Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación
IIC2233 — Programación Avanzada 2020-1

23 de abril de 2020
Actividad Sumativa

Actividad Sumativa 02

Iterables

Entrega

- **Lugar:** En su repositorio privado de GitHub, en la **carpeta** `Actividades/AS02/`
- **Hora del *push*:** 16:50

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

Luego de haber demostrado tus increíbles habilidades como programador creando el DCChallenge y la DCComisaría Virtual, una empresa de desarrollo escucha sobre lo bueno que eres programando y te pide ayuda. La empresa quiere crear **DCCorner PograShop** una página en donde todos los chilenos podrán realizar sus compras del supermercado de forma *online* y recibir sus pedidos a través de *delivery*. Para lograr esto deberás aplicar todos tus conocimientos sobre **iterables**, ¡y así, DCCorner PograShop será un éxito y todos estarán ansiosos por probarla!

DCCORNER PROGRASHOP

Cuando esté completo, el programa DCCorner Prograshop iniciará con la ejecución del `main.py`, que extrae los datos de **clientes** y **productos** de archivos CSV y los almacena por separado en dos listas, para luego procesarlos y mostrárselos a los clientes de la forma más atractiva posible. Por temas de privacidad, los nombres de los clientes que se cargan vienen encriptados, por lo que el programa debe aplicar un par de funciones sobre la información para obtener los valores reales. Si bien una parte importante del programa ya está hecha, este aún contiene varias funcionalidades a implementar, como la impresión en pantalla de productos por categoría, el cálculo del total de compra de cada cliente, la búsqueda de productos según su disponibilidad y el *display* de las distintas ofertas de forma ordenada, por lo que será tu deber completarlas.

Archivos

Para esta actividad se te hará entrega de los siguientes archivos.

- `main.py`: Este es el archivo principal del programa. Puedes ejecutarlo para probar el funcionamiento de tu programa completo. **Ya viene implementado, y no debe ser modificado**.
- `entidades.py`: Este archivo contiene a las clases `Cliente`, `Producto`, e `IterableOfertones`, **que ya vienen implementadas y no se deben modificadar**; pero también contiene a la clase `IteradorOfertones` que **debes completar**.
- `funcionalidades.py`: Este archivo contiene funciones que **deberás completar** para que el resto del programa se ejecute correctamente.

La carpeta `data` contiene archivos CSV que se leen para poblar el programa. Este cargado de información ya viene implementado en `main.py`. Para evitar problemas al momento de leer los archivos, evita abrir los CSV con Excel.

Entidades base y función de desencriptado

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

- `class Producto`: Posee los atributos `id_` (`int`), `nombre` (`str`), `categoria` (`str`), `precio` (`int`), `disponible` (`bool`) y `descuento_oferta` (`int`).
- `class Cliente`: Posee los atributos `id_` (`int`), `nombre` (`str`) y `carrito` (`list`). Este último es una lista con las instancias de productos a comprar.

Por otro lado, se te entrega la función `def desencriptar(cliente_encriptado)` dentro del archivo `funcionalidades.py` **que también viene implementada, no debes editarla**, sin embargo, se te explica su funcionamiento. Esta función recibe una lista con información encriptada de un cliente. Esta lista es de la forma `[id_, nombre, [id_producto_1, id_producto_2, ...]]`. **Esta función retorna la lista de la información que recibe, pero con el nombre desencriptado.**

Funciones a implementar

Como **DCCorner PograShop** es una tienda virtual que se centra en la elegancia, te han puesto la condición de que **no puedes usar `for` ni `while`, salvo dónde se te indique explícitamente**. Además, para cada una de las funciones se especifica una **restricción** adicional de implementación. Estas deben ser seguidas para obtener el puntaje completo en cada parte.

Las funciones del archivo `funcionalidades.py` a completar son:

- `def obtener_clientes(lista_clientes_encriptados)`: Esta función recibe una lista de listas con la información de cada cliente. La función debe retornar la misma lista de listas entregada pero con los nombres desencriptados. **Aquí debes utilizar `map` y la función `desencriptar(cliente_encriptado)`.** Un ejemplo del argumento que recibe esta función es el siguiente:

```
[
    [id_, nombre, [id_producto_1, id_producto_2,...]],
    [id_, nombre, [id_producto_1, id_producto_2,...]],
    ...
]
```

- `def categorizar(productos, categoria)`: Esta función recibe una lista de instancias de la clase `Producto` y el nombre de una categoría. Debe retornar una lista con las instancias recibidas que pertenecen a la categoría especificada. **Aquí debes utilizar `filter`.**
- `def calcular_precio(productos)`: Esta función recibe una lista de instancias de la clase `Producto`, y retorna un `int` con la suma de sus precios. **Aquí debes utilizar `reduce` y no puedes usar `sum`.** Solo debes considerar el atributo `precio` de cada producto para calcular el monto total.
- `def generar_productos_disponibles(clientes)`: Esta es una **función generadora** que recibe una lista de instancias de la clase `Cliente`. Este generador entrega todos los pares `(cliente, producto)` posibles como tuplas, donde `cliente` es una instancia de la clase `Cliente` en la lista entregada y `producto` es un producto en el carrito del cliente (en el atributo `carrito` de `Cliente`), que además está **disponible**. Un producto está disponible si su atributo `disponible` es `True`. **En esta función tienes la libertad de poder usar `for` y `while` si gustas**.

Importante: Si ya completaste alguna de las funciones anteriores, sube una actualización de tu avance a tu repositorio remoto.

Iterable e iterador de ofertones

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

Ambas clases se encuentran en el archivo `entidades.py` y la intención de estas es poder recorrer los productos, de forma ordenada, según sus descuentos (de mayor a menor porcentaje). El porcentaje de descuento está indicado en el atributo `descuento_oferta` de cada producto, como un `int` que va entre `0` y `100`. Deberás completar los métodos `__iter__()` y `__next__()` de la clase `IteradorOfertones`.

- `class IterableOfertones`: Esta clase recibe como argumento una lista de productos.
  - `def __iter__(self)`: Este método retorna una instancia `IteradorOfertones`.
- `class IteradorOfertones`: Esta clase recibe como argumento una instancia `IterableOfertones` y guarda una copia de ésta como atributo.
  - `def __iter__(self)`: Este método retorna a la instancia misma del iterador (`self`).
  - `def __next__(self)`: Este método es el encargado de encontrar el **siguiente producto con el mayor porcentaje de descuento, aplicar el descuento**, y retornar dicho producto.[1]

[1] Las funciones `max` y `sorted` te pueden ser de utilidad. Revisa las notas más adelante.

Bonus: Re-implementar sin `for` y `while` (1 punto)

Como *bonus*, puedes implementar la función `def cargar_bonus(ruta_clientes)`, que se encuentra en el archivo `main.py` **replicando el funcionamiento de** `def cargar_encriptados(ruta_clientes)` **pero sin el uso de** `for` **y** `while`. Puedes agregar definiciones de funciones auxiliares si lo necesitas.

Para utilizar tu función en lugar de la carga original, puedes comentar la línea 38[2] y descomentar la línea 39[3] de `main.py`.

Notas

A continuación se explica brevemente el funcionamiento de dos funciones que pueden ser de ayuda para la implementación de `IteradorOfertones`. Nota que **no utilizarlas es válido y se considera correcto**, es solo una sugerencia.

- 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. 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])
  ```

Requerimientos

- (1.25 pts) Completa la función `def obtener_clientes(lista_clientes_encriptados)` correctamente.
- (1.25 pts) Completa la función `def categorizar(productos, categoria)` correctamente.
- (1.25 pts) Completa la función `def calcular_precio(productos)` correctamente.
- (1.25 pts) Completa la función `def entregar_producto(productos)` correctamente.
- (1 pt) Completar `class IteradorOfertones`.
  - (0.25 pts) Completa el método `__iter__()` correctamente.
  - (0.75 pts) Completa el método `__next__()` correctamente.
- **(*Bonus* 1 pt)** Re-implementar `def cargar_encriptados(ruta_clientes)`.

[2] `encriptados = cargar_encriptados(os.path.join('data', 'clientes_encriptados.csv'))`
[3] `encriptados = cargar_bonus(os.path.join('data', 'clientes_encriptados.csv'))`