Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación
IIC2233 — Programación Avanzada 2019-2

5 de septiembre de 2019
Actividad Formativa

Actividad 03

Iterables y Funciones de Orden Superior

Introducción

Eres un fanático de la música: ves música, respiras música, consumes música, y obviamente, escuchas música. Tu pasión en el área, y además tus habilidades de programación te motivan a buscar trabajo en Apple, en particular, con el reproductor iTunes. Sin embargo, como ven que ni siquiera has completado el curso de Programación Avanzada, te echan sin siquiera cuestionárselo. Estás a punto de renunciar a tu sueño, hasta que ves que en el edificio de al lado se encuentra la compañía **Pear**, donde coincidentemente necesitan con urgencia programadores que puedan aportar a su nuevo reproductor, **yoMelodias**. El fundador de Pear, **Steve Ruz** está tan desesperado por lanzar lo antes posible el nuevo reproductor, que no le importa tu experiencia previa en la Universidad, solo necesitan programadores. Al ver que la vida te da una segunda oportunidad, decides cumplir tu sueño (o bueno, algo parecido).

Figura 1: yoMelodias

Objetivo

Debido a que la compañía no es precisamente de las más grandes del mercado, el presupuesto de los servidores no es muy elevado, por lo que es primordial que los recursos de memoria sean utilizados eficientemente. Además el jefe técnico de Pear, **"Súper F" Florenzano** sabe que en Apple utilizan `for` y `while` para su plataforma, por lo que ha decidido que en Pear, la cual ahora te incluye, **solo se puede usar `for` y `while` dentro de funciones generadoras o estructuras de datos por comprensión**. Por esto, todo código que escribas en esta actividad debe seguir esta regla.

Bases de datos

Encriptación

Para mantener a los intrusos *hackers* lejos de sus preciadas bases de datos, ~~Apple~~ **Pear** decidió encriptarlas mediante el sistema **arquetipos**. Esta encriptación consiste en que cada palabra se modifica bajo la clave `arquetipos`, lo que significa que debes intercambiar las letras de la palabra `arquetipos` por los primeros diez números, desde el cero, según la siguiente tabla:

a	**r**	**q**	**u**	**e**	**t**	**i**	**p**	**o**	**s**
0	1	2	3	4	5	6	7	8	9

Cuadro 1: Clave arquetipos

Por ejemplo, al encriptar `'zapato'` utilizando el patrón arquetipos obtenemos la palabra `'z07058'`. También vale notar que si se encripta `'hernan4444'`, los números también se cambian por las letras a las que están asociadas, por lo que se obtendría `'h41n0neeee'`. Este hecho genera una propiedad interesante: **el proceso de encriptación y desencriptación es exactamente el mismo**. Puedes asumir que las letras a desencriptar solo serán letras minúsculas y los números.

Lamentablemente el genio tecnológico de Pear, **Steve Ruz**, olvidó solicitarle a su equipo de ayudantes que programaran la función `def decodificar(string)` cuyo propósito es decodificar el argumento `string` siguiendo el método descrito, y retornar el *string* decodificado. Así que te deja como primera tarea completar esta función, cuya base puedes encontrar en `decodificador.py`.

Para completar `def decodificar(string)`, recuerda la regla de sobre el uso de `for` y `while`. Puede que te sea de ayuda el método `chain`[1] de `itertools` que encadena dos iteradores. A su vez, puede que te sea de ayuda el método `''.join()`[2] de `str`, que recibe un iterador de *strings* y los une.

Archivos y obtener las bases de datos: yoNube

Los archivos de bases de datos ya encriptados son los siguientes:

- `artistas.csv`: Archivo que contiene la información de los artistas dentro de la base de datos, de la forma: `ID_Artista,Nombre_Artista,Genero,Año_formacion`.
- `canciones.csv`: Archivo que contiene la información de las canciones dentro de la base de datos, de la forma: `ID_Cancion,Nombre_Cancion,ID_Artista,Duracion`.
- `usuarios.csv`: Archivo que contiene la información de los usuarios que tienen una cuenta **Pear**, de la forma: `Nombre_del_Usuario,Username,Fecha_de_ingreso`
- `ratings.csv`: Archivo que contiene la información de los *ratings* (número del 1 al 5) dados por usuarios a canciones de los artistas, de la forma: `Username,ID_Cancion,Rating`

Ya que las bases de datos son muy grandes para pasártelas completas, en Pear te piden que las "descargues" desde su servicio de almacenamiento en línea **yoNube**. El sistema es bastante simple, pero debes procurar seguir las instrucciones al pie de la letra para obtener los archivos sin ningún problema. Los archivos necesarios para este proceso son `data_base.iTunes` y `yoNube.py`, los cuales deben estar en el mismo directorio.

[1] Puedes encontrar una referencia de su uso **aquí**.
[2] Puedes encontrar una referencia de su uso **aquí**.

Figura 2: yoNube

IMPORTANTE: Para obtener las bases de datos **debes ejecutar el archivo** `yoNube.py` **dentro de la carpeta AC03 de tu repositorio**, ya que incluye un archivo `.gitignore` que deberás subir. Además, durante la ejecución del archivo se irá señalando el estado del proceso de descarga para obtener los siguientes archivos de la base de datos:

- `data_base/canciones.csv`
- `data_base/artistas.csv`
- `data_base/usuarios.csv`
- `data_base/ratings.csv`
- `.gitignore`

Funciones Generadoras

Para la lectura de las bases de datos de **Pear** se debe hacer uso eficiente de la memoria, por lo que se te pide que utilices las siguientes funciones generadoras, que se encuentran en el archivo `archivos.py`. **Las cuales ya vienen implementadas en tu programa.** Cada una de estas funciones lee el archivo correspondiente según el `path` entregado y retornar un generador de las líneas del mismo, tal y como se encuentren (codificadas).

- `def leer_canciones(path)`
- `def leer_artistas(path)`
- `def leer_usuarios(path)`
- `def leer_ratings(path)`

Decorando la lectura de archivos

Para poder acceder al contenido de las bases de datos, las funciones de lectura de archivos no sirven por sí solas, ya que estas solo retornan el texto encriptado que leen de los archivos. Por lo tanto, hace falta utilizar **decoradores** para que decodifiquen y retornen las entidades esperadas: `Artista`, `Cancion`,

`Usuario`, `Rating`. Estas cuatro son `namedtuples` que puedes encontrar en `archivos.py`, y deben ser usadas para instanciar las entidades cargadas de los archivos.

Deberás implementar el decorador `@desencriptar(decodificar, tipo_archivo)`, que recibe como argumentos a la función `decodificar` definida antes y el tipo de entidad que la función decorada procesa. El decorador retornado por `@desencriptar(decodificar, tipo_archivo)` recibirá una función generadora, que entrega líneas de alguno de los archivo de la base de datos, y la función decorada (*wrapper*) que esta retorna es una función generadora con las instancias de la entidad correspondiente al `tipo_archivo` especificado. Es decir, internamente este decorador desencripta el texto generado por la función decorada y retorna, entidad por entidad, instancias de la *named tuple* necesaria.

A continuación dejamos un ejemplo de cómo se podría ver el decorador `@desencriptar`:

```
def desencriptar(decodificar, tipo_archivo):
    #Codigo de decorador...
    #Decodifica lo entregado desde el archivo
        ...
        funcion_decodificadora(linea_de_archivo)
    #Y segun tipo_archivo, entrega la instancia correspondiente
            ...
            instancia = Entidad(atributos_desencriptados)
            yield instancia

    return funcion_decorada


@desencriptar(funcion_decodificadora, "canciones")
def leer_canciones(path):
    ...
```

Aquí el flujo del decorador comienza recibiendo lo que entrega la función decorada con tal de decodificarlo, para posteriormente, mediante el `str` en la variable `tipo_archivo`, poder crear la instancia de la *named tuple* `Entidad`. Esto último es importante, ya que las *named tuples* no reciben una misma cantidad de atributos.

Consultas

Finalmente llegó el momento de que Steve Ruz vea materializado su arduo trabajo de delegaciones, es por ello que debes completar las siguientes funciones con tal de poder acceder a la información más relevante para la compañía. Debes implementar estas consultas, que puedes encontrar en el archivo `main.py`.

Por un lado, se quiere conocer información asociada a los usuarios como su antigüedad o las calificaciones que estos han dado al contenido de la plataforma.

- `def usuarios_por_antiguedad(usuarios)`: Esta función recibe un generador de usuarios y debe entregar otro generador con los usuarios ordenados por su antiguedad en sistema, es decir, desde el más antiguo al más reciente. Probablemente la forma más intuitiva de hacer esta consulta sea utilizando el generador completo, no te preocupes, ¡puedes hacerlo![3] Para esta consulta te

[3] Puedes encontrar una referencia de como ordenar en Python **aquí**.

será conveniente usar el método `datetime.strptime()` de la librería `datetime`, usando el formato `"%d/%m/%Y"`.

- `def ratings_usuario(username, ratings)`: Esta función recibe un *username* y un generador de *ratings*, debe entregar una **lista** con todos los *ratings* que ha emitido ese usuario.

Y por otro, se desea conocer información de las canciones y artistas.

- `def canciones_artista(nombre_artista, canciones, artistas)` Esta función recibe el nombre de un artista, un generador de canciones y un generador de artistas. Debe entregar un generador de canciones con las canciones de dicho artista.
- `def rating_promedio(nombre_cancion, canciones, ratings)` Esta función recibe el nombre de una canción, un generador de *canciones* y un generador de *ratings*. Debe entregar un `float` que represente el *rating* promedio que obtuvo esa canción.

Propuesto: decorador contador

Además de las consultas solicitadas, al jefe técnico **Súper F** le gustaría obtener los datos de la cantidad de consultas que hacen los usuarios mediante su plataforma, por lo que requiere que implementes un decorador para llevar las cuentas de cada una de la consultas descritas anteriormente y entregar sus resultados mediante una última consulta (función) `cantidad_consultas`.

El decorador `@contador` debe llevar la cuenta de cada una de las consultas para poder mostrarlas más adelante con la función `cantidad_de_consultas`. Para llevar la cuenta de las consultas podrías entregarle alguna estructura a `@contador`, como un `dict`. De implementarse, `def cantidad_consultas(consulta)` es una función que recibe un `str` que indica la consulta de la cual se quiere conocer su estadística, para finalmente retornar el número correspondiente.

A continuación se muestra un ejemplo del flujo esperado de este decorador y función:

```
@contador(...)
def consulta_1(...):
    ...

@contador(...)
def consulta_2(...):
    ...

consulta_1(...)

consulta_2(...)
consulta_2(...)
consulta_2(...)

cantidad_de_consultas("consulta_1") # 1
cantidad_de_consultas("consulta_2") # 3
```

Requerimientos

- (3.00 pts) Función `decodificar` y decorador `@desencriptar`.
  - (1.50 pts) Función `decodificar`.
  - (1.50 pts) Decorador `@desencriptar`.
- (3.00 pts) Consultas.
  - (0.75 pts) Consulta `usuarios_por_antiguedad`.
  - (0.75 pts) Consulta `ratings_usuarios`.
  - (0.75 pts) Consulta `canciones_artistas`.
  - (0.75 pts) Consulta `rating_promedio`.
- (Propuesto) Decorador `@contador` y consulta `cantidad_consultas`.
  - Decorador `@contador`.
  - Consulta `cantidad_consultas`.

Entrega

- **Lugar:** En su repositorio privado de GitHub, en la **carpeta** `Actividades/AC03/`
- **Hora del *push*:** 16:30

Auto-evaluación

Como esta corresponde a una actividad formativa, te extendemos la instancia de responder, después de terminada la actividad, una auto-evaluación de tu desempeño. Esta se habilitará a las **16:50 de jueves 5 de septiembre** y tendrás plazo para responderla hasta las **23:59 del día siguiente**. Puedes acceder al formulario mediante el siguiente enlace:

`https://forms.gle/VsVY2UUPRK5WUNaKA`

El asistir, realizar la actividad y responder la auto-evaluación otorgará como bonificación al alumno 2 décimas para sumar en su mejor actividad sumativa del semestre.