Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación
IIC2233 — Programación Avanzada 2018-2

8 de noviembre de 2018
Evaluada

Actividad 12

I/O: Serialización

Introducción

Tras semanas de lucha por la conquista de *PyKitchen*[1] entre el malvado mafioso *Enzini Tini Tamburini* y el Chef Funcional Nebil, junto a su equipo de *sous-chefs*: Chef Decorador Jaime y Chef Excepcional Felipe —todos conocidos como los DCChefs—, la batalla ha llegado a su fin. Por su victoria, los DCChefs han sido premiados con el privilegio de cocinar las recetas escritas en el *PyKitchen Cookbook*. Sin embargo, han quedado secuelas de la guerra entre ambos equipos. En esto, ¡El malvado *Tini* ha invadido las premisas de la organización, contaminado los ingredientes y alterado el libro de recetas *PyKitchen Cookbook*!

A raíz de esto, el equipo de DCChefs ha solicitado tu ayuda. Como miembro del DCC (**D**epartamento de **C**ocina **C**omputacional), te sientes preparado para esta aventura. Tu objetivo en esta actividad será cocinar, de manera correcta, las recetas provistas en el *PyKitchen CookBook* (de ahora en adelante, por simplicidad: **PKC**).

El equipo de DCChefs **te ha provisto con una interfaz de cocina que utilizarás para probar que tu código funcione correctamente**.

Instrucciones

Estructura de archivos

Dentro de los archivos provistos se encuentran los siguientes:

- `RecetasLockJSON.json`: **Este archivo contiene los atributos que deben contener los distintos tipos de recetas**. En otras palabras, contiene los atributos que deben contener las `Bebidas` y los atributos que deben contener las `Comidas`. El malvado *Tini Tamburini* no fue capaz de dañar este archivo, por lo que su contenido es fidedigno.
- `recetas.book`: Este es conocido como el famoso **PKC**. Corresponde a un archivo serializado con `pickle`, donde se almacena una **lista donde cada elemento es del tipo** `Receta`. Sin embargo, el malvado *Enzini Tini Tamburini* ha contaminado este archivo agregando atributos inútiles a las recetas e ingredientes inválidos. Es tu misión procesar las recetas y cocinarlas correctamente.
- `ingredientes.txt`: El equipo de PyKitchen ha logrado proveerte un archivo con todos los ingredientes que sí son válidos en una receta. Usarás este archivo para limpiar las recetas serializadas con `pickle`.

[1] Una organización sin fines de lucro de hombres y mujeres, que luchan por el uso de Python en la cocina diaria.

Objetivo

En esta actividad deberás implementar algunas funciones en las clases `Receta` y `Comida` (ambas ubicadas en `clases.py`), así como funciones en el archivo `backend.py`. Revisaremos lo que hay que hacer parte por parte:

Clase `Receta` (ubicada en `clases.py`)

1. Implementa la función `def abrir_recetas_lock(self)`, que deberá leer el archivo `RecetasLockJSON.json` y retornar un `set` de los atributos permitidos en las recetas.
2. Implementa la **deserialización** del objeto con `pickle`. Al momento de deserializar, asegúrate de realizar lo siguiente:
   a) Recuerda que el archivo desde donde se deserializará está contaminado. Cada instancia tiene más atributos de los que deberían estar según `RecetasLockJSON.json`. Por ello, debes remover todos los atributos que sean inválidos[2].
   b) La lista de ingredientes que se cargue (atributo `ingredientes`) podría tener ingredientes inválidos. Debes limpiar esta lista de acuerdo al contenido de `ingredientes.txt`[3].
3. Implementa la **serialización** del objeto con `pickle`. Al momento de serializar, debes *setear* el atributo `llave_segura` con el valor que retorna la función `def encriptar(self)`, que ya está implementada. Esto servirá para verificar que la receta está limpia antes de cocinar con ella.

Clase `Comida` (ubicada en `clases.py`)

Para «meter la comida al horno»[4] debemos serializarla en formato `json`. Al revés, para sacar la comida del horno debemos deserializarla desde un `json`.

1. Implementa la **serialización** de `Comida` en formato `json` en la clase `ComidaEncoder`. Al serializar, el `json` debe contener un atributo `fecha_ingreso`, que contiene un *string* representando el `datetime` del momento en que se metió al horno[5]. No olvides incorporar todos los atributos necesarios para reconstruir la instancia (*i.e.* `nombre`, `nivel_preparacion`, `ingredientes`, etcétera).
2. Modifica la **inicialización** de `Comida`. Como sabrás, al deserializar un `json` que representa una comida crearemos una instancia de `Comida` a partir de esos datos. Las comidas tienen un **nivel de preparación**, que indica el porcentaje de cocción del alimento. Si este número es igual o mayor a 100 decimos que la comida está preparada, por lo que la *property* `preparado`—ya implementada—devolverá `True`. Asimismo, si el nivel de preparación sobrepasa el 100 %, entonces la *property* `quemado` devolverá `True`. Para que todo funcione bien, debes asegurarte que se cumpla lo siguiente en el `__init__`:
   a) El inicializador recibe el `nivel_preparacion` que tenía la comida antes de meterla al horno, y la `fecha_ingreso` de esta al horno. Debes *setear* el `nivel_preparacion` de tal forma que aumente en un 1 % por cada minuto que pasó entre la `fecha_ingreso` y el momento en que se deserializa. Por ejemplo, si el `nivel_preparacion` que llega al inicializador es de 50, y `fecha_ingreso` es una fecha de hace 10 minutos, entonces debes *setear* `nivel_preparacion` en `50 + 10`. Cuida de verificar que `fecha_ingreso` no sea `None`.

[2] ***Hint:*** En el paso anterior, implementaste la lectura de `RecetasLockJSON.json`.
[3] ***Hint:*** La lectura de `ingredientes.txt` está implementada en `def abrir_ingredientes(self)`.
[4] Esto se explica más adelante en la sección del modulo `backend.py`.
[5] `Comida` implementa `def date_a_str` y `def str_a_date` que convierten un `datetime` a (y desde) un *string*.

b) La fecha de ingreso al horno nos deja de interesar, por lo que no tienes que guardarla en la instancia.

Módulo `backend.py`

Todas las funciones mencionadas a continuación están dentro de la clase `PyKitchen` que modela todo el funcionamiento del programa. **Sólo debes editar las funciones mencionadas a continuación**:

1. `def cargar_recetas(self)`. Esta función debe cargar todas las recetas del **PKC** ubicadas en `recetas.book`, y agregarlas en el atributo `recetas` de `PyKitchen`.
2. `def guardar_recetas(self)`. Esta función debe guardar todas las recetas ubicadas en el atributo `recetas` del objeto `PyKitchen` en el archivo `recetas.book`.
3. `def cocinar(self)`. Esta función se encarga de preparar las recetas. Solo debes preparar recetas que hayan sido limpiadas, esto es, su *property* `verificada` devuelve `True`.

   Para preparar una receta basta con crear una comida a partir de esta con el método `Comida.de_receta(receta)`. Luego, cada comida debe guardarse como JSON en la carpeta `horno` con el formato `nombre_receta.json`. **Importante:** la carpeta `horno` debe existir en tu repositorio y no estar vacía[6].
4. `def despachar_y_botar(self)`. Esta función se encarga de sacar todas las comidas en el horno. Para ello debes hacer lo siguiente:
   a) Deserializar todas las comidas (en formato `json`) de la carpeta `horno`.
   b) Agregar al atributo `comidas` del objeto `PyKitchen` todas las comidas que no estén preparadas.
   c) Agregar al atributo `despachadas` todas las comidas que estén preparadas **pero no quemadas**.
   d) Debes imprimir los nombres de las comidas quemadas, pero no guardarlas.

Interfaz gráfica & tests

Se incluye en la actividad el archivo `main.py` para que la puedas probar. En la línea 10 puedes elegir entre tres modos:

- **Modo 0:** Correr el programa con consola.
- **Modo 1:** Correr el programa usando la GUI de PyKitchen.
- **Modo 2:** Correr un conjunto de tests unitarios.

Si alguno de los modos corre con errores, probablemente tengas un error en tu implementación. Sin embargo, lo contrario no es cierto: si el programa corre sin errores, aún es posible que hayas hecho algo mal o te falte algo, por lo que no hay garantía de máximo puntaje.

Notas

- La función `datetime.datetime.now()` entrega la fecha y hora actuales. También recuerda que puedes realizar sumas y restas de objetos `datetime` para obtener objetos `timedelta`, que repre-

[6]Se suele dejar un archivo `.keep` sin contenido para evitar que la carpeta quede vacía

sentan deltas de tiempo. Esta clase `timedelta` tiene un atributo `timedelta.seconds` y no tiene `timedelta.minutes`.

- Para abrir varios archivos en una carpeta puedes usar el módulo *built-in* `os`:
  - `os.listdir`
  - `os.path.join`

Requerimientos

- (3,25 pts) Implementación de los métodos en las clases `Receta`, `Comida` y `ComidaEncoder`
  - (2,0 pts) Clase Receta
    - (0,75 pts) Uso de `getstate` de `Recetas` en formato `pickle`.
    - (0,25 pts) Implementación `abir recetas_lock.json`.
    - (0,3 pts) Filtrado de atributos incorrectos utilizando el archivo `RecetasLOCK.json`.
    - (0,3 pts) Filtrado de ingredientes incorrectos en cada `Receta`.
    - (0,4 pts) Uso de `setstate` en `Recetas`.
  - (0,5 pts) Clase Comida
    - (0,25 pts) Modificación del init para utilizar el parámetro fecha ingreso.
    - (0,25 pts) Se evita el almacenamiento de la `fecha_ingreso` como atributo, *property*, o similar.
  - (0,75 pts) Clase Encoder
    - (0,75 pts) Implementación correcta, agregando atributo `fecha_ingreso`
- (2,75 pts) Implementación de funciones en `backend.py`
  - (1,0 pt) `def cocinar(self) -> None:`
  - (1,25 pts) `def despachar_y_botar(self) -> None:`
  - (0,25 pts) `def cargar_recetas(self) -> None.`
  - (0,25 pts) `def guardar_recetas(self) -> None.`

Entrega

- **Lugar:** En su repositorio privado de GitHub, en la **carpeta** `Actividades/AC12/`
- **Hora del último *push*:** 16:40