



17 de Marzo 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:** 16:40

### Parte 1: Bienvenid@ a IIC2233

Bienvenid@ a tu primera actividad en IIC2233. El objetivo de esta primera parte es entender cuáles son las principales páginas relacionadas con el curso. Los requisitos para empezar son:

1. Haber instalado Git en tu computador (más info en [este enlace](#)). Si tienes MacOS o Linux, muy probablemente ya lo tendrás instalado.
2. Haberte registrado en GitHub y en el curso en el formulario a través de la ([AvanzadaApp](#)).
3. Una vez registrado, debió llegarte un correo de GitHub con una invitación a un repositorio en la organización del curso (revisa bien tu correo utilizado para la cuenta de GitHub).

#### 1.1 *Syllabus* ([syllabus](#))

El *syllabus* es un **repositorio** de GitHub donde, como equipo docente, subiremos todos los enunciados de las tareas, actividades y ayudantías. Puedes ver el *syllabus* en [github.com/IIC2233/syllabus](https://github.com/IIC2233/syllabus).

El *syllabus* tiene asociado un foro de *issues*. En este foro puedes preguntar sobre la materia o algo sobre los enunciados de las tareas. También los ayudantes podrán colocar avisos importantes, ya sean detalles administrativos o aclaraciones de enunciados, **por lo que es tu deber prestar atención constante al contenido de este foro**.

IIC2233 / syllabus

Watch 10 Star 6 Fork 2

Code Issues 3 Pull requests 0 Wiki Insights Settings

Repositorio oficial del curso IIC2233 Programación Avanzada

6 commits 1 branch 0 releases 2 contributors

Figura 1: Llegar al foro a través del *syllabus*.

## 1.2. Repositorio de apuntes (**contenidos**)

Los apuntes con los contenidos del curso se encontrarán en un repositorio llamado **contenidos**, al que puedes llegar haciendo clic en un vínculo ubicado en la página principal del curso, o yendo directamente a [github.com/IIC2233/contenidos](https://github.com/IIC2233/contenidos).

## 1.3. Tu repositorio personal

Luego de registrarte en el curso, se te ha creado un repositorio **personal y privado** donde deberás trabajar y entregar tus actividades y tareas. Este se ubica en [github.com/IIC2233/usuario-iic2233-2022-1](https://github.com/IIC2233/usuario-iic2233-2022-1), donde debes reemplazar **usuario** por tu usuario de GitHub.

Tu repositorio también tiene asociado un foro de *issues*. En él recibirás —en forma **automatizada**— el detalle de la corrección de tus actividades y tareas. **NO respondas los mensajes ahí, puesto que nadie leerá lo que escribas.** Si tienes dudas sobre tu corrección usa el correo del curso o solicita una recorrección mediante el formulario indicado.

The screenshot shows a GitHub repository page for 'IIC2233/igbasly-iic2233-2022-1'. The repository is private and was generated from 'IIC2233/student-repository-template-2022-1'. The main tab is selected, showing the 'main' branch with 1 branch and 0 tags. The commit history lists four commits by 'igbasly': 'Update README.md' (Initial commit, 9 days ago), 'Actividades' (Initial commit, 9 days ago), 'Tareas' (Initial commit, 9 days ago), and '.gitignore' (Initial commit, 9 days ago). The 'Code' button is highlighted. On the right, there's an 'About' section with details about the repository owner and its purpose. Below the code area, there's a sidebar with links to 'Readme', 'stars', 'watching', and 'forks'.

Figura 2: Ejemplo de repositorio personal

The screenshot shows the 'Issues' tab of the same GitHub repository. There are 2 open issues. The search bar shows 'is:issue is:open'. The issues listed are '#2 opened now by igbasly' (labeled 'AF2') and '#1 opened 6 days ago by igbasly' (labeled 'AS1'). The 'Issues' button is highlighted. The top navigation bar includes 'Code', 'Issues', 'Pull requests', 'Actions', 'Projects', 'Wiki', 'Security', 'Insights', and 'Settings'.

Figura 3: *Issues* del repositorio personal.

## Parte 2: Ahora sí, bienvenid@ a IIC2233

El objetivo de esta parte es tener un primer contacto con **git**, el sistema de control de versiones que se utiliza en este curso.

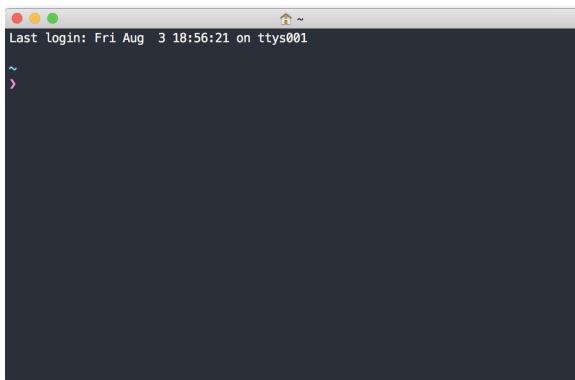
## Aprender lo básico de la terminal

Necesitamos aprender a navegar por las carpetas del computador usando la consola. En esta parte de la guía, sigue todos los pasos y verifica que te sale lo mismo que está aquí.

Para abrir la consola:

- **macOS o Linux:** busca el programa **Terminal** o similar.
- **Windows:** busca el programa **Git Bash**. Este programa es una consola que además implementa algunos de los comandos que podrías encontrar en Linux, por lo que es mucho más amigable que el “Símbolo del Sistema”.

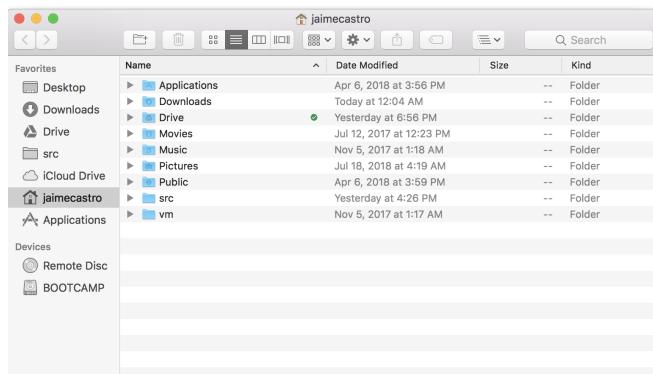
La consola en todo momento está ubicada en una carpeta (o directorio). Al abrir la consola, ésta se abre en la carpeta de tu computador que contiene tus datos. Esta carpeta suele llamarse `home` o “`~`”. Dentro de ella, suele encontrarse el escritorio o la carpeta de documentos.



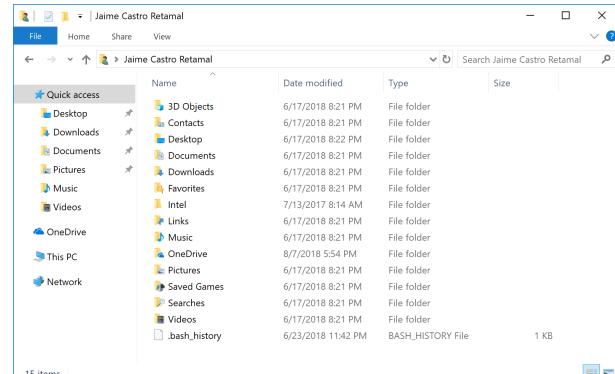
(a) Consola abierta en macOS



(b) Consola abierta en Windows



(a) “Home” en macOS



(b) “Home” en Windows

Muchas veces las consolas muestran en qué parte están ubicadas, pero el comando `pwd` también nos entrega esa información. Escríbelo en tu consola y apreta `enter`.

```
Last login: Fri Aug 3 18:56:21 on ttys001
~
> pwd
/Users/jaimecastro
~
```

(a) `pwd` en macOS estando en el “*home*”

```
Jaime Castro Retamal@DESKTOP-8B6SC33 MINGW64 ~
$ pwd
/c/users/Jaime Castro Retamal
Jaime Castro Retamal@DESKTOP-8B6SC33 MINGW64 ~
$ |
```

(b) `pwd` en Windows estando en el “*home*”

También podemos saber qué es lo que contiene el directorio actual, con el comando `ls`.

```
Last login: Fri Aug 3 18:56:21 on ttys001
~
> pwd
/Users/jaimecastro
~
> ls
Applications Documents Drive Movies Pictures src
Desktop Downloads Library Music Public vm
~
```

(a) `ls` en el “*home*” en macOS

```
'My Documents'@
NetHood@
NTUSER.DAT
ntuser.dat.LOG1
ntuser.dat.LOG2
NTUSER.DAT{ab042e11-728c-11e8-8d38-8479ef28c3e1}.TM.blf
NTUSER.DAT{ab042e11-728c-11e8-8d38-8479ef28c3e1}.TMContainer000000000000000000000000
1.retrans-ms
NTUSER.DAT{ab042e11-728c-11e8-8d38-8479ef28c3e1}.TMContainer000000000000000000000000
2.retrans-ms
ntuser.ini
OneDrive/
Pictures/
PrintHood@
Recent@
'Saved Games'/
Searches/
SendTo@
'Start Menu'@
Templates@
Videos/
Jaime Castro Retamal@DESKTOP-8B6SC33 MINGW64 ~
$ |
```

(b) `ls` en el “*home*” en Windows

Supongamos que queremos mover la ubicación de la consola a la carpeta “*Desktop*” (Escritorio). Para ello, usamos el comando `cd` (*change directory*). En este caso, debemos teclear `cd Desktop`. Puedes verificar que cambiaste de directorio utilizando `pwd` nuevamente.

```
Last login: Fri Aug 3 18:56:21 on ttys001
~
> pwd
/Users/jaimecastro
~
> ls
Applications Documents Drive Movies Pictures src
Desktop Downloads Library Music Public vm
~
> cd Desktop
~/Desktop
~
```

(a) `cd Desktop` en macOS

```
ntuser.dat.LOG1
ntuser.dat.LOG2
NTUSER.DAT{ab042e11-728c-11e8-8d38-8479ef28c3e1}.TM.blf
NTUSER.DAT{ab042e11-728c-11e8-8d38-8479ef28c3e1}.TMContainer000000000000000000000000
1.retrans-ms
NTUSER.DAT{ab042e11-728c-11e8-8d38-8479ef28c3e1}.TMContainer000000000000000000000000
2.retrans-ms
ntuser.ini
OneDrive/
Pictures/
PrintHood@
Recent@
'Saved Games'/
Searches/
SendTo@
'Start Menu'@
Templates@
Videos/
Jaime Castro Retamal@DESKTOP-8B6SC33 MINGW64 ~
$ cd Desktop
Jaime Castro Retamal@DESKTOP-8B6SC33 MINGW64 ~/Desktop
$ |
```

(b) `cd Desktop` en Windows

Si nos queremos devolver a la carpeta que contiene a “*Desktop*”, usamos `cd ..` (los dos puntos significan “la carpeta que contiene la actual”).

```

Last login: Fri Aug 3 18:56:21 on ttys001
~
> pwd
/Users/jaimecastro

~
> ls
Applications Documents Drive Movies Pictures src
Desktop Downloads Library Music Public vm

~
> cd Desktop
~/Desktop
> cd ..
~

> |

```

(a) `cd ..` en macOS

```

MINGW64/c/Users/Jaime Castro Retamal
NTUSER.DAT{ab042e11-728c-11e8-8d38-8479ef28c3e1}.TMContainer000000000000000000000000
1. regtrans-ms
NTUSER.DAT{ab042e11-728c-11e8-8d38-8479ef28c3e1}.TMContainer000000000000000000000000
2. regtrans-ms
ntuser.ini
OneDrive/
Pictures/
PrintHood
Recent@ 'Saved Games'/
Searches/
SendTo@ 'Start Menu'@ Templates@ Videos/
Jaime Castro Retamal@DESKTOP-8B6SC33 MINGW64 ~
$ cd Desktop
Jaime Castro Retamal@DESKTOP-8B6SC33 MINGW64 ~/Desktop
$ cd ..

Jaime Castro Retamal@DESKTOP-8B6SC33 MINGW64 ~
$ |

```

(b) `cd ..` en Windows

## Clonar repositorio personal

En esta parte, clonaremos tu repositorio personal para que puedas entregar tus actividades y tareas.

1. Asegúrate de que la terminal esté dentro de la carpeta donde quieras tener tu repo, como el escritorio o alguna carpeta propia. Si no, navega usando los comandos mencionados hasta encontrarlo.
2. Ve a la página de tu repositorio y copia el vínculo que permite clonar el repositorio (busca el botón verde que resalta y que muestra la siguiente foto).
3. En la consola, ejecuta el comando para clonarlo: `git clone url_copiada`.
4. Deberías ver que se creó la carpeta con el contenido de tu repositorio personal<sup>1</sup>.

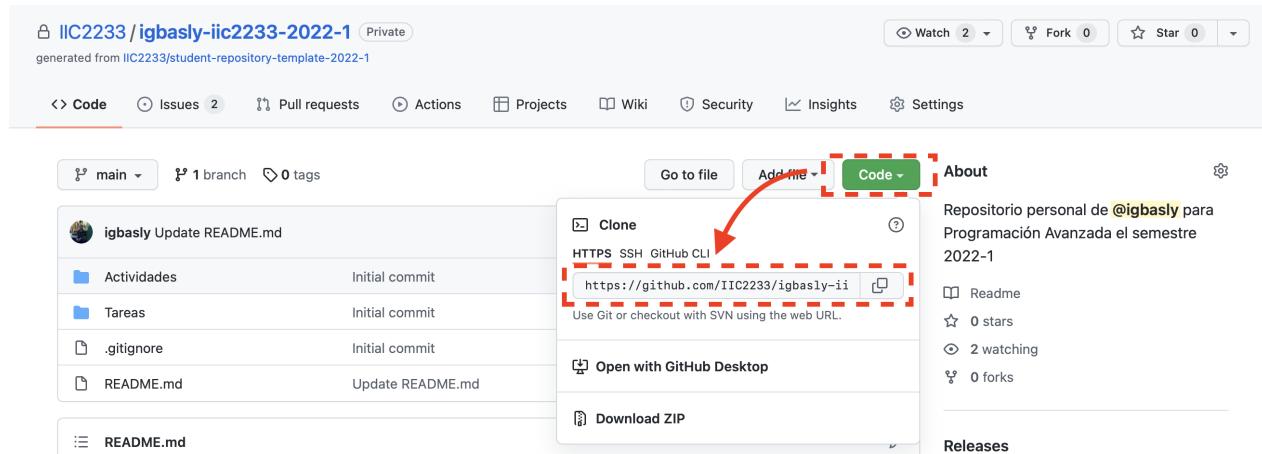


Figura 10: Como encontrar enlace de repositorio.

En caso de que ya hayas copiado tu repositorio personal en un lugar que no te guste, simplemente puedes mover la carpeta completa a un directorio que prefieras, los archivos que permiten que `git` funcione se encuentran dentro de tu carpeta y seguirán funcionando normalmente.

## Clonar otros repositorio importantes del curso

Aprovechando el vuelo, repite los mismos pasos del punto anterior pero para `syllabus` y para `contenidos`, ya que tendrás que estar pendiente de estos repositorios durante el semestre.

<sup>1</sup>Si no sabes dónde se creó, recuerda usar el comando `pwd` para ver la ubicación actual de tu consola.

## El primer *add*, *commit* y *push*

Haremos nuestro primer *commit* dentro del repositorio personal. Para trabajar en él, tu consola deberá estar **dentro** de la carpeta del repositorio (podrías usar el comando `cd`).

1. Dentro del repositorio, hay un archivo llamado `README.md` creado. En este, hay información sobre el uso de tu repositorio, y dejamos espacios para que rellenes con tus datos. Abre el archivo en tu computador con algún editor de texto y rellena los espacios con tus datos.
2. Ahora vuelve a la consola. Revisa el estado del repositorio utilizando `git status`. Observa que tu archivo recién editado aparece listado como “*modified*” en “*Changes not staged for commit*”.
3. Agrega el archivo `README.md` al *staging area* mediante el comando: `git add README.md`.
4. Revisa el estado del repositorio (`git status`) y verifica que tu archivo recién editado aparece listado como “*modified*” en “*Changes to be committed*”.
5. Utiliza: `git commit -m "README actualizado"`. El texto entre comillas se conoce como mensaje del *commit* y debe describir lo que fue agregado mediante `git add`.
6. Vuelve a revisar el estado (`git status`) del repositorio nuevamente. No deberías ver listado a `README.md`, pero si un mensaje que dice “*Your branch is ahead of ‘origin/main’ by 1 commit.*”, lo cual significa que el *commit* fue creado correctamente,
7. Ahora, para escribir tus cambios en GitHub utiliza: `git push`.
8. Vuelve a revisar el estado del repositorio, debe decir: *Your branch is up to date with ‘origin/main’ nothing to commit, working tree clean*.
9. Despues ve el contenido del repositorio en un navegador web (GitHub) y verifica tus cambios. Si los ves, ¡lo lograste!

**IMPORTANTE:** Cuando escribas el `README.md` de tus tareas, debes seguir este mismo proceso, **no lo edites desde el navegador**, porque eso generará conflictos entre la versiones del archivo de tu computador y la versión del repositorio

## Parte 3: Actualizar tus repos locales

Ya hablamos de como agregar cambios locales (computador) y enviarlos a la versión remota tu repositorio (GitHub), pero no hemos hablado de como **traer cambios** del repositorio remoto a nuestro repositorio local.

Lo anterior, asume que posterior a que clonamos un repositorio, como `syllabus` o `contenidos`, se hayan subido cambios que queremos ver en nuestro repositorio local. **La solución no es clonar cada vez, si no que actualizar la versión ya clonada.** Para eso es: `git pull`.

Hoy (jueves 17) a las 15:00 se subirán nuevos archivos al *Syllabus* y deberás actualizarlo para poder completar la actividad, sin embargo si clonaste el repositorio luego de esa hora, este paso no será necesario, por lo que puedes pasar directamente a la **Parte 4**.

### Actualizar *syllabus*

A continuación actualizaremos los contenidos del repositorio del curso `syllabus`. Para lograrlo, tu consola deberá estar ubicada dentro de la carpeta del repositorio.

- Primero, revisa el contenido de `syllabus` en tu computador, notarás que no hay muchos archivos, solo podrás ver este enunciado.

- Debemos esperar a que los ayudantes actualicen el repositorio *syllabus* del curso. Puedes revisar en [github.com/IIC2233/syllabus](https://github.com/IIC2233/syllabus) en el navegador. Si ves que hay archivos de extensión .py y otros de extensión .csv en la carpeta AF1, significa que ya es tiempo de actualizar tu repositorio local.
- Antes, en consola, revisa el estado del repositorio usando `git status`.
- Ahora, ejecuta el comando `git pull`. Este último debería descargar cambios que fueron subidos por los ayudantes a este repositorio.
- Si ves el contenido en tu computador nuevamente, te encontrarás con algunos nuevos archivos como `main.py` y `platos.csv`, lo que significa que puedes comenzar con la actividad.

## Parte 4: 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:

- Lee el enunciado completo, incluyendo las notas. Puedes revisar los otros archivos subidos a medida que lees, o al final, como te acomode.
- 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**.
- Haz `git add`, `git commit` y `git push` de los archivos copiados inmediatamente, para comprobar que el uso de Git esté funcionando correctamente.
- 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.
- 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).
- 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 ítem 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 recién 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**.

## AF1: Introducción



Figura 11: Logo de Purble DCCPlace

Comenzando un nuevo año de presencialidad en la universidad y con la motivación en ascenso, para entretenerse 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 Purble DCCPlace para salvar tu aburrimiento al no tener internet.

## Flujo del programa

Purple DCCPlace 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 menús 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=categoría1, tiempo=t1, precio=p1, ingredientes={ing1, ...}),  
Plato(nombre=nombre2, categoria=categoría2, 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 información 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 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<sup>2</sup> 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 sería:

```
1  {
2      "categoría_1": [plato_x, ..., plato_y],
3      "categoría_2": [plato_r, ..., plato_s],
4      ...,
5      "categoría_n": [plato_u, ..., plato_v]
6 }
```

- `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`.
  - `"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**:

```
1  {
2      "precio total": 12700,
3      "tiempo total": 20,
4      "cantidad de platos": 3,
5      "platos": ["Hamburguesa", "Pizza", "Hot dog"]
6 }
```

## 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:

---

<sup>2</sup>Cada plato pertenece a solo una categoría.

¡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]<sup>3</sup> 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()`.

## 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.

---

<sup>3</sup>Tienes que completar la función `separar_por_categoria()` primero.