Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación
IIC2233 — Programación Avanzada 0b11111100100-0b10

0b1100 de 0b1011 de 0b11111100100
Actividad Sumativa

Actividad Sumativa ~~04~~ 0b100

I/O y Serialización

Entrega

- **Lugar:** En su repositorio privado de GitHub, en la **carpeta** `Actividades/AS04`
- **Hora del *push*:** 16:50

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

Dado que estamos en tiempos de cuarentena, con tus amigues has decidido desarrollar una aplicación *totalmente original* para ver películas, series y anime. Lamentablemente, el malvado *hacker* Dr. Pinto, reconocido mundialmente por su audacia, ha infectado el *back-end* de tu aplicación estrella, **DCCine**.

Figura 1: Logo de DCCine.

El ataque del Dr. Pinto ha dañado los archivos de **DCCine**, por lo que tu tarea es reparar los archivos dañados, lo que deberás hacer en tres etapas, utilizando tus conocimientos de **I/O** y **serialización**: **pickle**, ***bytes*** y finalmente **json**. Es importante recalcar que las reparaciones son **independientes** unas de otras, es decir, las puedes resolver por separado. Además se te entregará una interfaz gráfica ya implementada para ayudarte a probar el código.

Flujo del programa

El programa está dividido en tres secciones diferentes e independientes, las cuales tendrás que completar. Cada una de estas partes aportará información a a la aplicación **DCCine**. Además, se te entrega una interfaz gráfica que podrás utilizar para ir revisando las distintas secciones que hayas desarrollado. Como se describió anteriormente, la parte sobre **pickle** te permitirá reparar las claves de los usuarios, la parte ***bytes*** te permitirá ver las carátulas de las películas favoritas de las distintas cuentas, y la parte **JSON** te entregará la información correspondiente a cada una de las películas o series. Recuerda que estas partes son independientes una de la otra, pero si completas las tres, podrás ver la interfaz entregada funcionando en su totalidad.

Las partes las puedes completar en el orden que prefieras, ya que trabajan de forma separada en la Interfaz. Sin embargo, te recomendamos hacerlas en este orden:

- Completar Pickle y obtener los usuarios y claves
- Completar *Bytes* y arreglar las imágenes.
- Completar JSON y obtener la información de las películas.

Ya que así están ordenadas las ventanas.

Pickle

Todo lo que necesitas para desarrollar esta sección, se encuentra en la carpeta `Actividad pickle/`.

El malvado Dr. Pinto logró hackear a algunos usuarios, porque sus claves no estaban almacenadas de forma segura. Para mejorar la seguridad de la plataforma, deberás revisar una base de datos, y guardar una nueva base de datos que tenga todas las claves encriptadas. Además, para que los usuarios sean capaces de verificar y ver sus contraseñas, cuando el programa consulte la base de datos, se deben desencriptar las contraseñas.

En esta sección, deberás completar los métodos y las funciones en el archivo `pickle_actividad.py`, utilizando la biblioteca de serialización `pickle`. Para probar los resultados puedes ejecutar el archivo `pickle_actividad.py` que ya viene con un *main* implementado. Esta parte de la actividad está subdividida en dos partes, correspondientes a la **carga** y el **guardado** de la lista con los usuarios.

Primero hay que hacer la parte de **carga**, en esta etapa se encuentra la función `cargar_instancia`. En esta funcion se va a deserializar con `pickle` el archivo `info_personas.bin`, en el cual se encuentra una lista poblada con instancias de la clase `Usuario`. En esta lista, hay algunos usuarios que tienen la clave encriptada, y otros que no la tienen encriptada. Al cargar este archivo, se deben desencriptar todas las claves que estén encriptadas.

Luego está la parte de **guardado** en donde se encuentra la función `guardar_instancia`. Aquí se va a serializar nuevamente la lista, en el archivo `archivo_encriptado.bin`, pero esta vez se deben encriptar todas las claves de los usuarios.

A continuación, se presenta la clase `Usuario` y las funciones que se te entregan hechas o que debes desarrollar Todas estas se encuentran en el archivo `pickle_actividad.py`.

Para que esta parte se pueda realizar de forma correcta se requiere hacer la parte de carga primero y luego la parte de guardado.

- Clases:
  - `class Usuario`: Es la clase que representa a un usuario de la plataforma.
    - `def __init__(self, nombre, peliculas, indices, password)`: Recibe información del usuario y la almacena como atributos de la instancia. **No se debe modificar**
    - `def __setstate__(self, state)`: Debe desencriptar la clave del usuario para conseguir su contraseña original si y solo si el atributo `clave_encriptada` es `True` . Si se cumple este caso deberás cambiar el atributo `password` con la función `desencriptar(password)` y cambiar `clave_encriptada` a `False`.
    - `def __getstate__(self)`: Debe encriptar la clave del usuario y debe retornar un diccionario con los atributos de la clase pero ahora con el atributo `password` encriptado y con el valor de `clave_encriptada` igual a `True`.
- Etapa de carga:
  - `def desencriptar(password)`: Recibe una clave encriptada, y retorna la clave desencriptada. **No se debe modificar**
  - `def cargar_instancia(ruta)`: Recibe la ruta que corresponde a la ubicación del archivo `info_personas.bin` (archivo con una lista de usuarios serializados) y retorna una lista con las instancias de la clase `Usuario`. Todas las instancias deben tener la clave desencriptada y el atributo `clave_encriptada` igual a `False`.
- Etapa de guardado:
  - `def encriptar(password)`: Recibe una clave desencriptada, y retorna la clave encriptada. **No se debe modificar**
  - `def guardar_instancia(ruta, objeto_lista_personas)`: Recibe la ruta que corresponde a la ubicación de un archivo vació llamado `archivo_encriptado.bin` y una lista con la información de todos los usuarios y con las claves desencriptadas. La función debe serializar la lista y guardarla en el archivo.

Bytes

Este ítem se encuentra en la carpeta `Actividad bytes/`, y tiene dos subdirectorios, `caratulas/` y `corruptas/`, que **NO** debes modificar, ya que traen los archivos con los que deberás trabajar.

Parte del daño realizado a la base de datos de `DCCine` ocurrió a los pósteres de las películas y series, cuyos *bytes* han sido corrompidos. Para que puedas ver sus carátulas, debes tomar las imágenes dañadas y repararlas, esto lo harás completando la función `reparar_imagen` que se encuentra en el archivo `bytes.py`. Una vez completada la función, solo debes ejecutar el archivo `bytes.py` y revisar si puedes ver las carátulas en la carpeta `caratulas/`. Las carátulas no son de buena calidad (para que la actividad no fuera demasiado pesada), así que no te preocupes por eso.

- `def reparar_imagen(ruta_entrada, ruta_salida)`: Esta función, que **debes completar**, recibe la ruta de la imagen a reparar, lee el archivo como *bytes* y mediante un algoritmo los modifica de

modo que la imagen pueda mostrarse correctamente. Para terminar, escribe los *bytes* corregidos en `ruta_salida`, la que ya incluye el nombre y formato. El algoritmo recorre **segmentos de 32 *bytes* contiguos** y para cada uno se sabe que:

1. Si el primer *byte* es 1, los primeros 16 *bytes* han sido invertidos (desde este *byte* igual a 1, hasta la mitad del *chunk*), por lo tanto, si la secuencia de los primeros 16 *bytes* es `1 0 5 ... 11 9 8`, ahora debe ser `8 9 11 ... 5 0 1`. Si el primer *byte* no es 1, entonces el orden de la secuencia no debe modificarse.
2. Los últimos 16 *bytes* **NUNCA** corresponden a la secuencia original, se hayan invertido o no los primeros 16 *bytes*, por lo que debes ignorarlos.

- `def reparar_imagenes(carpeta_entrada, carpeta_salida)`: Esta función recibe la carpeta donde se encuentran las carátulas corruptas y la carpeta donde se espera guardar las carátulas reparadas. Llama a `repara_imagen` para cada archivo en `carpeta_entrada`. **No debes modificarla**.

JSON

Todo lo que necesitas para desarrollar esta sección, se encuentra en la carpeta `Actividad json`.

El malvado Dr. Pinto no solo hackeó los usuarios, sino que con el fin de que nadie pueda acceder a sus películas favoritas, encriptó toda la información. Es por esto que en esta sección, deberás deserializar y desencriptar la información contenida en el archivo `peliculas.json` para obtener los nombres y datos de todas las películas del **DCCine**. Esto lo deberás hacer en el archivo `peliculas.py`.

El archivo `peliculas.json` contiene un diccionario donde las llaves corresponden a los **nombres** de las películas, y cada valor corresponde a una **lista** en donde se contiene el **número**, **director**, **duración** y **puntuación** correspondiente a cada película. El diccionario contenido en el archivo tiene la información de cada película en el siguiente formato:

```
diccionario[nombre_película] = [número_película, director, duración, puntuación]
```

Tu tarea es completar el archivo `peliculas.py`, el cual contiene las siguientes funciones y clases:

- Clases:
  - `class Pelicula`: Esta clase almacena los atributos de cada película, y será utilizada para imprimir la información relacionada a cada una, por lo cual viene implementada y **no la debes modificar**.
    - `def __init__(self, nombre, director, duracion, puntuacion)`: Constructor de la clase. **No debes modificarlo**.
    - `def __repr__(self)`: Entrega una representación de cada película. **No debes modificarlo**.
- Funciones:
  - `def desencriptar(string)`: Decodifica el *string* encriptado y lo retorna. Esta función deberás utilizarla para desencriptar cada `str` contenido en el diccionario del archivo `peliculas.json`. **No debes modificarla**.
  - `def cargar_peliculas(ruta)`: Recibe la ruta del archivo `peliculas.json` y **debes completarla**. Debes deserializar el diccionario contenido en el archivo y **personalizar la deserialización**, para desencriptar los *strings* contenidos en el diccionario. Para esto, deberás utilizar como

`object_hook` la función `desencriptado`. Esta función debe retornar el diccionario deserializado.

- `def desencriptado(diccionario)`: Esta función permite personalizar la deserialización, y **debes completarla**. Recibe un diccionario del archivo `peliculas.json`, el cual deberás desencriptar los *string* que representan el **nombre de la película**, **director**, **duración** y **puntuación** mediante la función `desencriptar(string)`. El **número** corresponde a un índice asociado a cada película y tiene formato `int`, por lo que no se encuentra encriptado. Luego debes retornar el diccionario desencriptado.

Para probar esta parte, puedes correr el archivo `peliculas.py`. Si todo está bien implementado, se imprimirá en la pantalla la información de las películas, en el siguiente formato:

```
"NOMBRE         | DIRECTOR           | DURACION    | PUNTUACION    |"
"Forrest Gump   | Robert Zemeckis    | 142 min     | 9.4           |"
```

Interfaz Gráfica

Como fue mencionado antes, se te entrega una interfaz gráfica para ayudarte a correr tu código. Para correrla debes ingresar a la carpeta **Interfaz Grafica/** y correr el archivo `main.py`, donde podrás interactuar con tres ventanas:

Ventana Inicial

Esta pide un **usuario** y una **clave**, las cuales son obtenidas resolviendo la sección de **Pickle**. Si todavía no has resuelto esta parte puedes probar con la llave maestra:

- **Usuario**: `cruz`
- **Clave**: `nosoyunrobot`

Esta se debería ver de la forma:

Figura 2: Ventana Inicial

Ventana de Películas

Aquí se muestran las carátulas de las películas que tienen guardadas les usuarios. Desde esta, hacer **clic** en las carátulas te llevará a la **Ventana Final** donde se mostrará la información de cada película.

La interfaz busca la carpeta **caratulas/** de la sección de ***bytes***, y si esta está vacía la rellena con una sorpresa. Dependiendo si completaste la sección de ***bytes*** correctamente la interfaz se verá así:

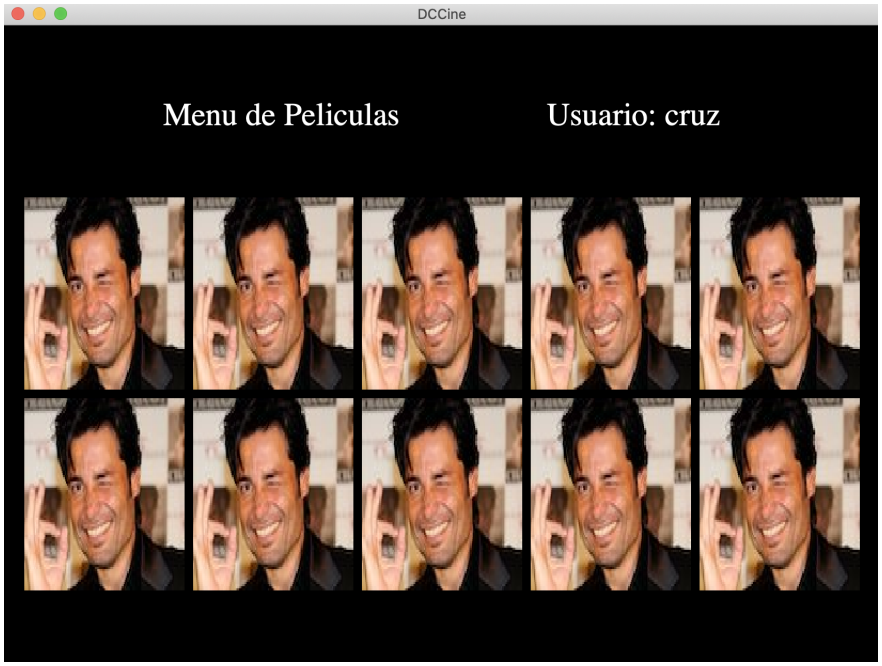

Figura 3: Ventana de Películas sin completar la parte de *bytes*

Figura 4: Ventana de Películas con la parte de *bytes* realizada

Ventana Final

Esta muestra la información de cada película, que se obtiene completando la parte de **JSON**.

La interfaz ocupa el resultado de la función `cargar_peliculas`, por lo que es importante que esta retorne la información en el formato correcto. Dependiendo si se ha completado la sección la ventana se verá de la forma:

Figura 5: Ventana Final sin haber completado la parte de JSON.

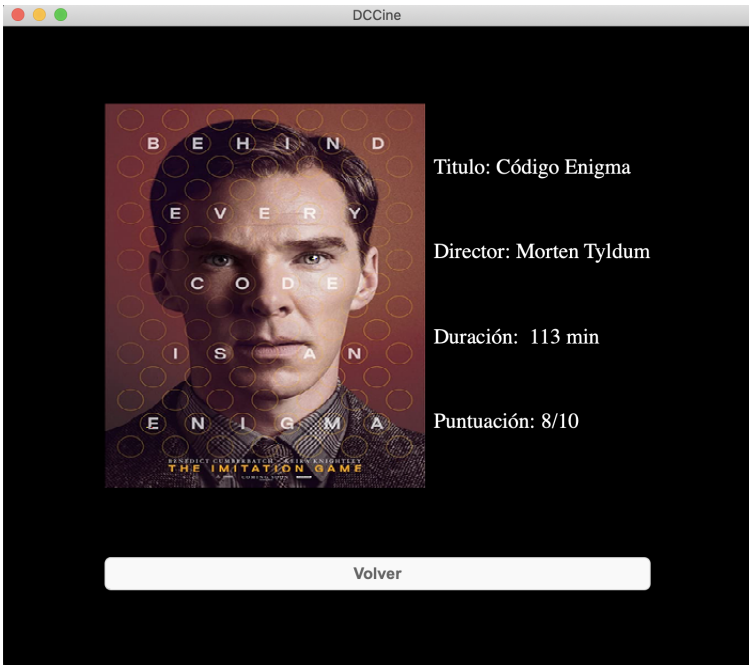

Figura 6: Ventana Final habiendo completado la parte de JSON.

Nota: Se entrega con la interfaz un archivo `parametros.py` donde se puede regular el tamaño de la ventana por si surge algún problema de resolución.

Requerimientos

- (2.50 pts) Parte Pickle
  - (0.75 pts) Completar `__setstate__` de la clase Usuario.
  - (0.75 pts) Completar `__getstate__` de la clase Usuario.
  - (0.50 pts) Completar función `cargar_instancia`.
  - (0.50 pts) Completar función `guardar_instancia`.
- (2.50 pts) Parte *Bytes*
  - (0.75 pts) Carga el archivo entregado en la ruta como *bytes*.
  - (1.00 pts) Recorre el archivo entregado en segmentos del largo pedido y lo procesa correctamente.
  - (0.75 pts) Guarda el archivo con el nombre y formato pedidos.
- (1.00 pts) Parte JSON
  - (0.50 pts) Completar `cargar_peliculas`.
  - (0.50 pts) Completar `desencriptado`.