Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación
IIC2233 — Programación Avanzada 2024-2

Publicación: 9 de Agosto de 2024
Actividad evaluada

Actividad 1
Estructuras de Datos

Entrega

- **Lugar:** Repositorio personal de GitHub — Carpeta: `Actividades/AC1`
- **Fecha máxima de entrega:** Jueves 22 de Agosto 23:59
- **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.**

Introducción

Comenzando un nuevo semestre y ya te aburres durante las ventanas. Así que has optado por desarrollar tu propia página de *streaming*. En la primera etapa, te propones desarrollar un programa que permita administrar diferentes consultas acerca de los animes.

Para su implementación deberás utilizar las estructuras *built-in* de Python con el objetivo de definir la función de cargado de los datos, así como las consultas que se utilizan.

Flujo del programa

Esta actividad consta de 2 partes. La primera referente a la carga de datos a partir de un archivo, preprocesar y retornarlos en un formato específico. Luego, la segunda parte es completar una serie de funciones para consultar la base de datos. Todas estas partes serán corregidas exclusivamente mediante el uso de *tests*.

Archivos

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

- Entregar Modificar `main.py`: En este archivo están las bases de funciones encargadas de consultar los datos. Al finalizar la actividad, debes asegurar que este archivo esté subido en el lugar de entrega correspondiente.
- No modificar `utilidades.py`: Contiene la definición de la *nametuple* a utilizar en la actividad.

- No modificar `data`: Carpeta que contiene diferentes archivos de texto plano. Cada archivo contiene diversos animes para cargar y consultar. Cada línea sigue el siguiente formato:

  `nombre,capitulos,puntaje,estreno,estudio,genero_1;...;genero_n`

  Donde `nombre`, `estudio` y `genero_1,...,genero_n` deben almacenarse como `str`, y `estreno` y `puntaje` como `int`.

- 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.

Cargar datos

Para poder leer los archivos deberás modificar las funciones presentes en el archivo `main.py`, sin embargo **no podrás hacer uso de clases para guardar esta información**. Las funciones son las siguientes:

- `def cargar_animes(ruta_archivo: str) -> list:`

  Esta función recibe la ruta a un archivo de texto que contiene todos los animes. El formato de cualquier archivo a leer será el informado en la sección anterior. El trabajo de esta función es abrir el archivo con *encoding* `"UTF-8"` y cargar todos los animes contenidos en dicho archivo, los cuales pueden ser 0, 1 o N animes. Debe retornar una lista de `namedtuples`, donde cada `namedtuple` contiene información de un Anime. Las `nametuples` de la lista deben estar en el mismo orden que estaban en el archivo y debes utilizar la `nametuple` entrega en el archivo `utilidades.py`.

  Por ejemplo, si el contenido del archivo es:

  ```
  Hunter x Hunter,62,9,1999,Nippon Animation,Acción;Aventura;Comedia;Shonen
  Sakura Card Captor,70,10,1998,Madhouse,Romance;Acción;Comedia;Shoujo
  ```

  La lista retornada se debe ver así:

```
[
    Anime(nombre="Hunter x Hunter", capitulos=62, puntaje=9,
          estreno=1999, estudio="Nippon Animation",
          generos={"Acción", "Aventura", "Comedia", "Shonen"}),

    Anime(nombre="Sakura Card Captor", capitulos=70, puntaje=10,
          estreno=1998, estudio="Madhouse",
          generos={"Romance", "Acción", "Comedia", "Shoujo"}),
]
```

  También debes asegurarte que los atributos de cada anime sean guardados como su tipo correspondiente, nombre como `str`, estudio como `str`, capítulos como `int`, puntaje como `int`, estreno como `int` y los géneros como una estructura que solo permita datos únicos y no repetidos.

  Ojo que los géneros están separados por un punto y coma (`;`), mientras que toda la demás información está separada por comas simples (`,`).

Consultas

La segunda parte de tu trabajo es implementar distintas consultas para poder extraer información relevante de la base de datos entregada. En específico tendrás que implementar las siguientes consultas:

- `def animes_por_estreno(animes: list) -> dict:`

  Esta función recibe una lista de animes (cada uno es una `namedtuple`). Con esta información, deberás organizar los animes según su estreno como un diccionario, donde cada llave es un año de estreno distinto y su valor es una lista con el nombre de los animes que pertenecen a ese año de estreno, un ejemplo sería:

```
{
  2010: [anime_x,..., anime_y],
  1990: [anime_r,..., anime_s],
  ...,
  2019: [anime_u,..., anime_v]
}
```

  Es importante mencionar que los años deben ser tratados en todo momento como `int`. Y que el orden de los años **no influye en la respuesta**. Como tampoco el orden de los animes en cada lista.

- `def descartar_animes(generos_descartados: set, animes: list) -> list:`

  Esta función recibe un conjunto de géneros y una lista de animes (cada uno es una `namedtuple`). Se debe buscar todos los animes **que no contengan** ninguno de los géneros indicados en el argumento `generos_descartados`. Luego, se debe retornar una lista con el nombre de los `animes` sin descartar.

  A modo de ejemplo, si la función se ejecutara del siguiente modo

```
descartar_animes(
    {"Comedia", "Horror"},
    [
        Anime("Hunter x Hunter", 62, 9, 1999, "Nippon Animation",
          {"Acción", "Aventura", "Comedia", "Shonen"}),

        Anime("Sakura Card Captor", 70, 10, 1998, "Madhouse",
          {"Romance", "Acción", "Comedia", "Shoujo"})
    ]
)
```

  El resultado de esta función será una lista vacía porque ambos animes poseen el género `"Comedia"` que se desea descartar.

- `def resumen_animes_por_ver(*animes: Anime) -> dict:`

  Esta función recibirá una cantidad desconocida de Animes. Dado que esta función puede recibir desde 0 hasta una cantidad arbitraria y desconocida de animes, se utilizará el operador $*$. Luego, el objetivo de esta función es retornar un diccionario con el resumen de todos los animes. El diccionario debe tener las siguientes llaves y para sus respectivos valores debes calcular:

  - `"puntaje promedio"`: Puntaje promedio entre todos los animes. Debe ser un `float` redondeado al primer decimal usando `round`. En caso de no recibir anime, este valor es 0.

- `"capitulos total"`: Cantidad total de capítulos que corresponde a la suma de los capítulos de cada anime. Debe ser un `int`. En caso de no recibir anime, este valor es 0.
- `"generos"`: Conjunto único de todos los géneros que serán vistos entre los diversos animes. En caso de no recibir anime, este conjunto deberá ser un `set` vacío.

A modo de ejemplo, si la función se ejecutara del siguiente modo:

```
resumen_animes_por_ver(
    Anime("Hunter x Hunter", 62, 9, 1999, "Nippon Animation",
          {"Acción", "Aventura", "Comedia", "Shonen"}),
    Anime("Sakura Card Captor", 70, 10, 1998, "Madhouse",
          {"Romance", "Acción", "Comedia", "Shoujo"})
    )
```

El diccionario a retornar será:

```
{
  "puntaje promedio": 9.5,
  "capitulos total": 132,
  "generos": {"Romance", "Acción", "Comedia", "Shoujo", "Aventura", "Shonen"}
}
```

- `def estudios_con_genero(genero: str, **estudios: list) -> int:`

Esta función recibirá el nombre de un género y una cantidad desconocida de estudios en forma de *keywords*. Donde cada estudio estará asociado a la lista de **Anime** que pertenecen a dicho estudio. Dado que esta función puede recibir desde 0 hasta una cantidad arbitraria y desconocida de estudios, se utilizará el operador `**`. De este modo, cada llave de este operador será un estudio, mientras que su valor corresponderá a una lista de Anime.

Esta función identificará cuales son los estudios que contengan al menos 1 anime con el género indicado. Luego se debe retornar la lista con dichos estudios. Para esto último, deberás usar la llave de cada elemento del operador `**`

A modo de ejemplo, si la función se ejecutara del siguiente modo:

```
estudios_con_genero(
    "Shonen",
    Nippon_Animation = [
            Anime("Hunter x Hunter", 62, 9, 1999, "Nippon Animation",
                  {"Acción", "Aventura", "Comedia", "Shonen"})
            ],
    Madhouse = [
            Anime("Sakura Card Captor", 70, 10, 1998, "Madhouse",
                  {"Romance", "Acción", "Comedia", "Shoujo"})
            ]
    )
```

Esta función retornará `["Nippon_Animation"]` porque es el único estudio que tiene un anime con el género `"Shonen"`.

Notas

- **No puedes hacer *import* de otras librerías externas a las entregadas en el archivo.**
- Recuerda que la ubicación de tu entrega es en **tu repositorio de Git**. En la rama (*branch*) por defecto del repositorio: `main`.
- Puedes implementar cualquier función en el orden deseado.
- 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.
- Siéntete libre de agregar nuevos `print` en cualquier lugar para revisar objetos, modificaciones, etc... Es una herramienta muy útil para encontrar errores. Y si aparece un error inesperado, ¡léelo! Intenta interpretarlo.

Objetivos de la actividad

- Implementar distintos tipos de estructuras *built-in* según corresponda.
- Reconocer las ventajas entre las estructuras *built-in* de Python.
- Utilizar elementos de la *standard library* de Python.

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. Para ejecutar estos *tests*, **primero debes posicionar tu terminal/consola en la carpeta de la actividad (`Actividades/AC1`)**. Luego, desde esta misma, debes escribir el siguiente comando para ejecutar todos los *tests* de la actividad:

- `python3 -m unittest discover tests_publicos -v -b`

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

- `python3 -m unittest -v -b tests_publicos.test_cargar_animes`
  Para ejecutar sólo el subconjunto de *tests* relacionado a la carga de datos.
- `python3 -m unittest -v -b tests_publicos.test_consultas_animes_por_estreno`
  Para ejecutar sólo el subconjunto de *tests* relacionado a la consulta `animes_por_estreno`.
- `python3 -m unittest -v -b tests_publicos.test_consultas_descartar_animes`
  Para ejecutar sólo el subconjunto de *tests* relacionado a la consulta `descartar_animes`.
- `python3 -m unittest -v -b tests_publicos.tests_consultas_resumen_anime`
  Para ejecutar sólo el subconjunto de *tests* relacionado a la consulta `resumen_animes_por_ver`.
- `python3 -m unittest -v -b tests_publicos.test_consultas_estudios_genero`
  Para ejecutar sólo el subconjunto de *tests* relacionado a la consulta `estudios_con_genero`.

Importante: recuerda que si `python3` no funciona, probar con el comando específico de tu computador. Este puede ser `py`, `python`, `py3` o `python3.11`.