Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación
IIC2233 — Programación Avanzada 2025-2

Publicación: 25 de septiembre de 2025
Actividad evaluada

Actividad 4
Programación Funcional

Entrega

- **Lugar:** Repositorio personal de GitHub — Carpeta: `Actividades/AC4`
- **Fecha máxima de entrega:** 25 de septiembre 17:20
- **Ejecución de actividad**: La Actividad será ejecutada **únicamente** desde la terminal del computador. **Los *paths* relativos utilizados en la Actividad deben ser coherentes con esta instrucción, y no pueden modificarse.**

Importante: Antes de comenzar, comprueba que Git este funcionando correctamente en tu repositorio privado. Para esto, **sube los archivos base de la actividad de inmediato** (*add*, *commit*, *push*). Se espera que en esta actividad (así como en las demás actividades y tareas) utilices Git a lo largo de **todo tu desarrollo** como una herramienta, no sólo como un método de entrega. Es por esto que recomendamos enfáticamente que vayas subiendo tus cambios constantemente (***push***), ya que **problemas de último minuto** relacionados con la entrega y Git **no serán considerados**.

Objetivo de la actividad

- Implementar una función generadora, utilizando correctamente `yield`.
- Aplicar conocimientos de iterables utilizando funciones `map`, `filter` y `reduce`.
- Utilizar librerías *built-ins*.

Introducción

Durante la semana de receso, todos los miembros del DCC se proponían descansar viendo sus películas favoritas. Con un montón de cabritas saladas y listos para empezar una buena noche de cine, notaron que una entidad maligna había atacado todos los servicios de *stream* presentes en el internet y que nadie podía disfrutar de su preciado tiempo libre.

Agobiados por esta situación, tú junto a un grupo estudiantes del DCC se decidieron a programar su propia plataforma de *stream*: ***DCCine+***. Para ayudarlos, estarás encargado de: cargar la información de las películas y sus categorías, e implementar una serie de consultas para operar sobre la información cargada.

Flujo del programa

Esta actividad consta en completar 2 partes aplicando el paradigma de Programación Funcional. La primera referente a la carga de datos a partir de un archivo, pre-procesar los datos y retornarlos en un formato específico. Luego, la segunda parte consiste en completar una serie de funciones para consultar los datos. Todas estas partes serán corregidas exclusivamente mediante el uso de *tests*.

Debes asegurarte de entregar, como mínimo, el archivo que tenga el *tag* de Entregar en la siguiente sección. Los demás archivos no es necesario subir, pero tampoco se penalizará si se suben al repositorio personal.

Archivos

En el directorio de la actividad encontrarás los siguientes archivos:

Archivos de datos

- No modificar `archivos/peliculas.csv`: Este archivo contiene la información de las películas disponibles. El formato del archivo es:

  ```
  id,titulo,director,año_estreno,rating_promedio
  ```

  donde `id` y `año_estreno` corresponden a números enteros, mientras que `rating_promedio` corresponde a un número decimal.

- No modificar `archivos/generos.csv`: Este archivo contiene la información de todos los géneros de las películas del archivo anterior. Una misma película puede estar relacionada con uno o más géneros. El formato del archivo es:

  ```
  genero,id_pelicula
  ```

  donde `id_pelicula` corresponde a un número entero.

Archivos de código

- Entregar Modificar `funciones.py`: Contiene las funciones necesarias para cargar y manejar los distintos tipos de consultas.
- No modificar `utilidades.py`: Contiene la definición de *namedtuples* y funciones necesarias para cargar y manejar la información.

  Las *namedtuples* implementadas son las siguientes:

  - `Pelicula` Posee los atributos `id_pelicula` (`int`), `titulo` (`str`), `director` (`str`), `estreno` (`int`) y `rating` (`float`).
  - `Genero` Posee los atributos `genero` (`str`) y `id_pelicula` (`int`).

- No modificar `main.py`: Contiene código para ejecutar las diferentes funciones sin el uso de *tests*.
- No modificar `tests_publicos`: Carpeta que contiene diferentes `.py` para ir probando si lo desarrollado hasta el momento cumple con lo esperado. **En la última hoja del enunciado se encuentra un anexo de cómo ejecutar los *tests* por parte o todos**.

Parte I. Cargar datos

Para cargar los datos de las distintas películas, deberás utilizar las *namedtuples* entregadas en el archivo `utilidades.py` y completar la siguiente función del archivo `funciones.py`:

- Modificar `def cargar_peliculas(ruta: str) -> Generator:`

  Esta función generadora recibe un `str` con la **ruta de un archivo** que contiene la información de las películas. Retorna un generador que entrega instancias de `Pelicula` según el contenido del archivo.

  Debes asegurarte abrir el archivo utilizando el *encoding* `UTF-8`, utilizar la *namedtuple* definida en el archivo `utilidades.py` y que los atributos de cada película sean guardados como su tipo de dato correspondiente.

  Finalmente, es importante destacar que el argumento `ruta` corresponde a la ruta del archivo a cargar, por ejemplo, `archivos/peliculas.csv`, `tests_publicos/peliculas.csv`, `peliculas_2.csv`, por lo que no es únicamente el nombre del archivo.

Parte II. Consultas

Para poder manejar el *DCCine+*, deberás completar una serie de consultas que trabajarán sobre los datos cargados.

Importante

> En esta segunda parte de la actividad, se espera que apliquen exclusivamente los contenidos de Programación Funcional y un uso correcto de generadores. Para lograr este objetivo, se espera que apliquen correctamente diferentes funciones como `map`, `filter` y `reduce`, uso de funciones anónimas (`lambda`) e *itertools*.
>
> Además, para forzar la aplicación exclusiva de los contenidos, es que se encuentra **estrictamente prohibido** el uso de: ciclos `for` y `while`; estructuras de datos `list`, `tuple`, `dict`, `set` definidas de forma normal o por comprensión; y otras librerías diferentes a las dadas en los archivos base.

Las funciones a completar en el archivo `funciones.py` son:

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

  Recibe un generador con instancias de `Pelicula`. Retorna un generador con los nombres de todos los directores, sin importar si están repetidos.

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

  Recibe un generador con instancias de `Pelicula`. Retorna un *string* con todos los títulos de las películas concatenados por una coma y un espacio (`", "`). Si no hay películas por concatenar, se retorna un *string* vació (`""`).

- Modificar `def filtrar_peliculas(generador_peliculas: Generator, director: str | None, rating_min: float | None, rating_max: float | None) -> Generator:`

  Recibe un generador con instancias de `Pelicula`. Además, puede recibir el nombre de un director, un *rating* mínimo y un *rating* máximo. Retorna un generador con las películas filtradas.

Las películas se filtran de forma que, en caso de haberse indicado:

- El nombre de un director: se filtran las películas de forma que solo queden las películas que tengan el mismo director que el indicado.
- Un *rating* mínimo: se filtran las películas de forma que solo queden las películas que tengan un *rating* equivalente o mayor al entregado.
- Un *rating* máximo: se filtran las películas de forma que solo queden las películas que tengan un *rating* equivalente o menor al entregado.

- Modificar `def filtrar_titulos(generador_peliculas: Generator, director: str, rating_min: float, rating_max: float) -> str:`

  Recibe un generador con instancias de `Pelicula`, el nombre de un director, un *rating* mínimo o un *rating* máximo. Esta función primero filtra las películas para seleccionar solo aquellas que tengan el mismo director que el indicado, tengan un *rating* igual o mayor al `rating_min` y un *rating* igual o menor al `rating_max`.

  Retorna un *string* con todos los títulos de las películas filtradas. Los títulos deben estar concatenados por una coma y un espacio (`", "`). Si no hay películas por concatenar, se retorna un *string* vacío (`""`).

- Modificar `def filtrar_peliculas_por_genero(generador_peliculas: Generator, generador_generos: Generator, genero: str | None) -> Generator:`

  Recibe un generador con instancias de `Pelicula`, un generador con instancias de `Generos` y puede recibir el nombre de un género de película. Retorna un generador que contiene todos los pares del generador de películas y el generador de géneros que:

  1. Correspondan a la misma película, es decir, que ambos elementos del par tengan el mismo id de película.
  2. El género corresponda al indicado en el *input*. Si no se indica un género, entones solo se deben retornar todos los pares que cumplen con el punto 1.

  Para lograr lo anterior, deberás investigar y utilizar la función `product` de la librería `itertools`.

Notas

- **No puedes hacer *import* de otras librerías externas a las entregadas en el archivo a completar.**
- Recuerda que la ubicación de tu entrega es en **tu repositorio de Git**. En la rama (*branch*) por defecto del repositorio: `main`.
- Recuerda que esta evaluación presenta corrección **automatizada**. Si entregas un código que se cae al momento de correr los *tests*, será evaluado con 0 puntos.
- Puedes probar tu código con los *tests* y ejecutando `main.py`.
- Si aparece un error inesperado, ¡léelo y revisa el código del *test*! Intenta interpretarlo y/o buscarlo en Google.

Ejecución de *tests*

En esta actividad se provee de varios archivos `.py` los cuáles contiene diferentes *tests* que ayudan a validar el desarrollo de la actividad.

Importante

> En esta Actividad los *tests* correspondientes a la Parte II verificarán que no se usen los ciclos `for`, `while`, ni estructuras prohibidas.

Para ejecutar estos *tests*, **primero debes posicionar tu terminal/consola en la carpeta de la actividad (`Actividades/AC4`)**. Luego, desde esta misma, debes escribir el siguiente comando para ejecutar todos los *tests* de la actividad:

- `python3 -m unittest discover tests_publicos -v`

En cambio, si deseas ejecutar un subconjunto de *tests*, puedes hacerlo si escribes lo siguiente en la terminal/consola:

- `python3 -m unittest -v tests_publicos.test_cargar_datos`
  Para ejecutar solo el subconjunto de *tests* relacionado a la Parte I.
- `python3 -m unittest -v tests_publicos.test_obtener_directores`
  Para ejecutar solo el subconjunto de *tests* relacionado a la consulta de `obtener_directores` de la Parte II.
- `python3 -m unittest -v tests_publicos.test_obtener_str_titulo`
  Para ejecutar solo el subconjunto de *tests* relacionado a la consulta de `obtener_str_titulos` de la Parte II.
- `python3 -m unittest -v tests_publicos.test_filtrar_peliculas`
  Para ejecutar solo el subconjunto de *tests* relacionado a la consulta de `filtrar_peliculas` de la Parte II.
- `python3 -m unittest -v tests_publicos.test_filtrar_titulos`
  Para ejecutar solo el subconjunto de *tests* relacionado a la consulta de `filtrar_titulos` de la Parte II.
- `python3 -m unittest -v tests_publicos.test_filtrar_peliculas_genero`
  Para ejecutar solo el subconjunto de *tests* relacionado a la consulta de `filtrar_peliculas_por_genero` de la Parte II.

Importante: recuerda que si `python3` no funciona, probar con el comando específico de tu computador. Este puede ser `py`, `python`, `py3` o `python3.12`.