programación Avanzada (2023-2)

Tarea 3

Entrega

- **Tarea y** `README.md`
  - **Fecha y hora oficial (sin atraso):** lunes 27 de noviembre de 2023, 20:00
  - **Fecha y hora máxima (2 días de atraso):** miércoles 29 de noviembre de 2023, 20:00.
  - **Lugar:** Repositorio personal de GitHub — Carpeta: `Tareas/T3/`

Objetivos

- Entender y aplicar el paradigma de programación funcional para resolver un problema.
- Manejar datos de forma eficiente utilizando herramientas de programación funcional:
  - Uso de `map`, `lambda`, `filter`, `reduce`, etc.
  - Uso de generadores y funciones generadoras.
  - Uso de diccionarios, listas y conjuntos por comprensión.
- Utilizar *web services* para obtener, procesar y/o guardar información de un contexto en particular:
  - Realización de consultas tipo `GET`, `POST`, `PATCH` y `DELETE`.
  - Uso de *token* para autentificación.

Índice

1. *DCCine*

¡Lograste liberar a **Conejochico**! ¿Pero a qué costo...?

En toda historia hay dos versiones, y en esta **Pacalein** entró en un bajón pues su mascota escapó, por lo que uno de sus amigos, **Hernan444** le recomienda probar el mundo de la realidad virtual para que pueda tener todas las mascotas que quiera. No muy convencida **Pacalein** terminó metiéndose a un servidor de realidad virtual y explorando diversos canales, terminó llegando a *DCCine*, un cine virtual que cuenta con un amplio repertorio y funciones a todas horas.

Luego de ser mucho tiempo cliente frecuente, el administrador de *DCCine*, **jtagle2**, acude a ella pues se enteró que era una experta programadora y una de las vacantes en el cine quedó libre. **Pacalein** después de tanto tiempo se sentía lista para seguir su vida sin su mascota a su lado, pero hacía tanto que no tocaba un código, que le pidió a tu versión de ese universo, que es experta en programación funcional y *webservices*, que le ayudara a conseguir el trabajo. ¡Ayuda a **Pacalein** a retomar su vida de adulta independiente!

Figura 1: Logo de *DCCine*

2. Flujo

En orden de ~~facilitar la corrección~~ lograr un código más ordenado, se pedirá que implementes como **mínimo**[1] una serie de **funciones** y **métodos**. Estas funciones y métodos están definidas en los archivos que les entregamos para esta evaluación. Solo debes completar estas funciones y métodos, es decir, no se debe cambiar su nombre, alterar los argumentos recibidos o cambiar lo que retornen.

Además, la corrección de esta tarea **será únicamente mediante el uso de *tests***, los cuales otorgarán puntaje de forma binaria (pasa todos los *test* o no obtiene el puntaje) a cada una de las funcionalidades a implementar.

En el directorio de la tarea encontrarás los siguientes archivos y directorios:

- Modificar `consultas.py`: Este archivo contiene las funciones a completar señaladas en la sección Programación Funcional.
- Modificar `peli.py`: Este archivo contiene la clase `Peliculas` con todos los métodos a completar señalados en la sección *Web Services*.
- No modificar `utilidades.py`: Este archivo contiene la implementación de las *namedtuples* a ocupar.
- No modificar `api.py`: Este archivo contiene todas las funcionalidades necesarias para levantar un servidor en el computador y poder ser consultado.
- No modificar `run_public_tests.py`: Este archivo es el que sirve para correr los *test* de la tarea.
- No modificar `test_publicos/`: Este directorio contiene los distintos *tests* de la evaluación.

[1] Puedes crear más funciones y métodos si lo consideras pertinente

3. Programación Funcional

Para ayudar a **Pacalein** a conseguir el puesto en *DCCine* necesita tu ayuda experta para entender cómo realizar una serie de consultas y aplicar los conocimiento del paradigma de la programación funcional.

Para esto, deberás interactuar con distintos tipos de datos, los cuales se explican con mayor detalle en Datos y completar las distintas funciones pedidas en Consultas.

3.1. Datos

Para que puedas implementar las funcionalidades, tendrás que interactuar con las siguientes *namedtuples*:

- `Personas`: Indica la información de una persona. Posee los atributos `id (int)`, `nombre (str)`, `genero (str)`, `edad (int)`.

Atributo	Descripción	Ejemplo
`id`	Un `int` que corresponde al identificador único de una persona.	573849
`nombre`	Un `str` que corresponde al nombre de la persona.	María
`genero`	Un `str` que corresponde al género de la persona.	Femenino
`edad`	Un `int` que corresponde a la edad de la persona.	12

- `Peliculas`: Indica la información de una película. Posee los atributos `id (int)`, `titulo (str)`, `genero (str)`, `rating (float)`.

Atributo	Descripción	Ejemplo
`id`	Un `int` que corresponde al identificador único de la película.	62764502
`titulo`	Un `str` que corresponde al título de la película.	Toy Story
`genero`	Un `str` que corresponde al género de la película.	Animación
`rating`	Un `float` que corresponde al *rating* de la película.	8.3

- `Funciones`: Indica las distintas funciones que tiene el cine para presentar sus películas. Posee los atributos `id (int)`, `numero_sala (int)`, `id_pelicula (int)`, `horario (int)`, `fecha (str)`.

Atributo	Descripción	Ejemplo
`id`	Un `int` que corresponde al identificador único de la función.	2826
`numero_sala`	Un `int` que corresponde al número de la sala donde se realiza la función.	1
`id_pelicula`	Un `int` que indica la película que tiene esa función.	85032662
`horario`	Un `int` que corresponde al módulo donde se realiza la función.	2
`fecha`	Un `str` correspondiente a la fecha en la que se da la función en formato `DD-MM-AA`. Puedes asumir que si una fecha tiene un año mayor o igual a 24, corresponde a los años 1900, mientras que si es menor a 24, corresponde a los años 2000.	25-12-97

- `Reservas`: Indica las reservas que ha hecho una persona para ver una función en específico. Posee los atributos `id_persona (int)`, `id_funcion (int)`, `numero_butaca (str)`.

Atributo	Descripción	Ejemplo
`id_persona`	Un `int` que corresponde al identificador de la persona que realizó la reserva.	573849
`id_funcion`	Un `int` que corresponde a la función en donde se realizó la reserva.	6237
`numero_butaca`	Un `str` que corresponde al número de butaca reservado.	A1

3.2. Consultas

Para poder ayudar a **Pacalein** a entrar al *DCCine* deberás completar consultas sobre los datos y se deben completar en el archivo `consultas.py`.

Importante: Se encuentra **prohibido el uso de ciclos `for` y `while`, el uso de los comandos que cambien el tipo de dato como `list()`, `tuple()`, `dict()`, `set()`, `entre otros`.**

Para completar las siguientes consultas podrás utilizar funciones como `lambda`, `map`, `filter`, `reduce`, `join`; las librerías `collections` e `itertools`. Se permite el uso de estructuras como listas, *sets*, diccionarios y generadores **siempre y cuando sean creadas por comprensión**.

El incumplimiento de cualquier regla explicada en los dos párrafos anteriores implicará nota mínima (1.0) en la evaluación completa.

3.2.1. Utiliza 1 generador

- Modificar `def peliculas_genero(generador_peliculas: Generator, genero: str) -> Generator:`[2]

  Recibe un generador con instancias de películas y un género de película. Retorna todas las películas donde su género sea `genero`. En caso de que ninguna película sea de ese género, se retorna un `Generator` vacío.

- Modificar `def personas_mayores(generador_personas: Generator, edad: int) -> Generator:`

  Recibe un generador con instancias de personas y una edad. Retorna todas las personas que tengan o sean mayores a dicha `edad`. En caso de que ninguna persona tenga esa o más edad, se retorna un `Generator` vacío.

- Modificar `def funciones_fecha(generador_funciones: Generator, fecha: str) -> Generator:`

  Recibe un generador con instancias de funciones y una fecha en formato `DD-MM-AAAA`. Retorna todas las funciones que se den en esa fecha. En caso de que ninguna función se realice en esa fecha, se retorna un `Generator` vacío.

- Modificar `def titulo_mas_largo(generador_peliculas: Generator) -> str:`

  Recibe un generador con instancias de películas y retorna el título más largo entre todas las películas. En caso de empate en el largo del título, se quedará la película con mayor *rating*. Finalmente, si se

[2] Para efectos del *output* de las consultas, se entiende por `Generator` los siguientes tipos de datos: listas, tuplas, diccionarios, *set*, generadores y objetos obtenidos a través de las funciones `map` y `filter`.

genera empate en el *rating* entre dos películas, se queda aquella que tenga una posición mayor en el generador.

- Modificar `def normalizar_fechas(generador_funciones: Generator) -> Generator:`

  Recibe un generador con instancias de funciones y retorna las instancias de funciones con las fechas normalizadas en el siguiente formato: `AAAA-MM-DD`.

- Modificar `def personas_reservas(generador_reservas: Generator) -> set:`

  Recibe un generador con instancias de reservas. Retorna un `set` con los identificadores de todas las personas que tienen al menos una reserva.[3] Puedes asumir que siempre existirá al menos una reserva en el generador.

- Modificar
```
def peliculas_en_base_al_rating(generador_peliculas: Generator,
                genero: str, rating_min: int, rating_max: int) -> Generator:
```

  Recibe un generador con instancias de películas, un género de películas y el rango de un *rating*. Retorna todos los títulos de las películas que sean del género recibido, y que su *rating* se encuentren en el rango indicado, el cual incluye el mínimo y el máximo, es decir puede tomar estos valores. En caso de que ninguna película cumpla con ser del género indicado o estar dentro del rango, se debe retornar un `Generator` vacío.

- Modificar `def mejores_peliculas(generador_peliculas: Generator) -> Generator:`

  Recibe un generador con instancia de películas. Retorna las 20 primeras películas con mayor *rating*. Por un lado, si el generador original contiene menos de 20 películas, se retornan todas. Por otro lado, si una o más películas estén empatando por *rating*, se utilizará el `id` para desempatar. En particular, se deberá elegir primero las películas con `id` más pequeño y luego aquellas con `id` más grande.

- Modificar
```
def pelicula_genero_mayor_rating(generador_peliculas: Generator,
                                 genero: str) -> str:
```

  Recibe un generador de instancias de películas y un género de película. Retorna el título de la película con mayor *rating* que pertenece al género entregado, en caso de haber más de una película con el mismo *rating*, se queda aquella que tenga una posición menor en el generador. Si no existe película con el género entregado, se debe retornar un *string* vacío (`""`).

3.2.2. Utiliza 2 generadores

- Modificar
```
def fechas_funciones_pelicula(generador_peliculas: Generator,
                generador_funciones: Generator, titulo: str) -> Generador:
```

  Recibe 2 generadores, con instancias de películas y funciones, y el título de una película. Retorna las fechas de todas las funciones de la película recibida. En caso de existir más de una función el mismo día, debe retornar dicha fecha las veces que cumpla con las condiciones, es decir, la respuesta puede contener fecha repetidas. Puede ocurrir que la película en cuestión no tenga ninguna función, en dicho caso debes retornar un generador vacío.

- Modificar
```
def genero_mas_transmitido(generador_peliculas: Generator,
                generador_funciones: Generator, fecha: str) -> str:
```

  Recibe 2 generadores, con instancias de películas y funciones, y una fecha en formato `DD-MM-AAAA`. Retorna el género de película más visto en la fecha recibida. En caso de 2 o mas géneros mas vistos,

[3] **Hint**: Recuerda cómo construir estructuras por compresión.

retorna el primero obtenido. En caso de que la fecha no tenga ninguna función, se debe retornar un *string* vacío (`""`).

- Modificar `def id_funciones_genero(generador_peliculas: Generator, generador_funciones: Generator, genero: str) -> Generator:`

  Recibe 2 generadores, con instancias de películas y funciones, además de un género de película. Retorna los identificadores de las funciones que transmitan películas del género recibido. Si no hay funciones que transmitan películas de ese género, retorna un `Generator` vacío.

- Modificar `def butacas_por_funcion(generador_reservas: Generator, generador_funciones: Generator, id_funcion: int) -> int:`

  Recibe 2 generadores con instancias de reservas y funciones, además de el identificador de una función. Retorna la cantidad de butacas ocupadas para la función recibida. Si no hay butacas ocupadas en la función recibida, retorna 0.

- Modificar `def salas_de_pelicula(generador_peliculas: Generator, generador_funciones: Generator, nombre_pelicula: str) -> Generator:`

  Recibe 2 generadores, con instancias de películas y funciones, además del nombre de una película. Retorna todos los números de sala donde se transmita la película recibida. En caso de que la película no exista o no tenga funciones, se retorna un `Generator` vacío.

3.2.3. Utiliza 3 o más generadores

- Modificar `def nombres_butacas_altas(generador_personas: Generator, generador_peliculas: Generator, generador_reservas: Generator, generador_funciones: Generator, titulo: str, horario: int) -> set:`

  Recibe generadores con instancias de personas, películas, reservas y funciones, además del título de una película y un horario. Retorna los nombres de todas las personas que fueron a ver la película indicada que se transmitió en una función dada en `horario`. Puedes asumir que siempre habrá al menos una persona que fue a ver dicha película en ese horario.

- Modificar `def nombres_persona_genero_mayores(generador_personas: Generator, generador_peliculas: Generator, generador_reservas: Generator, generador_funciones: Generator, nombre_pelicula: str, genero: str, edad: int) -> set:`

  Recibe generadores con instancias de personas, películas, reservas y funciones, además del titulo de una película, el género de una persona y una edad. Retorna los nombres de todas las personas que fueron a ver la película indicada, son del género `género` y su edad es mayor o igual a `edad`. En caso de que ninguna persona cumpla con todos los requerimientos, se retorna un `set` vacío.

- Modificar `def genero_comun(generador_personas: Generator, generador_peliculas: Generator, generador_reservas: Generator, generador_funciones: Generator, id_funcion: int ) -> str`

  Recibe generadores con instancias de personas, películas, reservas y funciones, además del identificador de una función. Retorna el género mayoritario que hay en la función de *id* `id_funcion`, en un *string* del formato:

  ```
  En la función {id_funcion} de la película {nombre_película} la mayor parte del público es {género}.
  ```

En el caso de que haya un empate entre 2 géneros:

```
En la función {id_funcion} de la película {nombre_película} se obtiene que la mayor
parte del público es de {genero_1} y {genero_2} con la misma cantidad de personas.
```

Finalmente, en el caso de que haya un empate entre todos los géneros que existen:

```
En la función {id_funcion} de la película {nombre_película} se obtiene que la cantidad
de personas es igual para todos los géneros.
```

- Modificar `def edad_promedio(generador_personas: Generator, generador_peliculas: Generator, generador_reservas: Generator, generador_funciones: Generator, id_funcion: int) -> str`

  Recibe generadores con instancias de personas, películas, reservas y funciones, además del identificador de una función. Retorna la edad promedio que hay en la función de *id* **`id_funcion`**. La edad promedio deber ser transformada a un **valor entero** utilizando el método de aproximación por techo. El *string* que se retorna tiene que tener el siguiente formato:

  ```
  En la función {id_funcion} de la película {nombre_película} la edad promedio del
  público es {promedio}.
  ```

  Puedes asumir que en la función a evaluar siempre habrá al menos una reserva.

- Modificar `def obtener_horarios_disponibles(generador_peliculas: Generator, generador_reservas: Generator, generador_funciones: Generator, fecha_funcion: str, reservas_maximas: int) -> set:`

  Recibe generadores con instancias de películas, reservas y funciones, además de una fecha en el formato `DD-MM-YY` y una cantidad de reservas máximas. Retorna los horarios que tengan butacas disponibles para ver la película con mayor *rating* en la fecha `fecha_funcion`. Para esta función puedes asumir que no existirá empate en el *rating*, que `reservas_maximas` siempre será un número mayor o igual a 1 y que en la fecha indicada siempre existirá al menos 1 función o más.

  Para saber si una función tiene butacas disponibles se debe tener en cuenta la cantidad de reservas máximas dadas por `reservas_maximas` que puede tener una función, donde si una función tiene una cantidad igual o mayor a `reservas_maximas` entonces no está disponible. Puede ocurrir que ninguna función esté disponible, en cuyo caso de retorna el `set` vacío.

- Modificar `def personas_no_asisten(generador_personas: Generator, generador_reservas: Generator, generador_funciones: Generator, fecha_inicio: str, fecha_termino: str) -> Generator:`

  Recibe generadores con instancias de personas, reservas y funciones, además de una fecha de inicio y término en el formato `DD-MM-YYYY`. Retorna todas las instancias de personas que no hicieron una reserva entre `fecha_inicio` y `fecha_termino`, inclusive. Puedes asumir que siempre habrá por lo menos una persona que cumpla con lo anterior.

4. *Web Services*

Ahora que **Pacalein** ya empezó a trabajar en el *DCCine* ha decidido relajarse un poco más y empezar a disfrutar de su trabajo en el cine. Su jefe, viéndola muy comprometida ha decidido darle la oportunidad de opinar con respecto a la cartelera del cine y así poder elegir qué películas proyectar.

4.1. API

Para hacer todo lo anterior, deberás trabajar con una API que maneja la información de distintas películas que se proyectan en el cine.

4.1.1. *Endpoints*

A continuación se describen los distintos *endpoints* que presenta esta API:

- `GET` `/`

  Retorna un mensaje personalizado. A continuación se deja un ejemplo la *response* de la consulta:

  Response

  ```
  {
    "result": "Hoy es 2023-05-23..."
  }
  ```

- `GET` `/peliculas`

  Retorna los títulos de todas las películas que se encuentran en la base de datos de la API. A continuación se deja un ejemplo la *response* de la consulta:

  Response

  ```
  {
    "result": [
      "Sherk", "El Dorado", "Spirit"
    ]
  }
  ```

- `GET` `/informacion`

  Entrega la sinopsis asociada a una película. Para indicar la película, deberá indicar el título en el parámetro `pelicula`.

  A continuación se deja un ejemplo la *response* de la consulta, donde la película entregada a través del parámetro `pelicula` sería `"Kung Fu Panda"`:

  Response

  ```
  {
    "result": "El panda Po, un fan..."
  }
  ```

- **GET** `/aleatorio`

  Retorna un enlace válido de la API. Tras realizar la *request*, la API retornará un enlace; en caso contrario, se indicará la causa del error.

  A continuación se deja un ejemplo la *response* de la consulta:

Response

```
{
  "result": "http://.../informacion?pelicula=Coco"
}
```

- **POST** `/update`

  Agrega una película y su respectiva sinopsis a la base de datos de la API. Para esto, en el *body* se debe indicar el título de la película y su sinopsis.

  Tras realizar la *request*, la API retornará el resultado de la operación. Si es exitoso, se indicará que se agregó la información correctamente; en caso contrario, se indicará la causa del error.

  A continuación se deja un ejemplo del *body* y la *response* de la consulta:

Body

```
{
  "pelicula": "Up!",
  "sinopsis": "Aventuras de un..."
}
```

Response

```
{
  "result": "Información agregad..."
}
```

- **PATCH** `/update`

  Actualiza la sinopsis de una película en la base de datos de la API. Para esto, en el *body* se debe indicar el título de la película y su nueva sinopsis.

  Tras realizar la *request*, la API retornará el resultado de la operación. Si es exitoso, se indicará que se actualizó la información correctamente; en caso contrario, se indicará la causa del error.

  A continuación se deja un ejemplo del *body* y la *response* de la consulta:

Body

```
{
  "pelicula": "Up!",
  "sinopsis": "Ellie + Carl"
}
```

Response

```
{
  "result": "Información actuali..."
}
```

- DELETE `/remove`

  Elimina una película de la base de datos de la API. Para esto, en el *body* se debe indicar el título de la película a eliminar.

  Tras realizar la *request*, la API retornará el resultado de la operación. Si es exitoso, se indicará que se eliminó la película correctamente; en caso contrario, se indicará la causa del error.

  A continuación se deja un ejemplo del *body* y la *response* de la consulta:

Body	***Response***

`{`
`  "pelicula": "Up!",`
`}`
	`{`
	`  "result": "Información elimina..."`
	`}`

4.1.2. Autentificación

Es importante saber que la API requiere *headers* de autenticación para cualquier *request* que realice cambios en su base de datos. Por esto, será necesario utilizar un *token* de autentificación para realizar consultas del tipo `POST`, `PATCH` y `DELETE`.

La forma de enviar la autenticación es incluir un diccionario como *header* de la consulta que tenga el siguiente formato:

```
{"Authorization": token_autentificacion}
```

El *token* de autentificación te será entregado al momento de realizar consultas en que lo necesite.

4.2. Consultas

Para poder utilizar la API, deberás completar las distintas consultas de la clase `Peliculas`, la cual se encuentran en el archivo `peli.py`. **Esta es la única parte de la tarea donde sí puedes hacer uso de estructuras de datos, `for` y `while`.**

La clase `Pelicula` cuenta con el atributo `base`, el cual deberás utilizar para realizar todas las *requests* a la API. A su vez, cuenta con múltiples métodos, los cuales deberás completar. Estos se explican a continuación:

4.2.1. Obtener información

Todos los métodos que se explican a continuación, realizan consultas que entregan información. Por lo mismo, todos estos métodos deben realizar *requests* del tipo `GET`.

- Modificar `def saludar(self) -> dict:`

  Realiza una *request* del tipo `GET` al *endpoint* `"/"` para obtener un mensaje de la API. Retorna un diccionario con 2 elementos: el *status code* de la consulta y el mensaje personalizado de la API contenido en `"result"`. El formato del diccionario a retornar es:

  ```
  {"status-code": ..., "saludo": ...}
  ```

- Modificar `def verificar_informacion(self, pelicula: str) -> bool:`

  Recibe el nombre de una película. Con este dato, deberás realizar una *request* del tipo `GET` al *endpoint* `"/peliculas"` y verificar si entre dichas películas, se encuentra la película consultada.

Este método debe retornar un booleano que indique si es que la película consultada existe entre las películas disponibles en la API.

- Modificar `def dar_informacion(self, titulo_pelicula: str) -> dict:`

  Recibe el título de una película. Con este dato, deberás realizar una *request* del tipo `GET` al *endpoint* `"/informacion"` para obtener la información asociada a dicha película.

  Este método debe retornar un diccionario con 2 elementos: el *status code* y la respuesta de la API contenido en `"result"`. El formato del diccionario a retornar es:

  ```
  {"status-code": ..., "mensaje": ...}
  ```

- Modificar `def dar_informacion_aleatoria(self) -> dict:`

  Realiza una *request* al *endpoint* `"/aleatorio"`. Si la consulta entrega un enlace a la API, se utiliza para realizar una segunda *request* del tipo `GET`.

  Este método debe retornar un diccionario con 2 elementos: el *status code* y la respuesta de la API contenido en `"result"`. El formato del diccionario a retornar es:

  ```
  {"status-code": ..., "mensaje": ...}
  ```

  El contenido de `"status-code"` y `"mensaje"` dependerá de la respuesta obtenida en la primera consulta:

  - Si es 200: el `"status-code"` corresponderá al *status code* de la segunda consulta y `"mensaje"` será la respuesta enviada por esta segunda consulta dentro de `"result"`.
  - Si no es 200: el `"status-code"` corresponderán al *status code* de la primera consulta y el `"mensaje"` será la respuesta enviada dentro de `"result"` por esta consulta.

4.2.2. Modificar información

En esta segunda parte deberás hacer diferentes consultas para agregar, eliminar y modificar información de la base de datos, por lo que deberás realizar *requests* del tipo `POST`, `PUT` y `DELETE` según corresponda.

- Modificar `def agregar_informacion(self, pelicula: str, sinopsis: str, access_token: str) -> str:`

  Recibe el nombre de una película, una sinopsis y un *token* de acceso a la API. Con estos datos, deberás realizar una *request* de tipo `POST` al *endpoint* `"/update"`, para agregar la película a la base de datos de la API, junto con el mensaje respectivo.

  Este método retorna un mensaje en función del *status code* de la *request*:

  - Si es 401: Se retorna `"Agregar pelicula no autorizado"`.
  - Si es 400: Se retorna el mensaje contenido dentro del JSON en `"result"`.
  - En otro caso: Se retorna `"La base de la API ha sido actualizada"`.

- Modificar `def actualizar_informacion(self, pelicula: str, sinopsis: str, access_token: str) -> str:`

  Recibe el nombre de una película, una sinopsis y un *token* a acceso de la API. Con estos datos, deberás realizar una *request* de tipo `PATCH` al *endpoint* `"/update"`, para actualizar la sinopsis de la película en la base de datos de la API.

  Este método retorna un mensaje en función del *status code* de la *request*:

- Si es 401: Se retorna "Editar información no autorizado".
- Si es 200: Se retorna "La base de la API ha sido actualizada".
- En otro caso: Se retorna el mensaje contenido dentro del JSON en "result".

- Modificar `def eliminar_pelicula(self, pelicula: str, access_token: str) -> str:`

  Recibe el nombre de una película y un *token* de acceso a la API. Con estos datos, deberás realizar una *request* de tipo `DELETE` al *endpoint* `"/remove"`, para eliminar la película en la base de datos de la API.

  Este método retorna un mensaje en función del *status code* de la *request*:

  - Si es 401: Se retorna "Eliminar pelicula no autorizado".
  - Si es 200: Se retorna "La base de la API ha sido actualizada".
  - En otro caso: Se retorna el mensaje contenido dentro del JSON en "result".

5. `.gitignore`

Para esta tarea **deberás utilizar un** `.gitignore` para ignorar los archivos indicados, este deberá estar dentro de tu carpeta `Tareas/T3/`. Puedes encontrar un ejemplo de `.gitignore` en el siguiente ***link***.

Los elementos a ignorar para esta tarea son:

- El enunciado.
- Los archivos que no debes modificar: `api.py`, `utilidades.py`, `run_public_tests.py`.
- La carpeta `test_publicos/`.

Recuerda **no ignorar archivos vitales de tu tarea como los que tú debes modificar, o tu tarea no podrá ser revisada**.

Es importante que hagan un correcto uso del archivo `.gitignore`, es decir, los archivos **deben** no subirse al repositorio debido al uso correcto del archivo `.gitignore` y no debido a otros medios.

6. Entregas atrasadas

La entrega atrasada de esta tarea sigue la política de atrasos del curso. Posterior a la fecha de entrega de la tarea se abrirá un formulario de Google Form, en caso de que desees que se corrija un *commit* posterior al recolectado, deberás señalar el nuevo *commit* en el *form*.

El plazo para rellenar el *form* será de 48 horas, en caso de que no lo contestes en dicho plazo, se procederá a corregir el *commit* recolectado.

7. Importante: Corrección de la tarea

Para esta tarea, el carácter funcional del programa será el pilar de la corrección, es decir, **sólo se corrigen tareas que se puedan ejecutar**. Por lo tanto, se recomienda hacer periódicamente pruebas de ejecución de su tarea y *push* en sus repositorios, corroborando que cada archivo de *tests* que les pasamos corra en un tiempo acotado de 5 minutos, en caso contrario se asumirá un resultado incorrecto.

En el **siguiente enlace** se encuentra la distribución de puntajes, cada una con el puntaje asociado a los *test*.

Importante: Todo ítem corregido automáticamente es evaluado de forma binaria: pasa todos los *tests* de dicho ítem o no los pasa todos (esto sea por error o *timeout*). Finalmente, todos los descuentos serán asignados manualmente por el cuerpo docente.

Una vez recopilado todo el contenido de la entrega, esta se ubicará en un directorio con nombre distinto a "`T3/`". Por lo tanto, *paths* que hagan uso de las rutas "`T3/...`" o "`Tareas/T3/...`" no se ejecutarán de forma correcta y el programa no funcionará. Es por esto que los *paths* usados deben ser relativos, asumiendo que al ejecutar el programa se ubicará la terminal dentro de la misma carpeta que contiene el contenido de la entrega, y que dicha carpeta no se llama "`T3/`".

Para terminar, si durante la realización de tu tarea se te presenta algún problema o situación que pueda afectar tu rendimiento, no dudes en contactar al ayudante de Bienestar de tu sección. El correo está en el **siguiente enlace**.

8. Restricciones y alcances

- Esta tarea es **estrictamente individual**, y está regida por el **Código de honor de Ingeniería**.
- Tu programa debe ser desarrollado en Python 3.10.
- Tu programa debe estar compuesto por uno o más archivos de extensión `.py` que estén correctamente ordenados por carpeta.
- Si no se encuentra especificado en el enunciado, supón que el uso de cualquier librería Python está prohibido. Pregunta en la *issue* especial del **foro** si es que es posible utilizar alguna librería en particular.
- Debes adjuntar un archivo `README.md` **conciso y claro**, donde describas los alcances de tu programa, cómo correrlo, las librerías usadas, los supuestos hechos, y las referencias a código externo. El no incluir este archivo o bien incluirlo pero que se encuentre vacío conllevará un **descuento** en tu nota.
- Esta tarea se debe desarrollar **exclusivamente** con los contenidos liberados al momento de publicar el enunciado. No se permitirá utilizar contenidos que se vean posterior a la publicación de esta evaluación.
- Se encuentra estrictamente prohibido citar código que haya sido publicado **después de la liberación del enunciado**. En otras palabras, solo se permite citar contenido que ya exista previo a la publicación del enunciado.
- Cualquier aspecto no especificado queda a tu criterio, siempre que no pase por sobre otro que sí sea especificado por enunciado.

Las tareas que no cumplan con las restricciones del enunciado obtendrán la calificación mínima (1,0)**.**