Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación
IIC2233 — Programación Avanzada 2021-1

27 de Mayo de 2021
Actividad Formativa

Actividad Formativa 6

Iterables

Entrega

- **Lugar:** En su repositorio privado de GitHub, en la **carpeta** `Actividades/AF6/`
- **Hora del *push*:** 16:30

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

Debido a la pandemia, los animales del campus San Joaquín han estado solos por mucho tiempo por lo que extrañan las caricias y alimentos que los estudiantes les daban en el pasado. Esto, sumado a la reducción del presupuesto para su alimentación, los ha llevado a una situación crítica donde se pueden escuchar por la noche los tristes y solitarios sollozos de los desamparados amigos peludos.

Por suerte, el DCC siempre tiene una solución (y nombres compatibles con DCC), y ha propuesto apadrinar a los solitarios animalitos del campus y también enviarles comida. Con esta idea millonaria, cualquier miembro de la comunidad UC puede decidir apadrinar un animal para costear sus necesidades veterinarias y recreativas (además por apadrinar se regalara un peluche representativo del animal) como también comprarles comida (ya sea la ideal a sus gustos o la que sea acorde a su especie). Así ha nacido la iniciativa y pagina web:

Lamentablemente, muchos miembros del DCC han estado ocupados con otros proyectos (que también empiezan con DCC), por lo que han pedido tu ayuda como estudiante de Programación Avanzada para implementar las consultas a la base de datos de la página.

Flujo de DCCachorritos

La tienda posee dos bases de datos como lo son los `animales` a apadrinar y las `comidas` que se encuentran a la venta. Tu deber es realizar consultas a estas bases de datos que permitirán obtener información para que quienes usen la página puedan tomar decisiones informadas y pertinentes.

Archivos

Archivos de datos

- `animales.txt`: Este archivo contiene la información de los animales a apadrinar. No debes modificarlo
  El formato del archivo es:

  `Nombre;Especie;Raza;Edad;Estatura;Precio`

  donde `precio` es el valor de apadrinar al animal de la fila correspondiente.
- `comidas.txt`: Este archivo contiene la información de las comidas de los animales a la venta. No debes modificarlo
  El formato del archivo es:

  `Nombre;Especie;Raza;Precio;Disponible`

  donde `disponible` es un valor booleano indicando si hay *stock* para la venta, y `especie` y `raza` indican a que tipo de animal va dirigido el producto.

Archivos de código

- `entidades.py`: Este archivo contiene las entidades `Mascota` y `Comida` que representan los datos utilizados en la actividad. No debes modificarlo
- `cargar.py`: Este archivo sirve para cargar los datos de `animales.txt` y `comidas.txt`. No debes modificarlo
- `parametros.py`: Este archivo contiene las rutas de las archivos de texto y además un diccionario con las clasificaciones de tamaños de los animales, donde la llave es el tamaño del animal y el valor es una tupla con los valores limite de dicho tamaño, este diccionario se vería de la siguiente manera:

  ```
  TAMANOS = {"PEQUENO": (0,30), "MEDIANO": (30,60), "GRANDE": (60,100)}
  ```

  No debes modificarlo
- `main.py`: Este archivo es el que realiza las consultas e imprime los resultado en pantalla. No debes modificarlo
- `consultas.py`: En este archivo debes trabajar y completar las consultas que te piden. Debes modificarlo
- `iterable.py`: Este archivo contiene las clases `IterableDescuentos` y `IteradorDescuentos`, que se utilizan para recorrer los datos de una manera personalizada. Debes modificarlo

Entidades base y carga de datos

Para que puedas implementar correctamente las funcionalidades, te entregamos las siguientes clases **ya implementadas** en el módulo `entidades.py`: No debes modificarlo

- `class Mascota`: Posee los atributos `nombre (str)`, `especie (str)`, `raza (str)`, `edad (int)`, `estatura (int)` y `precio (int)`.
- `class Comida`: Posee los atributos `nombre (str)`, `especie (str)`, `raza (str)`, `precio (int)` y `disponible (bool)`.

Para cargar los datos y utilizar las clases anteriores, te entregamos las siguientes funciones generadores **ya implementadas** en el módulo `cargar.py`: No debes modificarlo

- `def cargar_mascotas(ruta: str) -> Generator[Mascota]`: Esta función generadora recibe un `str` con la ruta del archivo y va retornando instancias de `Mascota` según los contenidos del archivo `*.txt`.
- `def cargar_comidas(ruta: str) -> Generator[Comida]`: Esta función generadora recibe un `str` con la ruta del archivo y va retornando instancias de `Comida` según los contenidos del archivo `*.txt`.

Funcionalidades a implementar

Consultas

Debes realizar las consultas solicitadas en el archivo `consultas.py`. Para esta actividad ten en mente el uso de funciones `lambda`, pero considera que el énfasis está en el uso de las funciones `map`, `filter` y `reduce`. Además esta prohibido el uso de ciclos `for` y `while` con excepción de la consulta `comida_ideal`.

- `def obtener_comidas() -> list`: debe retornar una lista con todos los nombres de las comidas, a partir del generador de comidas obtenido de la función `cargar_comidas` de `cargar.py`.
- `def agrupar_por_tamano(tamanos: dict) -> list`: recibe un diccionario donde las llaves son los strings `PEQUENO`, `MEDIANO` y `GRANDE` y el valor de cada uno es una tupla con el intervalo de tamaños permitidos. Este diccionario se encuentra en el archivo `parametros.py`. Con esto, y utilizando el generador de mascotas obtenidos de la función `cargar_mascotas` de `cargar.py`, se debe lograr seleccionar y agrupar las mascotas según su estatura, ordenándolas en pequeñas, medianas y grandes.

  Por ejemplo, si la estatura de una mascota está en el rango definido por la tupla en el valor `"PEQUENO"` del diccionario, entonces la mascota se clasifica como pequeña.

  Se debe retornar una lista de listas donde las listas internas corresponden a las agrupaciones pequeños, medianos y grandes de la forma:

  ```
  [[animal_pequeña1, ...], [animal_mediana1, ...], [animal_grande1, ...]]
  ```

- `def precio_total(especie: str) -> int`: recibe una especie animal en particular. Debe retornar el valor total de los animales de la especie indicada, sumado con el valor de las comidas de esta especie. Además de esto, para esta consulta, es necesario utilizar ambos generadores del archivo `cargar.py`.

- `def comida_ideal(raza: str, especie: str) -> Generator[Comida]:` recibe una raza y una especie y además se debe utilizar el generador de comida de `cargar.py`. En esta función debes utilizar `yield`, para implementar una función generadora. El objetivo será crear un generador que retorne todas las comidas disponibles para esa raza y especie. **Acá puedes utilizar `for` o `while` si lo deseas, pero recuerda mantener todo en generadores y evitar crear listas.**
- `def precio_comidas(raza: str, especie: str) -> int:` Recibe una especie y una raza para poder utilizar la función generadora `comida_ideal`. Debe retornar la suma total de los precios de las comidas ideales.

Iterable e iterador de descuentos

En esta última parte, deberás aplicar tus conocimientos de **iterables personalizados**, para implementar la clase `IteradorDescuentos`, que corresponde al iterador de la clase `IterableDescuentos`.

Ambas clases se encuentran en el archivo `iterable.py` y la intención de estas es poder recorrer los animales, de forma ordenada, según su edad (de mayor a menor edad), y otorgarles un descuento al precio por edad. El descuento consiste en un 10 % de descuento por cada año de edad, con un tope de 35 % (es decir, `min(0.1 * edad, 0.35)`). Luego, antes de retornar la instancia que corresponda, se le debe agregar un atributo `descuento_por_edad` indicando el porcentaje de descuento calculado, y actualizar el `precio` aplicando el descuento (si corresponde un 10 % de descuento, entonces el precio será el 90 % del valor inicial). Deberás completar los métodos `__iter__()` y `__next__()` de la clase `IteradorDescuentos`.

- `class IterableDescuentos`: Esta clase recibe como argumento una lista de mascotas. No debes modificarlo
  - `def __iter__(self)`: Este método retorna una instancia `IteradorDescuentos`.
- `class IteradorDescuentos`: Esta clase recibe como argumento una instancia de `IterableDescuentos` y guarda una copia de ésta como atributo.
  - `def __iter__(self)`: Este método debe retornar a la instancia misma del iterador (`self`). Debes modificarlo
  - `def __next__(self)`: Este método es el encargado de encontrar la **siguiente mascota con la mayor edad (si es que existe), calcular el descuento y modificar el precio**, y retornar dicha mascota (con sus atributos actualizados).[1] Debes modificarlo

Notas

- Puedes ejecutar el archivo `main.py` para verificar el comportamiento de las consultas. En algunas consultas, los argumentos entregados son aleatorios por lo que los resultados pueden variar en cada ejecución.
- Recuerda que en los iteradores cuando no quedan objetos por recorrer el iterador debe levantar una excepción de tipo `StopIteration`.
- Recuerda que las consultas se realizan externamente, por lo que solo debes trabajar con las variables que recibes en las funciones. Es por esto por ejemplo que en la consulta `agrupar_por_tamaño` no debes utilizar las variables que vienen incluidas en el archivo `parametros.py`, pues estas se le entregaran a la función en el archivo `main.py`.
- La función `max` retorna el máximo en un iterable. Pero también puede recibir un argumento `key`, que permite específicar una forma de acceder al valor según el cual se encuentra el valor máximo.

[1] Las funciones `max` y `sorted` te pueden ser de utilidad. En las notas puedes encontrar más información.

Por ejemplo, la siguiente línea obtiene aquella tupla dentro de la lista que tiene mayor valor en su primera componente.

```
max([ (2, 2), (4, 0), (1, 5) ], key=lambda t: t[0])
```

La función `lambda t: t[0]` recibe cada elemento y retorna el valor que se compara, en este caso, la primera posición de la tupla. El resultado de la línea anterior es `(4,0)`.

- La función `sorted` retorna un iterable con el contenido ordenado de forma creciente, y de forma similar al caso anterior permite un atributo `key` idéntico que permite especificar según que medida ordenar. El siguiente ejemplo retorna en orden creciente las tuplas de una lista según la segunda posición:

```
sorted([ (2, 2), (4, 0), (1, 5) ], key=lambda t: t[1])
```

Objetivo

- Aplicar conocimientos de iterables utilizando funciones `map`, `filter` y `reduce`.
- crear un iterador personalizado definiendo correctamente los metodos `__iter__` y `__next__`.
- Implementar una función generadora, utilizando correctamente `yield`.