Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación
IIC2233 — Programación Avanzada 2020-2

17 de Septiembre de 2020
Actividad Propuesta

Actividad Propuesta 01

Iterables

Entrega

- **Lugar:** En su repositorio privado de GitHub, en la **carpeta** `Actividades/AP01/`
- **Hora del *push*:** Esta es una actividad de práctica por lo que no tiene hora de entrega 😊

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

¡Se viene el 18 de Septiembre! ¡La fiesta más importante del año! Hemos notado que durante estos últimos meses, el *delivery* ha tomado mucha importancia para nuestras compras y sabemos que este fin de semana no va a ser una excepción. Es por esto que el DCC te ha contactado por tus grandes habilidades computacionales para que combines el amor de los chilenos por el 18 y el *delivery*, creando un emprendimiento épico: el **DCCiochero**.

Flujo de DCCiochero

Tú como administrador de este *delivery*, y aprovechando que eres alumno del curso de Programación Avanzada, decides elaborar un programa que permita manejar los datos de tus clientes y hacer consultas sobre los pedidos. Estas consultas te permiten conocer mejor las estadísticas de tu emprendimiento y manejar correctamente los datos, para tomar decisiones informadas y pertinentes. El programa consta de `DCClientes`, que son los clientes que han participado del delivery, y `DCComida`, las comidas que proporcionas en tu servicio. Como ya posees las bases de datos, como administrador te interesaría saber:

- Top Clientes: cuáles son tus clientes más fieles.
- Filtro Dieciochero: cuáles de tus clientes compran tus comidas dieciocheras.

Archivos

Archivos de datos

- `dcclientes.csv`: Este archivo contiene la información de cada cliente presente en nuestra base de datos. La primera línea del archivo contiene el nombre de las columnas y el resto de las líneas contienen la información de cada cliente,

  ```
  id,nombre,comuna,id_comidas
  ```

  Donde:

  - `id`: número identificador del cliente.
  - `nombre`: nombre del cliente.
  - `comuna`: nombre de la comuna donde reside el cliente.
  - `id_comidas`: lista de números identificadores (`id`) de las comidas que ha pedido el cliente. Cada `id` está separado por `;`. Un ejemplo sería `1;4;5` que equivale a las comidas con `id` 1, 4 y 5.

- `dccomida.csv`: En este archivo encontrarás los datos de todas las comidas. La primera línea del archivo contiene el nombre de las columnas y el resto de las líneas contienen la información de cada uno, separadas por coma, de la forma:

  ```
  id,nombre,precio,dieciochero
  ```

  Donde:

  - `id`: número identificador de la comida.
  - `nombre`: nombre de la comida.
  - `precio`: precio de la comida (sin costo de envío).
  - `dieciochera`: es `True` si la comida es dieciochera, `False` si no lo es.

Archivos de código

- `cargar_datos.py`: Este archivo sirve para cargar los datos de `dcclientes.csv` y `dccomida.csv`. Posee 2 funciones: (**Este archivo ya está implementado y no tienes que modificarlo**)
  - `def cargar_comida(path)`: Esta función recibe un str con la ruta del archivo y devuelve una lista con las instancias de DCComida según lo entregado en los archivos `.csv`
  - `def cargar_cliente(path)`: Esta función recibe un str con la ruta del archivo y devuelve una lista con las instancias de `DCCliente` según lo entregado en los archivos `.csv`.
- `entidades.py`: Este archivo tiene las entidades del programa ya modeladas. Se encarga de guardar correctamente los atributos en las clases.**No debes modificarlo.** Contiene:
  - `class DCCliente`:
    - `def __init__(self, id, nombre, comuna, productos)`
      - `self.id`: contiene el `id` del cliente en formanto `int`.
      - `self.nombre`: contiene el `nombre` del cliente en formato `str`.
      - `self.comuna`: contiene el nombre de la `comuna` del cliente en formato `str`.
      - `self.ids_comida`: contiene una `list` que tiene todos los ids de las comidas que compró este cliente.
    - `def obtener_comida_comprada(self, comidas)`: Recibe un generador de `DCComida` y printea los nombres de toda la comida comprada por el cliente
    - `def __str__(self)`: Este método se usa para imprimir la información de los clientes de manera ordenada.
  - `class DCComida`:
    - `def __init__(self, id, nombre, precio, dieciochero)`:
      - `self.id`: contiene el `id` de la comida en formanto `int`.
      - `self.nombre`: contiene el `nombre` de la comida en formato `str`.
      - `self.precio`: contiene el `precio` de la comida en formato `int`.
      - `self.dieciochera`: contiene un `bool` que identifica si la comida es dieciochera o no.
    - `def __str__(self)`: Este método se usa para imprimir la información de las comidas de manera ordenada.
- `consultas.py`: Este archivo es el principal del programa. Aquí es donde debes demostrar tus habilidades y **completar las funciones que se explican en la parte I**. Sirve para hacer los filtros interesantes para nuestro DCCiochero. Además, hay otras funciones que ya están implementadas. Estas se explican a continuación y **no debes modificarlas**:
  - `def modificador_costo(ponderador)`: Es una función matemática que servirá para modificar el costo de envío según cuántas comidas lleve el cliente, de tal forma que el costo de envío disminuya a medida que se realizan más pedidos. Esta función recibe un `int` `ponderador` y le realiza una modificación para que disminuya una cierta cantidad, luego lo devuelve.
  - `def comidas_dieciocheras_por_cliente(cliente, lista_comida)`: Esta función recibe una instancia de `DCCliente` y toma los id que tiene en `comidas` para revisar si son dieciocheros o no. Retorna una lista ordenada con los *bools* atributos `dieciochero` de sus comidas.

- `def clientes_dieciocheros(lista_clientes, lista_comida)`: Esta función recibe las listas que contienen todos los clientes y comidas, a partir de ambas listas se crea un diccionario que posee a los `clientes_dieciocheros`. Para determinar si un cliente es `dieciochero` se verifica que el cliente haya pedido al menos una comida `dieciochera`, de lo contrario lo agrega a una lista de `clientes_apagados`, para luego printearlos.
- `def top_clientes(clientes, comidas)`: Recibe las listas de todos los clientes y comidas, utiliza las funciones `precio_total` y `obtener_comidas` (funciones que tú deberás completar según se explica en la parte I del enunciado) para obtener los 5 mejores clientes y los devuelve.

Parte I:

En esta parte tendrás que usar las funciones `map`, `filter` y `reduce` para hacer la consulta Top Clientes que corresponden a los 5 clientes que más dinero han gastado. La consulta se llama a través de la función `top_clientes(clientes, comidas)` que viene implementada y que ocupa las funciones `obtener_comida`, `obtener_comidas`, `precio_total` las cuales debes completar.

Además debes hacer la consulta Filtro Dieciochero que nos da los clientes que han comprado al menos una comida dieciochera, para esto debes completar la función `filtrar_dieciocheros`. En el archivo `consultas.py` deberás completar las siguientes funciones:

- `def obtener_comida(id, comidas)`: recibe un id de comida y una lista de instancias de la clase `DCComida`. Aquí debes utilizar `filter` y asociar el id a su respectiva comida de la lista comidas y retornar su instancia.
- `def obtener_comidas(ids, comidas)`: recibe una lista de ids de comidas y una lista de instancias de la clase `DCComida`. Aquí debes utilizar `map` y la función `id_a_comida(id, comidas)` para asociar cada id a su respectivo producto de la lista comidas y retornar una lista de instancias de estas.
- `def precio_total(comdias_cliente)`: recibe una lista de instancias de comidas. Aquí debes utilizar `reduce` para sumar el precio de todas las comidas y retornar el total.
- `def filtrar_dieciocheros(cliente, lista_comida)`: Recibe una instancia de `DCCliente` y una lista de los atributos `dieciochero` de sus instancias de `DCComidas`. Deberás utilizar `filter` y la función `productos_por_clientes` para filtrar si posee al menos un artículo dieciochero y devolver la lista con los que cumplan esta condición.

Parte II:

En esta sección deberás usar `yield` para implementar una función generadora y `next` para utilizarla. En esta parte deberás usar el `modificador_costo` que **ya viene implementado**. En el archivo `calcular_costos.py` deberás completar las siguientes funciones:

- `def costo(costo_inicial)`: Esta es una **función generadora** que recibe un `int` que corresponde al costo de envío base para todos los clientes. Deberás usar el `modificador_costo` y pasarle como parámetro un entero `ponderador` que es inicialmente igual a cero, y deberá incrementar a cada iteración en 0.4. El costo de envio se calculará como:

$$costo\_envio = (costo\_envio \times modificador\_costo(ponderador))$$

  Además debes considerar que no queremos dar un descuento tan grande en costo de envio, así que la función anterior será solo válida hasta que el `costo_envio` sea un 30 % del `costo_inicial`. En esta función deberás usar `yield` en lugar de `return` y deberás retornar un generador

- `def calcular_costo_envio(comidas, costo_inicial)`: recibe una lista de instancias de DCComida de un cliente y un `int` que corresponde al costo de envío base para todos los clientes. Acá deberás utilizar la función generadora `costo` para crear un generador y deberás usar el método `next` para calcular el costo de envio del cliente según el número de comidas que tenga. Debes retornar este costo de envio final.

Notas

- Recuerda que los generadores pueden ser recorridos una sola vez, por lo que si quieres iterar sobre ellos debes buscar soluciones diferentes de `list(generador)`.
- La librería `itertools` puede ser de ayuda para esta actividad y para trabajar con iterables en general.

Objetivos de la actividad

- Aplicar conocimientos de Iterables utilizando funciones map, filter y reduce
- Implementar una función generadora, utilizando correctamente yield
- Utilizar generadores haciendo uso del método next