Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación
IIC2233 — Programación Avanzada 2019-1

14 de marzo de 2019
Actividad Sumativa

Actividad 01

Estructuras de Datos: *built-ins*

Objetivo

Esta actividad tiene por objetivo evaluar el conocimiento y uso de estructuras de datos básicas.

A lo largo de esta actividad es importante que para cada situación busques la estructura más adecuada para representar los datos. La actividad está especialmente diseñada para que utilices al menos una vez, según el caso, cada una de las siguientes estructuras: `namedtuple`, `deque`, `dict`, `defaultdict` y `set`. Aprovecha las propiedades de cada una para implementar las funcionalidades pedidas. Si utilizas listas para resolver todos los casos no obtendrás el puntaje completo la sección correspondiente.

Introducción

Se acabaron las vacaciones para todos los estudiantes de la UC, pero el DCC se niega a dejar de jugar. Durante las vacaciones, el **Gran Polea** completó cuatro juegos de una famosa saga, sin embargo, no le pareció suficiente. Por ello, será tu misión ayudarlo a programar un nuevo juego, totalmente original y distinto, para competir con los vicios del Gran Polea. El nombre de este juego 100 % original será **DCCims**.

Figura 1: Logo de DCCims

El Juego

DCCims es un juego de simulación social y estrategia donde el usuario simula una comunidad vecinal. Los personajes del juego son conocidos como **Cims** y son seres que simulan a personas de la realidad. Como información básica, éstos poseen un **nombre** por el cual los llaman y una **edad**. También, como

los humanos, son capaces de sentir una o más **emociones**. Además, cada Cim tiene una colección de **acciones** que representan actividades que el personaje quiere realizar, como correr, jugar o saltar. Tanto las emociones que sienten y las acciones que poseen cambian a lo largo del juego, y específicamente, cambian a medida que estos interactúan con otros Cims. Finalmente, nuestros personajes se agrupan en **familias**. Estos grupos son modificables, por lo que tienen la capacidad de agregar y eliminar a sus integrantes. También son capaces de entregar información sobre las emociones de la familia completa.

Programa

En primer lugar deberás crear las **entidades** correspondientes a los Cims y las agrupaciones correspondientes a las familias. Luego, debes almacenar estas entidades de forma en que puedas acceder rápidamente a ellas y puedas realizar distintas consultas y acciones.

Tanto los Cims como las familias tendrán un **identificador único** para encontrarlos donde sean almacenados, sin embargo éste identificador **no** debe ser parte de los atributos de cada entidad.

Por otra parte, como simulaciones de la realidad, los Cims tienen comportamiento similar a los humanos. Si bien pueden sentir varias emociones al mismo tiempo, cada sentimiento puede estar presente una única vez. En cambio, las acciones que quiere realizar un Cim sí pueden repetirse. Es más, deben ser guardadas en el orden que desea realizarlas.

Para lograr satisfacer al **Gran Polea**, debes realizar lo pedido a continuación para asegurar el correcto funcionamiento de tu programa.

Archivos

Junto al enunciado tendrás varios archivos que se deben utilizar en esta actividad.

- `main.py`: En este archivo está el esqueleto del programa que debes realizar. ¡Procura editar sólo las partes necesarias! Al final de este módulo hay una serie de pruebas junto con sus resultados esperados, para que veas si tus funciones de interacciones y consultas funcionan correctamente.
- `cims.txt`: Cada línea de este archivo contiene la información básica de un Cim de la siguiente forma: `id,nombre,edad,accion1;accion2;...` (las emociones vienen en un archivo aparte). Cada dato del Cim está separado por una coma '`,`'. Las acciones están separadas entre sí por un punto y coma '`;`'. Por ejemplo:

  `0,Dante,20,programar;cantar`

- `familias.txt`: Cada línea de este archivo contiene la información de una familia de la siguiente forma: `id_familia,id_miembro1;id_miembro2;...`. El primer dato es el identificador de la familia, mientras que el resto de los datos son identificadores de cada uno de los Cim que pertenecen a la familia, y se encuentran separados por un punto coma '`;`'. Los identificadores de familia no se repiten en el archivo.
  Importante: No hay una cantidad fija de miembros por cada familia, pero puedes suponer que cada familia tiene al menos un miembro. Ejemplo:

  `3,14;15;9;2`

- `emociones.txt`: Cada línea de este archivo contiene las emociones de un Cim en particular, de la siguiente forma: `id_cim,emoción` donde el primer dato es el identificador de un Cim y el segundo

dato es una emoción que el Cim tiene.
Muy importante: Un mismo Cim puede tener asignada más de una emoción en el archivo. Ejemplo:

```
0,feliz
1,nervioso
0,triste
0,feliz
```

Poblar el sistema

A continuación debes cargar el contenido de los distintos archivos al programa. Para ello debes completar las funciones `cargar_cims`, `cargar_familias` y `cargar_emociones` como corresponde. El **Gran Polea** desea que la información básica de cada entidad se guarde de manera eficiente, por lo que tendrás que elegir **estructuras de datos *built-in* de Python** de acuerdo a los datos.

- `cargar_emociones(ruta_archivo_emociones)`: Esta función recibe la ruta al archivo que contiene las emociones de los Cims del sistema: `emociones.txt`. Esta función debe cargar el contenido del archivo guardando la información de cada línea en una estructura de datos adecuada, de forma que las emociones de cada Cim se puedan asociar una única vez a ese Cim, de acuerdo a su identificador. Recuerda que las emociones pueden venir repetidas. Esta función debe retornar la estructura creada.
- `cargar_cims(ruta_archivo_cims, ruta_archivo_emociones)`: Esta función recibe la ruta al archivo `cims.txt` y la ruta al archivo `emociones.txt`. Se debe cargar el contenido del archivo de Cims leyendo cada línea de manera que, luego de cargar todos los Cims, se pueda obtener la información de cada uno de manera eficiente. La estructura elegida para guardar cada Cim debe guardar todos sus datos, incluyendo las emociones. Ésta a su vez debe ser eficiente. Esta función debe retornar la estructura creada.
- `cargar_familias(ruta_archivo_familia, cims)`: Esta función recibe la ruta al archivo de las familias de Cims del sistema: `familias.txt`, y la estructura que contiene los Cims cargados previamente. Se debe cargar el contenido del archivo guardando la información de cada línea de manera que, luego de cargar todas las familias, se pueda obtener cada uno de manera eficiente y que cada familia esté relacionada con sus respectivos Cims. Esta función debe retornar la estructura creada. **Importante:** la información de cada miembro debe quedar guardada en la estructura. Si guardas solo el identificador de cada miembro, **no recibirás todo el puntaje**.

Interacciones de los Cims

Ahora que tienes todos los datos de los Cims y las familias guardadas en estructuras de datos, tendrás que completar dos funciones de juego para relacionar a los Cims:

- `interactuar(id_cim_1, id_cim_2, cims)`: Recibe dos identificadores de Cims distintos y la estructura de datos que guarda todos los Cims. Los dos Cims referenciados deben extraer la **primera acción de su grupo de acciones** y "gastarla" en una interacción. Si ambas acciones son iguales, entonces cada Cim gana una emoción al azar del otro (recuerda que el Cim no puede tener dos emociones iguales). Si son distintas, entonces cada Cim recibe la acción recién gastada del otro **respetando el orden de las acciones para las siguientes interacciones**. **Hint:** para obtener un valor aleatorio de una estructura de datos (no necesariamente listas), te recomendamos usar el método `sample` de la librería `random`. `sample`. Este método recibe una estructura de datos y un entero `k`, y retorna una lista con `k` valores aleatorios de la estructura. Ejemplo: `sample([1, 2, 3], 1)` puede retornar `[1]`, `[2]` o `[3]`.

- `modificar_familia(id_cim, id_familia, cims, familias, agregar=True)`: Recibe los identificadores de un Cim y de una familia, y las estructuras con todos los Cims y todas las familias y un booleano que por defecto es `True`. Si `agregar` es verdadero, el Cim se agrega a la familia, mientras que si es falso éste será eliminado de la familia.

Consultas

Última paso. El **Gran Polea** te solicitó completar algunas funciones que le agreguen utilidad al programa que estás haciendo. A continuación se entrega una descripción de cada una.

- `emociones_comunes_por_familia(id_familia, familias)`: Esta función recibe el `id` de una familia, junto con las familias que se cargaron previamente. Debe retornar **una lista** de las emociones que tienen en común **todos** los integrantes de la familia entregada.

  Ejemplo: `['inspirado', 'vacilón', 'dormido']`

- `acciones_mas_repetidas(cims)`: Esta función recibe a los Cims y busca las acciones que se repiten más veces entre todos ellos. Debe retornar **una lista** con un listado de las acciones más repetidas y con la cantidad de veces que se repiten.

  Ejemplo: `[['programar', 'leer enunciado'], 5]`

Importante

En esta actividad está **prohibido trabajar con clases personalizadas, con el uso de excepciones y con variables globales**.

No sólo se espera que uses las estructuras de datos adecuadas para guardar la información, sino también que las utilices de manera correcta.

Por último, debes completar de manera correcta el módulo `main.py`, ya que se utilizará este módulo para corregir tu actividad.

Notas

- Recuerda lo visto en los contenidos y en ayudantía. Específicamente las estructuras disponibles para guardar información: `namedtuple`, `deque`, `dict`, `defaultdict` y `set`. Piensa bien cuál necesitas en cada caso.
- Si tienes problemas para abrir los archivos, considera cambiar el *encoding* (opciones: `'utf-8'` y `'latin-1'`). Por ejemplo: `open(archivo, modo, encoding="utf-8")`

Requerimientos

- (2.40 pts) Poblar el sistema
  - (0.60 pts) `cargar_emociones`.
    - (0.30 pts) Se guardan las emociones para cada Cim en la estructura adecuada.
    - (0.30 pts) Retorna la estructura de todas las emociones para cada Cim deseada.
  - (1.2 pts) `cargar_cims`.
    - (0.30 pts) Carga de manera correcta todos los Cims del archivo.
    - (0.30 pts) Las acciones se guardan en la estructura adecuada.
    - (0.30 pts) Se guardan las emociones correspondientes en la estructura de cada Cim.
    - (0.30 pts) Retorna la estructura deseada.
  - (0.6 pts) `cargar_familias`.
    - (0.30 pts) Carga de manera correcta todas las familias del archivo.
    - (0.30 pts) Retorna la estructura deseada.
- (1.8 pts) Lógica del sistema
  - (0.9 pts) Función `interactuar`.
    - (0.1 pts) Se realiza la interacción correcta con los Cims pedidos.
    - (0.4 pts) Se respeta el orden de las acciones.
    - (0.4 pts) Se controla la repetición de emociones de manera eficiente.
  - (0.9 pts) Función `modificar_familia`.
    - (0.45 pts) Funciona correctamente al agregar un Cim a una familia.
    - (0.45 pts) Funciona correctamente al quitar un Cim de una familia.
- (1.8 pts) Consultas
  - (0.9 pts) La función `emociones_comunes_por_familia` retorna el resultado en formato correcto **aprovechando las propiedades de la estructura de datos asignada**.
  - (0.9 pts) La función `acciones_mas_repetidas` retorna el resultado en formato correcto **aprovechando las propiedades de la estructura de datos asignada**.

Entrega

- **Lugar:** En su repositorio privado de GitHub, en la **carpeta** `Actividades/AC01/`
- **Hora límite del último *push*:** 16:30:00