Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación
IIC2233 — Programación Avanzada 2021-2

30 de Septiembre de 2021
Actividad Formativa

Actividad Formativa 3

Iterables e Iteradores

Entrega

- **Lugar:** En su repositorio privado de GitHub, en la **carpeta** `Actividades/AF3/`
- **Hora del *push*:** 16:40

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**.

Introducción

El DCCinepolis ha llegado a Chile y compró todos los cines que encontró para poder reproducir películas, pero lamentablemente debido a las medidas sanitarias casi nadie va a los cines por lo que no está saliendo rentable el negocio. Como ahora la gente ve las películas *online*, al Jefe de marketing de DCCinepolis se le ha ocurrido una excelente idea... DCCabritas: un sitio web donde se recomendarán y reproducirán listas de películas personalizadas, según las preferencias de los usuarios y entre grupos de amigos. Tienen casi todo listo: el sitio, los primeros usuarios y las películas. Solo resta implementar los algoritmos que armarán las listas de películas para los usuarios y es por eso que te buscan a ti, ya que escucharon de tus fantásticas habilidades en el uso de iterables e iteradores.

Flujo del programa

La ejecución del programa parte con el inicio de sesión. Primero se muestran todos los usuarios en la consola y se selecciona con un número aquel con el que deseas ingresar. Una vez iniciada la sesión, el programa creará una `ListaReproduccion` llamada "Principal", la que contendrá las 20 películas que más puntaje tengan respecto a las preferencias del usuario seleccionado. Luego de lo anterior, con la sesión iniciada, tendrás varias opciones:

- **Cerrar Sesión**: Permite volver al menú de elección de `Usuario`
- **Reproducir Lista**: Esta es la funcionalidad central de DCCabritas. Si la seleccionas, se mostrarán todas las `ListaReproduccion` del `Usuario`. Al elegir una, ¡se reproducirá por completo! Todes les `Usuario` vienen con una lista "Principal" por defecto, que contiene las 20 películas que más afinidad tienen con la persona.
- **Crear Watch Party!**: Como ver películas solos a veces puede ser un poco fome, DCCabritas le da esta opción a los usuarios para que puedan formar listas compartidas. Esta funcionalidad crea una nueva lista de reproducción que mezcla la lista principal del usuario actual con la de otros usuarios, tantos como se desee (pero si son muchos, quizás no hayan películas en común). Esta lista queda guardada en tus listas y la de los usuarios elegidos.
- **Recomendar Amigue**: Como no conocemos a todos los integrantes de la plataforma, queremos crear algún método que nos recomiende al usuario que más se parezca a nosotres. Esta funcionalidad nos recomendará a otra persona de **DCCabritas** con quien tenemos gustos muy compatibles, para poder invitarle a un *DCCabritas and DCChill*.

Archivos

Archivos de datos

Ambos archivos están en la carpeta datos

- `usuarios.csv`: En este archivo encontrarás a los distintos usuarios, sus preferencias sobre los géneros de películas y un actor o actriz que el usuario quiere evitar ver. El formato se verá así:

```
Nombre,Categoria1,Categoria2,...,Categoria6,ActorProhibido
```

Atributo	Descripción
Nombre	Nombre del usuario.
Categorías	Afinidad del usuario con la categoría, un valor del 0 al 5 (incluye a ambos).
ActorProhibido	Actor o actriz que el usuario no quiere ver.

- `peliculas.csv`: En este archivo encontrarás los datos de las distintas películas que el DCCabritas puede mostrar, y su puntuación en distintos géneros. El formato se verá así:

```
Nombre,Categoria1,Categoria2,...,Categoria6;Actor1;...;ActorN
```

Atributo	Descripción
Nombre	Nombre de la película.
Categorías	Puntuación del 1 al 5 sobre las categorías de la película.
Actores	Actores que participan en la película (pueden ser más de 1).

Las categorías son `deportes`, `arte`, `ciencia ficción`, `fantasía`, `comedia` y `acción`. Cada película tiene asociado un valor del 1 al 5 dependiendo "qué tanta relación" hay entre la película y una categoría en particular (notar que no es algo absoluto, una película de comedia puede tener una alta puntuación en arte también). Por ejemplo, "Los Guardianes De La Galaxia" es una película de `ciencia ficción` y `comedia`, pero además hay `acción`; por lo que tendrá un puntaje medio en esa categoría. No tiene casi nada de `arte` o `deportes`, por lo que tendrá un puntaje asociado a estas categorías muy pequeño (no necesariamente cero).

Archivos de código

En el directorio de la actividad encontrarás los siguientes archivos de código:

- `main.py`: Archivo principal. Es el que debes ejecutar. No debes modificarlo
- `cargar_archivos.py`: Este archivo se encarga de cargar los datos de los archivos, por medio de las funciones `cargar_peliculas` y `cargar_usuarios`. No debes modificarlo
- `dccabritas.py`: Este archivo contiene a la clase `DCCabritas`. No debes modificarlo
- `funciones.py`: Este archivo contiene las funciones `filtrar_prohibidos`, `calcular_afinidades`, `encontrar_usuario_mas_afin` y `encontrar_peliculas_comunes`. Debes modificarlo
- `lista_reproduccion.py`: Este archivo contiene la clase `ListaReproduccion` y la clase `IterarLista`. Debes modificarlo
- `usuario.py`: Este archivo contiene la clase `Usuario`. Debes modificarlo
- `pelicula.py`: Este archivo contiene la clase `Pelicula`. No debes modificarlo
- `parametros.py`: Archivo que contiene los parámetros utilizados, como las categorías y las rutas. No debes modificarlo

Clases Implementadas

A continuación se describen dos de las clases con las que vas a tener que trabajar para esta actividad, que se encuentran en los archivos `dccabritas.py` y `pelicula.py`. Cada una de estas clases es utilizada de manera interna, y solo dejamos una breve explicación de **algunos de** los métodos y atributos que podrías necesitar.

Clase DCCabritas

`class DCCabritas:` No debes modificarlo
Representa al `DCCabritas`, se encarga de la mayor parte del flujo del programa e incluye los siguientes métodos y atributos:

- `def __init__(self):` Constructor de la clase, solamente se cargan los datos de los usuarios y se guardan en un diccionario:
  - `self.usuarios`: Un `dict` donde las llaves corresponden a nombres de usuario, y los valores el `Usuario` correspondiente.
  - `self.peliculas`: Generador que retorna la siguiente película disponible. Este atributo corresponde a una *property* con un *getter* definido .

Clase Pelicula

`class Pelicula:` No debes modificarlo
Clase que representa a las películas. Tiene los siguientes métodos y atributos importantes (entre otros):

- `def __init__(self, nombre, rankings, actores):` Inicializador de la clase con los siguientes atributos:
  - `self.nombre`: Un `str` con el nombre del video.
  - `self.rankings`: Un `dict` que contiene los puntajes de los distintos géneros del vídeo.
  - `self.actores`: Un `tuple` que contiene a los actores que participaron en la película.
- `def reproducir(self):` Este método simula la reproducción de una película.

Clase Usuario

`class Usuario:` Debes modificarlo
Clase que representa a los distintos usuarios que verán películas. En su mayoría ya viene implementada, y a continuación se describen métodos útiles de esta clase, pero deberás completar su método `ver_videos` de la forma especificada más adelante.

- `def __init__(self, nombre, preferencias, actor_prohibido):` No debes modificarlo
  Inicializador de la clase con los siguientes atributos:
  - `self.nombre`. Un `str` con el nombre del usuario.
  - `self.preferencias`: Un `dict` que contiene las preferencias de género de película del usuario.
  - `self.actor_prohibido`: Un `str` que es el nombre de un actor que el usuario no debe escuchar.
  - `self.listas_reproduccion`: Un `dict` con las listas de reproducción del usuario.
  - `self.afinidades`: Un `dict` con las distintas afinidades a las distintas películas.
  - `self.peliculas_favoritas`: Un `set` que contiene los nombres de las películas favoritas del `Usuario`. Implementado como el *getter* de una *property*.
- `def ver_videos(self, nombre_lista: str) -> tuple:` Debes modificarlo
  Generador que recibe el nombre de una `ListaReproduccion`. Deberás completarla en la **Parte III** de la actividad.
- `def calcular_afinidad(self, pelicula: Pelicula) -> float:` No debes modificarlo
  Calcula y guarda la afinidad de una película con el usuario.
- `def calificar_pelicula(self, pelicula) -> bool:` No debes modificarlo
  Retorna si a un usuario le gustó o no una película.

Parte I: Herramientas Básicas DCCabritas

En esta primera sección, deberás completar cuatro funciones que permiten a **DCCabritas** realizar varias de sus funcionalidades básicas. En el archivo `funciones.py` encontrarás las funciones que deberás modificar, encargadas de que el programa pueda crear las listas exitosamente, crear las *Watch Party* y *Recomendar Amigues*. **Las funciones en este archivo deben ser completadas sin usar ciclos `for` o `while`**:

- `def filtrar_prohibidos(iterar_peliculas: iter, actor_prohibido: str) -> filter:` **Debes modificarlo**
  Esta función recibe un iterador que itera sobre todas las películas, y debe retornar un iterador que itera sólo sobre aquellas que **NO** tienen al `actor_prohibido`. Para ello debes usar el método `filter` que aprendiste esta semana. Esta debe retornar el `filter` resultado.
- `def calcular_afinidades(catalogo_peliculas: zip, usuario: Usuario) -> map:` **Debes modificarlo**
  Esta función se encarga de calcular la afinidad del usuario `usuario` con cada una de las películas en `catalogo_peliculas`. Debe retornar un `map` que por cada elemento de `catalogo_peliculas`, retorna una `tuple` donde el primer valor es la `Pelicula` y el segundo su `afinidad`. Debes usar el método `calcular_afinidad` de la clase `Usuario` para calcular la afinidad de cada película.
- `def encontrar_peliculas_comunes(usuarios_watch_party: list) -> set:` **Debes modificarlo**
  Esta función recibe una lista de instancias de los `Usuario` seleccionados para la **Watch Party!**. Deberás retornar un `set` de las películas favoritas en común de los usuarios, es decir, la intersección de todos los conjuntos de películas favoritas. Deberás hacerlo con `reduce`.
- `def encontrar_usuario_mas_afin(usuario: Usuario, otros_usuarios: list) -> Usuario:` **Debes modificarlo**
  Esta función debe encontrar al usuario que tenga mayor compatibilidad con el usuario conectado. El argumento `usuario` recibe a este usuario, mientras que `otros_usuarios` es la lista de las otras personas inscritas en **DCCine**. Debe primero encontrar a todos los usuarios que tengan al mismo `actor_prohibido` que el `usuario`. Luego, sobre este subconjunto, debe encontrar al usuario con quien tenga mayor compatibilidad. Puedes ocupar tanto `map`, `filter` y/o `reduce`, según lo necesites. **Importante**: Para encontrar la **compatibilidad** entre dos usuarios, sumándolos. Es decir:

```
compatibilidad = usuario_A + usuario_B
```

Parte II: Iteradores y Generadores

Si bien crear **Watch Parties** puede ser muy entretenido, de nada sirve si no podemos reproducirlas. A continuación, deberás implementar los métodos y clases que permiten a **DCCabritas** iterar sobre películas y reproducirlas. En esta sección deberás trabajar en el archivo `lista_reproduccion.py`, y también completar un método en `usuario.py`.

En el archivo `lista_reproduccion.py` encontraras las clases `IterarLista` y `ListaReproduccion`, que deberás implementar de manera que `ListaReproduccion` sea una iterable. La idea detrás de `ListaReproduccion` es que contenga las películas favoritas de un usuario, y tiene un nombre. Al iterar sobre ella, retorna estas películas en orden descendiente con respecto a su afinidad con respecto al usuario:

Clase ListaReproduccion

`class ListaReproduccion`: Clase que simula una lista de reproducción.

- `def __init__(self, lista_videos):` **No debes modificarlo**
  Inicializador de la clase con los siguientes atributos:
  - `self.conjunto_videos`: Una `set` que contiene películas favoritas del usuario.
  - `self.usuario`: Una `str` que es el nombre del usuario.
  - `self.nombre`: Una `str` con el nombre de la `ListaReproduccion`.

- `def __iter__(self):` **Debes modificarlo**
  Debe crear y retornar una instancia de `IterarLista`. `IterarLista` debe recibir una **copia** de `self.conjunto_videos` [1]

- `def __str__(self):` **No debes modificarlo**
  Retorna el nombre de la lista.

Clase IterarLista

`class IterarLista:` **Debes modificarlo**
Esta clase se encarga de iterar sobre `ListaReproduccion`, que le entregará el `set self.conjunto_videos`. Para eso tiene definido los métodos `__iter__` y `__next__`:

- `def __init__(self, lista_videos):` **No debes modificarlo**
  Inicializador de la clase con los siguientes atributos:
  - `self.conjunto_videos`: Un `set` de tuplas en las que el primer valor corresponde a una `Pelicula`, mientras que el segundo es un `float` que representa la afinidad que tiene la `Pelicula` con el `Usuario`.

- `def __iter__(self):` **Debes modificarlo**
  Este método debe retornar un Iterador. Debes aplicar tus conocimientos de Iterables para saber cuál.

- `def __next__(self):` **Debes modificarlo**
  Retorna una `Pelicula`. Debe retornar la `Pelicula` con la mayor afinidad del `set self.conjunto_videos`, y luego eliminarla del conjunto. Una vez el conjunto esté vacío, debe levantar una excepción apropiada. Recuerda que cada película está representada en una tupla, donde el primer valor es la `Pelicula`, y el segundo la `afinidad`[2].

Parte III: Clase Usuario

En esta última parte, deberás completar un método de la clase `Usuario`, dándole la habilidad de ver las películas. En el archivo `usuario.py` encontrarás el método que deberás modificar, donde se simula el de que un usuario pueda reproducir una `ListaReproduccion`.

Deberás implementar un **generador** que permite que el `Usuario` pueda ver las películas e ir entregando *feedback* sobre si disfrutó o no la función. Esto lo harás completando el método `ver_videos` de `Usuario` para que cumpla con lo siguiente:

`def ver_videos(self, nombre_lista: str) -> tuple:` **Debes modificarlo**
Generador que recibe el nombre de una `ListaReproduccion`. Debe iterar sobre ella, y en cada iteración realizar las siguientes acciones:

- Reproducir la Película ocupando el método correspondiente. Recuerda que `ListaReproduccion` retorna una instancia de `Pelicula` en cada iteración.
- Calificar la película usando el método de `Usuario calificar_pelicula`. Este método retorna un `bool` indicando si le gusto la película.
- Hacer `yield` de una tupla. El primer elemento es la calificación que recibió la `Pelicula`. El segundo elemento es el atributo `rankings` de la `Pelicula`

[1] Puedes usar el método de los `set` `copy()` que retorna una copia de este.
[2] La función `max` y su argumento `key` son muy útiles en esté método.

Finalmente, debe imprimir un mensaje diciendo que se llegó al final de la `ListaReproduccion`.

Notas

- Recuerda que para las funciones `map`, `filter` y `reduce`, es necesario que entregues una función. Esta función puede ser un `lambda`, ¡pero también puedes definir una función auxiliar dentro de otra función!
- Siéntete libre de agregar nuevos `print()` en cualquier lugar de tu código para encontrar errores. Es una herramienta muy útil pero solo recuerda borrarlos antes de hacer el commit final para evitar confundir a tu corrector.
- Puedes modificar el parámetro `TIEMPO_REPRODUCCION` si deseas probar tu código más rápidamente.
- A pesar de que no debes modificar todos los archivos, es muy útil verlos igual. No solo te ayudará a entender que está pasando por atrás del programa, pero también, ¡ver código ajeno ayuda a aprender nuevas formas de programar!.

Objetivos

- Aplicar el conocimiento de iterables en un contexto de código similar a la realidad.
- Ser capaz de crear tus propios objetos iterables.
- Identificar y resolver problemas relacionados a iterables e iteradores.