Trabajo Práctico Nro. 2

[75.40] Algoritmos y Programación I

Jugársela

1er cuatrimestre 2023

.UBAfiuba
FACULTAD DE INGENIERÍA

Índice

Estructuras utilizadas

Respecto a las estructuras que conforman este programa, a simple vista se puede notar cuáles son las que juegan un rol principal:

1) **informacion_api** -> Lista de diccionarios. Es la estructura troncal respecto a la API, puesto que almacena todos los datos disponibles sobre la Liga Profesional Argentina hasta la fecha, separados por categorías: temporadas, equipos y fixtures.

    a. **temporadas** -> Diccionario. Esta estructura almacena, como claves, los años en los que se efectuaron los partidos de la Liga. El valor para cada uno de estos varía según la temporada, puesto que, al haberse efectuado cambios en el sistema de juego, nos vimos en la obligación de almacenar estos datos de manera diferente. Tal que el diccionario resulta:

"2015-2019": {
"$Año_1$": [($Equipo_1$, $Puntos_1$), ($Equipo_2$, $Puntos_2$), ($Equipo_3$, $Puntos_3$) ...],
"$Año_2$": [($Equipo_1$, $Puntos_1$), ($Equipo_2$, $Puntos_2$), ($Equipo_3$, $Puntos_3$) ...],
"$Año_3$": [($Equipo_1$, $Puntos_1$), ($Equipo_2$, $Puntos_2$), ($Equipo_3$, $Puntos_3$) ...],
. . . }

El valor para cada año es una lista de tuplas, con las posiciones finales de cada equipo y su puntaje; empezando con el primer lugar y terminando con el último.

"2020": {
"$Grupo_1$": [($Equipo_1$, $Puntos_1$), ($Equipo_2$, $Puntos_2$), ($Equipo_3$, $Puntos_3$) ...],
"$Grupo_2$": [($Equipo_1$, $Puntos_1$), ($Equipo_2$, $Puntos_2$), ($Equipo_3$, $Puntos_3$) ...],
"$Grupo_3$": [($Equipo_1$, $Puntos_1$), ($Equipo_2$, $Puntos_2$), ($Equipo_3$, $Puntos_3$) ...],
. . . }

Para la temporada 2020 el valor será otro diccionario, cuyas claves contendrán el nombre del grupo y su valor una lista de tuplas, con las posiciones finales de cada equipo y su puntaje; empezando con el primer lugar y terminando con el último. Es importante aclarar que cada grupo tendrá su ranking individual.

"2021-2023": {
"Primera Fase": [($Equipo_1$, $Puntos_1$), ($Equipo_2$, $Puntos_2$), ($Equipo_3$, $Puntos_3$) ...],
"Segunda Fase": [($Equipo_1$, $Puntos_1$), ($Equipo_2$, $Puntos_2$), ($Equipo_3$, $Puntos_3$) ...]
. . . }

Para las temporadas entre 2021 y 2023 el valor será otro diccionario con dos claves: "Primera Fase" y "Segunda Fase". Cada una tiene su propio ranking, por eso la distinción. Los valores para cada fase serán una lista de tuplas con las posiciones finales de cada equipo y su puntaje; empezando con el primer lugar y terminando con el último.

b. **equipos** -> Diccionario. Esta estructura almacena como claves los nombres de todos los equipos que alguna vez participaron en la liga (desde el primer año, hasta el último. Previo a generar una clave, verifica que esta no esté. De ese modo se garantiza que el equipo no se repita). El valor para cada clave es otro diccionario:

"Nombre Equipo": {
"id": Número de id (str),
"escudo": Imagen png del escudo (str),
"estadio": Nombre del estadio (str),
"dirección": Dirección del estadio (str),
"ciudad": Ciudad donde está el estadio (str),
"capacidad": Capacidad de espectadores (str),
"superficie": Tipo de suelo del estadio (str),
"foto": Imagen png del estadio (str),
"plantel": [$jugador_1$, $jugador_2$, $jugador_3$, ..., $jugador_n$] (list),
"estadísticas": {
"Intervalo de $tiempo_1$": Cantidad de goles,
"Intervalo de $tiempo_2$": Cantidad de goles,
"Intervalo de $tiempo_3$": Cantidad de goles,
... }

*Cabe aclarar que, si el equipo **NO PARTICIPA** en la temporada 2023, el mismo VA A TENER VALOR **"NONE"** EN LA CLAVE "PLANTEL" y "ESTADISTICAS", puesto que únicamente se debe mostrar el plantel/las estadísticas de los equipos que juegan la temporada 2023.*

c. **fixtures** -> Esta estructura almacena como claves los id de todos los partidos que AÚN NO SE JUGARON (o sea, desde la fecha en la que se corre el programa en adelante). El valor para cada fixture es otro diccionario:

"ID del Fixture": {
"fecha": Fecha en la que se juega el partido (año, mes, día) (tuple),
"local": Nombre del equipo que juega de local (str),
"visitante": Nombre del equipo que juega de visitante (str)}

2) **transacciones_listado** -> Lista de listas. Esta estructura es una lista maestra que surge al leer el archivo "transacciones.csv" y volcar toda su información dentro de otras listas. Sirve para llevar un registro de todas las transacciones que se realizaron y se van realizando. Una vez se finaliza el programa, los datos de esta lista reescribirán el archivo anteriormente mencionado.

transacciones_listado: list = [
[“$mail_1$”, “$fecha_1$”, “tipo de transacción$_1$”, “$importe_1$”],
[“$mail_2$”, “$fecha_2$”, “tipo de transacción$_2$”, “$importe_2$”],
[“$mail_3$”, “$fecha_3$”, “tipo de transacción$_3$”, “$importe_3$”],
…]

3) **usuarios_diccionario** -> Diccionario. Al igual que la estructura de "transacciones", los datos de "usuarios" se definen a partir de un archivo; en este caso, "usuarios.csv". Sirve para relevar los datos de los usuarios previamente registrados en el programa. Esta información es la siguiente:

usuarios_diccionario: dict = {
“$email_1$”: [“$usuario_1$”, “$contraseña_1$”, dinero apostado$_1$, “fecha última $apuesta_1$”, $dinero_1$],
“$email_2$”: [“$usuario_2$”, “$contraseña_2$”, dinero apostado$_2$, “fecha última $apuesta_2$”, $dinero_2$],
“$email_3$”: [“$usuario_3$”, “$contraseña_3$”, dinero apostado$_3$, “fecha última $apuesta_3$”, $dinero_3$],
… }

Flujo del programa

Al inicio del programa se realizan todas las solicitudes a la API. Esta decisión la tomamos con la idea de que haya una pantalla de carga inicial y que luego el código corra usando la información en memoria. Debido a que los llamados a la API están limitados a un máximo de 10 (diez) por minuto, nos vimos forzados a aplicar un "frezee" de 6.0 segundos con la librería de Python "time" para que no haya inconvenientes al respecto; esto, sumado al hecho de que la interacción con la API de por sí requiere de tiempo para efectuarse, genera que al correr el programa el usuario deba esperar entre dos y tres minutos. A raíz de esto se agregó una pantalla de carga con un porcentaje, indicando a cada momento cuál es el estado de la solicitud de datos. La solicitud de información a la API ocurre en un orden específico, puesto que para extraer ciertos datos se requieren primero de otros. Se empieza pidiendo los años en los que se realizaron los partidos de la Liga Profesional Argentina; ya teniendo los años, se procede con las posiciones (rankings) finales de todos los equipos participantes y los puntajes de cada uno; nuevamente, para cada temporada, se pide la información de todos los equipo y si estos forman parte del lineup 2023, con sus id se solicitan sus planteles y las estadísticas generales de goles por minuto en la temporada; por último, se pide el fixture de todos los partidos que se jugaron y se jugaran este año (si la fecha de algún partido es posterior a la del día en el que se corre el programa, este no será agregado a la base de datos). La única comunicación que no se realiza al inicio es la de predicciones, debido a las limitaciones de llamados por día, no pudimos implementar cada dato a la variable información_api. Una vez finalizado esto, a través de las funciones "usuarios_csv_to_diccionario" y "transacciones_csv_to_listado", el programa vuelca la información de los archivos "usuarios.csv" y "transacciones.csv" en las variables "usuarios_diccionario" y "transacciones_listado" respectivamente. De este modo se puede ingresar, validar e iterar los datos presentes en ambos archivos.

Hecho todo lo detallado hasta ahora, se le proporcionará tres opciones al usuario:

1) **Iniciar sesión:** Solicita al usuario ingresar mail y contraseña, para luego verificar si ambos datos se encuentran en "usuarios_diccionario". De ser así, se continua con el programa.

2) **Registrarse:** Solicita al usuario registrar sus datos (mail, usuario, contraseña), con el método hash de la librería passlib.hash se encripta la contraseña. Los datos se almacenan en "usuarios_diccionario", siendo el mail la clave y el valor una lista con el resto de la información; cabe aclarar que tanto los datos monetarios (dinero apostado y dinero en cuenta) y la fecha de la última apuesta, se toman como 0 (cero). En caso de ingresar datos ya existentes, el programa se lo notifica al usuario. Tras esto, se vuelve al menú de inicio y se le solicita al usuario ingresar sesión.

3) **Salir:** Sale del programa.

Se procede con el menú principal, compuesto con las siguientes opciones:

1) **Consultar plantel:** Itera sobre la base de datos "información_api", tomando el diccionario de temporadas (informacion_api[0]) e imprimiendo los equipos participantes. Luego, se le solicita al usuario ingresar el nombre de uno de los equipos en pantalla; se valida que esté bien escrito y que sea parte de la temporada 2023. Hecho esto, se imprime el plantel.

2) **Tabla de posiciones:** Se le solicita al usuario ingresar un año entre 2015-2023 y se valida el input. Luego se imprime el ranking de posiciones. Dependiendo la temporada, se itera de manera diferente debido a los cambios en el sistema de juego que hubo tras el año 2020.

3) **Información de equipos:** Se le solicita al usuario ingresar el nombre de un equipo; se valida que esté se encuentre en la base de datos. Se itera "equipos" (informacion_api[1]) y se imprime la información disponible en la API respecto a su estadio, una foto de este y una imagen del logo del equipo.

4) **Estadística de goles 2023:** Se le solicita al usuario ingresar el nombre de un equipo; se valida que esté bien escrito y que sea parte de la temporada 2023. Se itera "equipos" (informacion_api[1]) y con un ciclo "for" se extrae la información de goles por minutos. Se agregan estos a dos listas diferentes ("minutos" y "goles"), se aplica el método "bar()" de la librería "matplotlib" y se muestra en una pantalla emergente el gráfico de goles por minuto realizado.

5) **Cargar dinero a la cuenta:** Se le solicita al usuario ingresar un monto equivalente a la cantidad de dinero que va a ingresar en su cuenta. Se valida que el input no sea un valor igual o menor a 0 (cero); de serlo, debe intentarlo nuevamente. Ya con el valor determinado, en "usuarios_diccionario" se suma dicho número a la cantidad de dinero en cuenta y se crea una nueva transacción con los datos personales de la cuenta, la fecha, el tipo de transacción (en este caso, "Deposita") y el monto ingresado. Por último, se le comunica al usuario cuánto dinero tiene disponible al momento de realizar la función.

6) **Usuario que más dinero apostó:** Con un ciclo "for" se itera "usuarios_diccionario" comparando el valor que se encuentra en la posición dos (número que equivale al dinero apostado por el usuario hasta la fecha) y se compara con el anterior, guardando en una variable el nombre de usuario de la persona que mayor cantidad de dinero haya apostado. Una vez terminado el ciclo, junto con una leyenda se imprime dicho nombre.

7) **Usuario que más veces ganó:** Similar al punto anterior, se itera con un ciclo "for" la variable "transacciones_listado" y si el tipo de transacción es "Gana" se agrega el nombre del usuario como clave de un diccionario "cantidad_apuestas" y su valor pasa a ser un uno. De ya estar el usuario en dicho diccionario, se le suma un uno al valor

anterior. De esta forma se genera un contador con el cual tras iterar nuevamente con un ciclo "for", se determina qué clave (o sea, qué usuario) posee el valor más grande (cantidad de veces ganadas). Hecho esto, se imprime junto con una leyenda el nombre del usuario y la cantidad de veces.

8) **Realizar apuesta:** En primera instancia se le muestra al usuario una lista de todos los equipos que participan en el torneo y se le solicita al mismo que elija uno de ellos. Una vez elegido se imprimen todos los partidos que va a jugar ese equipo, contra quiénes juega y las fechas. Se le indica al usuario que escoja uno de los partidos por el que quiere apostar.
    1. En caso de no haber disponible ningún partido de ese equipo, se le notifica al usuario de esto mismo y se vuelve a empezar.
    2. Se le recuerda al usuario cuál es el equipo local, cuál el visitante y se le pide que apueste por uno de los tres resultados posibles del partido: Ganador(V), Empate, Ganador(L).
    3. Se le pide al usuario que ingrese un monto de dinero de su cuenta para apostar. En caso de no disponer del dinero en la cuenta, se le ofrecen dos opciones:
        1. Cambiar el monto a apostar.
        2. Cancelar la operación. De esa forma puede ir a cargar dinero.

    d. Se simula el partido, se escoge aleatoriamente un resultado y un ratio de pago. Dependiendo de la predicción que ofrece la API el usuario gana más o menos dinero. Siendo estas las distintas posibilidades:

        1. Si acierta el resultado…

            1. …y **es diferente** al que la API predijo: Se le devuelve la cantidad apostada + la cantidad apostada * ratio.
            2. …y **es igual** al que la API predijo: Se le devuelve la cantidad apostada + la cantidad apostado * ratio/10.
            3. …y **es un empate**: Se le devuelve la cantidad apostada + la cantidad apostada * ratio/2.

        ii. Si falla el resultado:

            1. Pierde el dinero apostado.

    e. Por último, se guardan las pérdidas/ganancias en los archivos de usuario y transacciones.

9) **Cerrar sesión:** Vuelve al menú de inicio de sesión.

Secciones importantes del código

Las partes que creemos como grupo tienen mayor relevancia en el código, son la interacción con la API al comienzo del programa, la apertura y cierre de los archivos “usuarios.csv” y “transacciones.csv”, el inicio de sesión y tanto el juego de apuestas como el pago que se realiza tras las mismas. Todas estas secciones se encuentran detalladas en los puntos anteriores de este informe.

Conclusión

En general la lógica del programa no resultó ser de una dificultad extraordinaria para ninguno de los tres. Hubo retos, sí, pero nada que no se pueda resolver tras un poco de trabajo. La participación de los integrantes del grupo fue justa y equilibrada. Consideramos que existió un buen ambiente y predisposición a la hora de trabajar en equipo a pesar de no haber tenido trato alguno antes del trabajo práctico.

Aún así, podemos destacar un par de hechos que sí supusieron conflictos menores:

1) Nos encontramos a ciegas ante la decisión grupal de cómo repartirse de manera equitativa las partes del código, lo cual retrasó nuestro inicio en relación con la fecha de presentación original de trabajo. Tras un meeting virtual, pudimos establecer sin problemas ciertos planteos subjetivos y dudas generales respecto al modo de participación.
2) Similar al punto anterior, uno de los conflictos fue el uso a paso firme de GIT y GitHub. Al no tener experiencia con los mismos, nos veíamos rápidamente dubitativos con su uso y temerosos de cometer algún error que afectará al proyecto o a los compañeros.

Por último, la interacción con la API fue un problema desde el primer momento. No sólo debido a la limitación de intentos por día, rasgo que impedía por largos periodos de tiempo el avance del equipo; sino también por la máxima cantidad de llamados por minuto (diez) que ofrece la API a los usuarios con plan gratuito. En una primera instancia, se desconocía de este hecho por lo que no pudimos identificar rápidamente de dónde surgían ciertos errores. Por suerte, luego de trasladar la problemática al equipo docente, se pudo solventar justo a tiempo para la fecha de entrega. De este modo, podemos concluir los siguientes puntos:

1) De manera individual aprendimos a colaborar en equipo y adaptarnos al ritmo de los demás, enfrentarnos como grupo ante las dificultades puntuales y generales que iban surgiendo y tomar responsabilidad sobre nuestras tareas.
2) A la par, ganamos experiencia valiosa tanto con GIT como GitHub. Sin mencionar la teoría y práctica del manejo de APIs. Además de esto, otro conocimiento no menos importante fue el de las distintas librerías; sus respectivos modos de uso y formas de instalarlas nos motivó a la búsqueda de herramientas de manera online y compartir, entre los integrantes, cómo emplearlas correctamente leyendo sus documentaciones y experimentando con las mismas.