Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación
IIC2233 — Programación Avanzada 2022-2

18 de Agosto de 2022
Actividad Formativa

Actividad Formativa 1

Estructuras Built-In

Entrega

- **Lugar:** En su repositorio privado de GitHub, en la **carpeta** `Actividades/AF1/`
- **Hora del *push*:** 16:20

Actividad Formativa 1

Antes de comenzar...

Para esta, y todas las actividades del semestre, como equipo docente esperamos que sigas el siguiente flujo de trabajo:

1. Lee el enunciado completo, incluyendo las notas. Puedes revisar los otros archivos subidos a medida que lees, o al final, como te acomode.
2. Antes de comenzar a programar, copia todos los archivos de la carpeta AF1 del *Syllabus* y pégalos en la misma carpeta de **tu repositorio personal**.
3. Haz `git add`, `git commit` y `git push` de los archivos copiados inmediatamente, para comprobar que el uso de Git esté funcionando correctamente.
4. En caso de encontrar un error con Git, contacta a un ayudante para resolver el problema lo antes posible, en caso de no hacerlo, tu actividad podría no entregarse correctamente.
5. Comienza a trabajar en tu actividad en tu repositorio local y recuerda subir a GitHub cada vez que logres un avance significativo (con los mismos comandos de Git de antes).
6. Todas las actividades y tareas tienen una fecha y hora de entrega, en la cual automáticamente se recolecta el último *commit* ***pusheado*** en tu repositorio. Esto no quiere decir que solo se consideran los cambios de ese último *commit*, si no que todos los avances **hasta** ese *commit*. Luego, es importante que realices `git push` de todos tus avances, antes de la fecha y hora de entrega. En este caso, es a las 16:40 de hoy.

Vamos de la mano en el primer *push*

Haremos otro *commit* dentro del repositorio personal, como dijimos en el item anterior, es recomendable siempre subir los archivos al repositorio local apenas sean publicados.

1. En el explorador de archivor iremos a la carpeta `Actividades/AF1` del Syllabus.
2. Copiaremos todos los archivos desde esta carpeta a la carpeta `Actividades/AF1` de tu repositorio.

3. Abriremos la terminal o git bash en este directorio.
4. Revisa el estado del repositorio utilizando `git status`. Observa que los archivos recien agregados aparecerán bajo *"Untracked files"*.
5. Agrega los archivos `.py` al *staging area* mediante `git add`. Recuerda que debes escribir el nombre de los archivos después de este comando. Luego revisa el estado del repositorio y verifica que se haya agregado con `git status` nuevamente.
6. Utiliza `git commit -m "mensaje"` para crear un *commit* con los cambios realizados. Recuerda escribir un mensaje **descriptivo**. Luego revisa el estado del repositorio.
7. Ahora, utiliza `git push` para subir tus cambios en GitHub. Luego, revisa el estado del repositorio (`git status`) y después **ve el contenido del repositorio en un *browser* para verificar tus cambios**.

Introducción

Figura 1: Logo de DCCooked

Comenzando un nuevo año de presencialidad en la universidad y con la motivación en ascenso, para entretenerte en tus ventanas se te ocurrió navegar por redes sociales. ¿El único problema? No lograste conectarte a EDUROAM, así que deberás conformarte con juegos locales. Es sabido en el mundo de la

computación que los alumnos del IIC2233 son los mejores cuando de estos programas se trata, por ello, deberás desarrollar tu propio DCCooked para salvar tu aburrimiento al no tener internet.

Flujo del programa

DCCooked es un programa que te permite explorar un menú con diferentes platos de comida, los cuales puedes clasificar por categoría o filtrar según sus ingredientes. El programa cuenta con un menú, que se detalla en la sección Menú del programa, con el cual podrás acceder a las funcionalidades mencionadas y ordenar los exquisitos platos que preparamos para ti.

Para su implementación deberás utilizar las estructuras *built-in* de Python con el objetivo de definir las funciones de cargado de los datos, así como las consultas que se utilizan. Primero deberás crear funciones para **cargar la información de los archivos**, sobre los platos del menú y los ingredientes disponibles, en estructuras de datos apropiadas. Luego deberás **completar las funciones** que te permitirán realizar consultas sobre esos datos. Estas funciones las podrás probar utilizando el archivo `main.py`, donde los menúes y las llamadas a las funciones que debes completar ya vienen implementados.

Archivos

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

- `main.py`: No debes modificarlo Este es el archivo principal del programa. Puedes ejecutarlo para probar el funcionamiento de tu programa completo. **Ya viene implementado**.
- `cargar.py`: Debes modificarlo En este archivo están las bases de las funciones encargadas de cargar los datos. Deberás completarlas.
- `consultas.py`: Debes modificarlo En este archivo están las bases de funciones encargadas de consultar los datos. Deberás completarlas.
- `platos.csv`: No debes modificarlo En este archivo de texto se encuentran todos los platos que tiene el sistema. Cada línea sigue el siguiente formato:

  `nombre,categoria,tiempo_preparacion,precio,ingrediente_1;...;ingrediente_n`

  Donde `nombre`, `ingrediente_1,...,ingrediente_n` y `categoria` deben almacenarse como `str`, y `tiempo_preparacion` y `precio` como `int`.
- `ingredientes.csv`: No debes modificarlo En este archivo de texto se encuentran todos los ingredientes que se puedan utilizar y sus cantidades disponibles. Cada línea sigue el siguiente formato:

  `nombre_ingrediente,cantidad_disponible`

  Donde `nombre_ingrediente` debe almacenarse como `str` y `cantidad_disponible` como `int`.

Lectura de Archivos

Para poder leer los archivos deberás modificar las funciones presentes en el archivo `cargar.py`, sin embargo **no podrás hacer uso de clases para guardar esta información**. Las funciones son las siguientes:

- `def cargar_platos(ruta_archivo: str) -> list`: Esta función recibe la ruta de `platos.csv` y carga los platos al sistema. Debe retornar una lista de `namedtuples`, donde cada **namedtuple** contiene información de un plato. La lista retornada se debe ver así:

```
[
Plato(nombre=nombre1, categoria=categoria1, tiempo=t1, precio=p1, ingredientes={ing1, ...}),
Plato(nombre=nombre2, categoria=categoria2, tiempo=t2, precio=p2, ingredientes={ing2, ...}),
...
]
```

También debes asegurarte que los atributos de cada plato sean guardados como su tipo correspondiente, nombre como `str`, categoría como `str`, tiempo de preparación como `int`, precio como `int` y los ingredientes como una estructura que solo permita datos únicos y no repetidos. Ojo que los ingredientes están separados por un punto y coma `;`, mientras que toda la demás informacioń está separada por comas simples `,`.

- `def cargar_ingredientes(ruta_archivo: str) -> dict`: Esta función recibe la ruta de `ingredientes.csv` y carga los ingredientes disponibles en el sistema. Debe retornar un diccionario donde las llaves será el nombre del ingrediente y el valor será la cantidad disponible correspondiente, indicada en el archivo.

Consultas

La segunda parte de tu trabajo es implementar distintas consultas en el archivo `consultas.py`, para poder extraer información relevante de la base de datos entregada. En específico tendrás que implementar las siguientes consultas:

- `def platos_por_categoria(lista_platos: list) -> dict:`

  Deberás organizar los platos según su categoría[1] como un diccionario, donde cada llave es una categoría y su valor es una lista con todos los platos que pertenecen a esa categoría, un ejemplo seria:

```
{
  "categoría_1": [plato_x,..., plato_y],
  "categoria_2": [plato_r,..., plato_s],
  ...,
  "categoria_n": [plato_u,..., plato_v]
}
```

- `def descartar_platos(ingredientes_descartados: set, lista_platos: list) -> list`: Esta función recibe un set de ingredientes y una lista de platos (cada uno es una `namedtuple`). Se debe buscar todos aquellos platos **que no contengan** los ingredientes de `ingredientes_descartados` y retornar una lista de `platos`.
- `def resumen_orden(lista_platos: list) -> dict`: Esta función recibirá una lista de platos y deberás retornar un diccionario con el resumen de la orden. El diccionario debe tener las siguientes llaves y para sus respectivos valores debes calcular:
  - `"precio total"`: Precio total de la orden, corresponde a la suma de los precios de cada plato. Debe ser un `int`.
  - `"tiempo total"`: Tiempo total de preparación que corresponde a la suma de los tiempos de preparación de cada plato. Debe ser un `int`.

[1] Cada plato pertenece a solo una categoría.

- `"cantidad de platos"`: Cantidad total de platos a ordenar. Debe ser un `int`.
- `"platos"`: Es una lista **con los nombres de todos los platos** de la orden.

El diccionario a retornar **debe tener el siguiente formato**:

```
{
  "precio total": 12700,
  "tiempo total": 20,
  "cantidad de platos": 3,
  "platos": ["Hamburguesa", "Pizza", "Hot dog"]
}
```

Menú del programa

Ya que estás funciones por si solas no hacen mucho, se ha desarrollado una humilde interfaz para la terminal, donde ustedes podrán ir evaluando el avance con su actividad. Esta interfaz se mostrará al ejecutar el archivo `main.py` y ser verá así:

```
¡Hola bienvenido a Purble DCCPlace!
Tu orden actual es:


¿Qué deseas hacer?
[1] Descartar ingredientes
[2] Preparar plato
[3] Terminar mi orden

[0] Salir sin preparar un plato
```

Una vez que hayas completado las funciones de `cargar.py` recién podrás acceder a las funcionalidades, antes te aparecerá un error.

Mediante el primer menú `[1]` podrás probar y hacer uso de la función `descartar_platos()` luego de seleccionar algunos ingredientes. Esto filtrará los platos según lo que ingreses, por ejemplo, si descartas `Tomate` al ingresar al menú `[2]`[2] no deberían aparecer las categorías `Hot Dog` ni `Hamburguesa`.

Al ingresar al segundo menú `[2]` te encontrarás con las categorías disponibles, por lo cual, podrás comprobar la implementación de la función `ordenar_por_categoria()`, si implementaste correctamente esto podrás seleccionar una categoría y ver los platos correspondientes, por ejemplo para la categoría `Hamburguesa` se verá así:

```
Selecciona un plato de Hamburguesa:
[1] Hamburguesa con queso
[2] Hamburguesa DCC

[0] Volver
```

En este caso al seleccionar un plato, podrás probar la implementación de la función `preparar_plato()`.

Finalmente con el menú `[3] Terminar mi orden` podrás comprobar la implementación de `resumen_orden()`.

[2] Tienes que completar la función `separar_por_categoria() primero`.

Notas

- Se agregaron `print` en el código base para que puedas revisar tu avance.
- Si aparece un error inesperado, ¡léelo! Intenta interpretarlo.
- Siéntete libre de agregar nuevos `print` en cualquier lugar para revisar objetos, modificaciones, etc... Es una herramienta muy útil para encontrar errores.
- No debes utilizar clases para resolver la actividad, con las estructuras *built-in* de Python es suficiente.

Objetivos

- Implementar distintos tipos de estructuras *built-in* según su corresponda.
- Reconocer las ventajas entre las estructuras *built-in* de Python.
- Utilizar elementos de la *standard library* de Python.