Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

IIC2233 Programación Avanzada (2025-2)

Tarea 3

Entrega

- **Tarea y** `README.md`
  - **Fecha y hora oficial (sin atraso):** lunes 27 de octubre de 2025, 20:00.
  - **Fecha y hora máxima (2 días de atraso):** miércoles 29 de octubre de 2025, 20:00.
  - **Lugar:** Repositorio personal de GitHub — Carpeta: `Tareas/T3/`.
    El código debe estar en la rama (*branch*) por defecto del repositorio: `main`.
  - **Pauta de corrección**: en este enlace.
  - **Bases generales de tareas (descuentos)**: en este enlace.
  - **Formulario entrega atrasada:** en este enlace. Se cerrará el miércoles 29 de octubre, 23:59.
- **Ejecución de tarea**: La tarea será ejecutada **únicamente** desde la terminal del computador. Además, durante el proceso de corrección, se cambiará el nombre de la carpeta "`T3/`" por otro nombre y se ubicará la terminal dentro de dicha carpeta antes de ejecutar la tarea.

Objetivos

- Entender y aplicar el paradigma de programación funcional para resolver un problema.
- Manejar datos de forma eficiente utilizando herramientas de programación funcional:
  - Uso de generadores y funciones generadoras.
  - Uso de `map`, `lambda`, `filter`, `reduce`, etc.
  - Uso de estructuras por comprensión.
  - Uso e investigación de las librerías `itertools` y `collections`.
- Utilizar conceptos de interfaces y `PyQt5` para implementar una aplicación gráfica e interactiva. Además en esta se debe entender y aplicar los conceptos de *back-end* y *front-end*.
- Aplicar conocimientos de señales.

Índice

1. *Departamento de las Colecciones del Cosmos*

En el vasto cosmos, dos gigantescas corporaciones mineras libran una guerra fría por los recursos: el Imperio Minero Celestial (IMC), dirigido con puño de hierro por el frío y calculador C. Morayoshi, quien exige eficiencia y pureza algorítmica absolutas; y el *Departamento de las Colecciones del Cosmos* (DCC), liderado por el astuto y pragmático Mr. Aransene Lupablo, quien valora el ingenio práctico por sobre los protocolos rígidos. Tras ser injustamente despedidos del IMC por "priorizar soluciones creativas sobre protocolos", tú y tus compañeros son reclutados por Herskowitz. Para financiar la operación y demostrar su superioridad, será necesario extraer minerales de diferentes planetas en distintas expediciones, allanando el camino hacia el objetivo final.

La carrera se intensifica con el rumor del legendario *Gatochico de Platino*. Mientras Reed moviliza toda la maquinaria del IMC para calcular la ruta perfecta, Herskowitz apuesta por vuestro talento para aprovechar cada cargamento de minerales y hackear atajos imposibles. Cada expedición de extracción no solo es una misión de recursos, sino un paso crucial en esta competencia personal entre dos visiones opuestas, donde el destino de ambas compañías se decide entre el orden implacable de Reed y la audacia ingeniosa que Herskowitz ha depositado en su nuevo equipo.

¿Serás capaz de probar con tu ingenio que el **DCC** supera a la competencia?

Figura 1: Imagen del legendario *Gatochico de Platino* sostenido por su descubridora

2. Flujo del programa

En *Departamento de las Colecciones del Cosmos*, el principal objetivo será poder realizar consultas en torno a distintas bases de datos que tendrás disponibles. Esta tarea comenzará con funciones para acceder a archivos de distintos tamaños. Tu objetivo en esta sección será almacenar la información de los archivos en `namedtuples` para poder utilizarla más adelante. Luego, utilizarás tus habilidades de programación funcional y generadores para realizar consultas, las cuales darán información específica sobre las distintas naves, planetas, misiones, minerales o astronautas. Por último, utilizando los contenidos de interfaces gráficas, deberás crear en total 3 ventanas: una ventana de entrada que da la bienvenida al programa, otra ventana principal para elegir de manera interactiva las consultas, y por último una para visualizar los resultados.

La realización de la tarea se dividirá en dos partes principales:

1. En la Sección 4: Programación Funcional, se pedirá que implementes las funciones mencionadas más adelante. Para que el código sea ordenado, estas funciones ya están declaradas en los archivos que te entregamos. Es muy importante que **no le cambies el nombre a la función y que no modifiques los argumentos recibidos**. Además de los archivos entregados por nosotros, está permitido crear nuevos archivos y/o crear otras funciones si lo estimas conveniente.

   Esta sección será evaluada mediante el uso de *tests*. Para poder asegurarse de la buena realización de las funciones, se podrá utilizar una serie de *tests* públicos. Es sumamente importante que **no realices la tarea basándote en los *tests* públicos**. Te debes basar en el enunciado y sólo puedes usar los *tests* públicos como apoyo. Debido a esto, es importante destacar que **si el alumno tiene los test públicos correctos, pero los *tests* privados incorrectos, no se otorgará puntaje**. En la Sección 6: *Tests* se explican los distintos tipos de *tests* que presenta la tarea y cómo ejecutarlos.

2. En la Sección 5: Interfaz Gráfica e Interacción, se explica como implementar la parte visual del programa, el cual será desarrollada utilizando los contenidos de interfaces gráficas aprendidos en el curso. Deberás implementar las 3 ventanas ya mencionadas.

   Esta parte de la tarea será corregida manualmente, por lo que es sumamente importante que **el código esté ordenado para facilitar el entendimiento**.

3. Archivos

Aunque -como se menciono anteriormente- puedes crear los archivos que quieras, en junto a este enunciado se encuentran los siguientes archivos y directorios base:

- Modificar `main.py`: Archivo, inicialmente vacío donde se instancian y conectan los distintos componentes del *frontend* y *backend*, permitiendo así ejecutar el programa.
- Modificar `backend/`: Directorio que contiene todo lo asociado al *backend* del programa. Adicionalmente, contiene el siguiente archivo:
  - Modificar `consultas.py`: Archivo que contiene las consultas del programa. Las funciones a implementar se explican en la Sección 4: Programación Funcional.
- Modificar `frontend/`: Directorio que contiene todo lo asociado al *frontend* del programa, es decir, todo lo relacionado a la interfaz gráfica.
- No modificar `data/`: Directorio que contiene una serie de archivos CSV necesarios para ejecutar la tarea. Dentro de esta carpeta habrá tres sub-carpetas (`S`, `M`, `L`) que contendrán CSV de distintos tamaños. En un principio esta carpeta se encontrará vacía, debido a que los distintos archivos se deben descargar y ordenar desde **este link**.

- No modificar `tests_publicos/`: Directorio que contiene los tests públicos de la tarea. Estos se encargarán de revisar lo implementado en `consultas.py`.

  Importante: los *tests* entregados en esta carpeta no serán los mismos que se utilizarán para la corrección de la evaluación.

- No modificar `utilidades.pyc`: Archivo que contiene las `namedtuples` a utilizar, junto a funciones útiles para el desarrollo de la tarea. Este archivo ya se encuentra implementado.

Los archivos y directorios listados anteriormente se deberán organizar de la siguiente manera dentro de la carpeta `T3/` una vez que se haya descargado toda la información necesaria:

```
T3/
├── backend/
│   ├── consultas.py
│   └── ...
├── frontend/
│   ├── sprites/
│   │   ├── plasma.png
│   │   └── ...
│   └── ...
├── data/
│   ├── S/
│   ├── M/
│   └── L/
├── tests_publicos/
│   ├── solution/
│   │   ├── test_1.py/
│   │   ├── test_2.py/
│   │   └── ...
│   ├── test_000_namedtuples_carga_de_datos
│   ├── test_00_naves_de_material_carga_de_datos.py
│   └── ...
├── utilidades.pyc
├── main.py
├── README.md
└── .gitignore
```

4. Programación Funcional

Para poder analizar a la gran cantidad de datos que posee el *Departamento de las Colecciones del Cosmos*, se necesitará de tu ayuda experta para procesar la información mediante la realización de diversas consultas, por lo que deberás aplicar tus conocimientos de programación funcional.

Para esto, deberás interactuar con distintos tipos de datos, los que serán explicados con mayor detalle en la Subsección 4.1: Datos y completar distintas funciones pedidas en la Subsección 4.2: Carga de datos de `namedtuples` y la Subsección 4.3: Consultas, que se realizará mediante la modificación de un código pre-existente.

4.1. Datos

Para poder interactuar con los datos entregados por los clientes, tendrás que leer los archivos CSV y procesar su contenido. Para poder manejar el contenido de estos archivos se te entregan las `namedtuples` del archivo `utilidades.pyc`.

A modo general, los datos pueden presentar atributos que seguirán un formato en específico:

- **Id:** Los datos contarán con identificadores únicos (id) que permitirán distinguir y asociar los datos. El formato de estos ids consitirán en un *integer* positivo.
- **Fecha:** Cada atributo que haga relación a una fecha consistirá en un *string* con el siguiente formato `"YYYY-MM-DD"` (`AÑO-MES-DÍA`).

A continuación se detallan los distintos tipos de datos y sus respectivas `namedtuples`:

4.1.1. `Astronauta`

Indica la información de cada Astronauta. Posee los atributos:

Atributo	Tipo	Descripción	Ejemplo
`id_astronauta`	`int`	Permite distinguir e identificar a cada usuario.	`99`
`nombre`	`str`	Nombre y apellido del astronauta, separado por un espacio.	`"Jacqueline Smith"`
`estado`	`str`	Indica el estado del astronauta.	`"Activo"`

Estos datos están ordenados según su `id_astronauta`.

4.1.2. `Nave`

Indica la información de cada nave. Posee los atributos:

Atributo	Tipo	Descripción	Ejemplo
`patente`	`str`	Permite distinguir e identificar una nave.	`"N-106"`
`material`	`str`	Indica el material principal del cual esta hecha la nave.	`"Titanio"`
`tamano`	`str`	Representa el tamaño de la nave.	`"XL"`
`capacidad_astronautas`	`int`	Entero que indica la capacidad máxima de astronautas en la nave.	`7`
`capacidad_minerales`	`float`	Decimal que indica cuantas toneladas de mineral puede llevar la nave.	`807.63`
`autonomia`	`float`	Indica la distancia que puede recorrer una nave medido en unidades astronómicas (UA).	`703.81`

Debido a la falta de instrumentos especializados, los minerales se almacenan en su forma impura dentro la nave considerando `capacidad_minerales`. Es decir, no se considera la pureza del mineral, sino que la cantidad total impura.

4.1.3. `Tripulacion`

Hace referencia a la relación entre un astronauta y una nave. Cada astronauta esta relacionado con una sola nave, mientras que cada nave puede estar relacionada a múltiples astronautas. Los grupos de astronautas relacionados a una nave se le llama Equipo, y cada nave posee un solo equipo asignado. Cada entrada de tripulación contiene los atributos:

Atributo	Tipo	Descripción	Ejemplo
`id_equipo`	`int`	Permite distinguir e identificar a un grupo de astronautas que poseen una nave asignada.	`617`
`patente_nave`	`str`	Permite distinguir e identificar la nave del equipo.	`"N-110"`
`id_astronauta`	`int`	Permite distinguir e identificar al astronauta.	`501`
`rango`	`int`	Entero entre 1 y N que representa el rango del astronauta dentro del equipo. Un número menor representa un rango menor.	

Las entradas de Tripulación están ordenadas según su `patente_nave`.

4.1.4. `Planeta`

Indica la información de un planeta. Posee los atributos:

Atributo	Tipo	Descripción	Ejemplo
`id_planeta`	`int`	Permite distinguir e identificar a un planeta.	`781`
`nombre`	`str`	Nombre del planeta.	`"P-348"`
`coordenada_x`	`float`	Coordenada X del planeta.	`-16850.91`
`coordenada_y`	`float`	Coordenada Y del planeta.	`2350.14`
`tamano`	`str`	Indica el tamaño del planeta.	`"L"`
`tipo`	`str`	Indica el tipo de planeta que es.	`"Rocoso"`

Las entradas de Planeta están ordenadas según su `id_planeta`.

4.1.5. `Mineral`

Indica la información de un mineral. Posee los atributos:

Atributo	Tipo	Descripción	Ejemplo
`id_mineral`	`int`	Permite identificar unicamente a un mineral.	`529`
`nombre`	`str`	Nombre del mineral.	`"Min-HDK"`
`simbolo_quimico`	`str`	Simbolo quimico del mineral.	`"Jw"`
`numero_atomico`	`int`	Número atomico del mineral.	`118`
`masa_atomica`	`float`	Masa atómica del mineral en g/mol.	`293.769`

4.1.6. `PlanetaMineral`

Indica la relación entre un planeta y un mineral, es decir los minerales que se encuentran en cada planeta. Cada entrada representa un mineral presente en un planeta y posee los atributos:

Atributo	Tipo	Descripción	Ejemplo
`id_planeta`	`int`	Identificador del planeta.	`285`
`id_mineral`	`int`	Identificador del mineral.	`654`
`cantidad_disponible`	`float`	Indica las toneladas de mineral disponible en el planeta. Esta representa el total impuro del mineral, es decir, como se encuentra en su estado natural.	`975971.505`
`pureza`	`float`	Decimal entre 0.05 y 0.99 que indica el nivel de pureza del mineral en el planeta.	`0.735`

Puedes asumir que el par `id_planeta, id_mineral` estará a lo más una vez en los datos, es decir, no está dos veces el mismo mineral en el mismo planeta, pero con distinta pureza.

4.1.7. `Mision`

Indica la información correspondiente a cada misión. Posee los atributos:

Atributo	Tipo	Descripción	Ejemplo
`id_mision`	`int`	Identificador de la misión.	`582`
`fecha`	`str`	Fecha de inicio de la misión.	`"2024-03-02"`
`hora`	`str`	Hora de inicio de la misión en formato HH:MM.	`"22:30"`
`id_equipo`	`int`	Identificador del equipo al que le es asignada la misión.	`735`
`id_planeta`	`int`	Identificador del planeta objetivo la misión.	`259`
`lograda`	`bool | None`	Indica el estado de la misión: • `True` = Logró su objetivo • `False` = No logró su objetivo • `None` = No realizada aun Una misión se considera realizada si es que esta fue lograda o no lograda (`lograda` es `True` o `False`).	`True`

4.1.8. `MisionMineral`

Indica los minerales requeridos para cada misión. Cada entrada representa un mineral requerido para una misión, cada misión puede requerir más de un mineral. Posee los atributos:

Atributo	Tipo	Descripción	Ejemplo
`id_mision`	`int`	Identificador de la misión.	`582`
`id_mineral`	`int`	Identificador del mineral.	`991`
`cantidad`	`float`	Indica las toneladas de material **puro** requeridas por la misión.	`4636.117`

Las entradas de MisiónMineral están ordenadas según su `id_mineral`.

4.2. Carga de datos de `namedtuples`

En la carpeta "`data/`" podrás encontrar tres sub-carpetas (`S`, `M` y `L`) con bases de datos de distintos tamaños: pequeños, medianos y grandes, respectivamente. Deberás copiar esta carpeta y ubicarla en tu carpeta de la tarea `T3/`, siguiendo la estructura indicada en Sección 2: Flujo del programa.

Para poder trabajar las consultas de esta tarea deberás cargar la información en **generadores** que contengan las `namedtuple` definidas en `utilidades.pyc`. La información de estos **generadores** se obtendrá a partir de archivos de extensión `.csv` que siguen el mismo formato de las `namedtuple` mencionadas en la sección anterior. Para asegurar el correcto funcionamiento de estas funciones durante la corrección de la tarea y ejecución de los tests, las funciones no deben modificar los paths recibidos de estas funciones.

La carga de información requiere que implementes las siguientes funciones que definen **generadores**. Cada función recibirán un `str` **`path`** correspondiente a la ruta del archivo `.csv` que le corresponde y deberá retornar un **generador** con instancias de la `namedtuple` usando la información del archivo.

Modificar
```python
def cargar_astronautas(path: str) -> Generator[Astronauta]
```

Modificar
```python
def cargar_naves(path: str) -> Generator[Nave]
```

Modificar
```python
def cargar_tripulaciones(path: str) -> Generator[Tripulacion]
```

Modificar
```python
def cargar_planetas(path: str) -> Generator[Planeta]
```

Modificar
```python
def cargar_minerales(path: str) -> Generator[Mineral]
```

Modificar
```python
def cargar_planeta_minerales(path: str) -> Generator[PlanetaMineral]
```

Modificar
```python
def cargar_mision(path: str) -> Generator[Mision]
```

Modificar
```python
def cargar_materiales_mision(path: str) -> Generator[MisionMineral]
```

Esta tarea contendrá *tests* que evaluarán el óptimo de la solución implementada, por lo que estas funciones serán usadas para cargar y crear los generadores que recibirán las consultas definidas en la siguiente sección. Por lo tanto, se recomienda **completar primero estas funciones antes de completar y probar cualquier consulta**.

4.3. Consultas

Utilizando los datos almacenados en generadores deberás procesar su información mediante las siguientes consultas.

4.3.1. Consultas de 1 generador

Modificar
```
def naves_de_material(generador_naves: Generator,
    material: str) -> Generator:
```

Recibe un **generador** con instancias de `Naves`, y un `str` con un **material** de nave en particular.

Retorna un **generador** con todas las naves cuyo `material` coincide con el **material** de nave recibido.

Modificar
```
def misiones_desde_fecha(generador_misiones: Generator, fecha: str,
    inverso: bool) -> Generator:
```

Recibe un **generador** con instancias de `Mision`, un `str` con una **fecha** en particular y un `bool` llamado **inverso**.

Retorna un **generador** de misiones que dependen de **inverso**. Si **inverso** es `False`, se retornan las misiones desde la **fecha**, contandola, en adelante. En caso de que **inverso** sea `True`, se retornan las misiones desde la **fecha**, contandola, hacia atrás.

Modificar
```
def naves_por_intervalo_carga(generador_naves: Generator,
    cargas: tuple[float, float]) -> Generator:
```

Recibe un **generador** con instancias de `Naves` y una `tuple` que contiene los valores de carga mínimo y máximo, respectivamente.

Retorna un **generador** todas las `Naves` cuya carga se encuentre en estos valores de carga, ambos inclusive.

Modificar
```
def planetas_con_cantidad_de_minerales(generador_planeta_mineral: Generator,
    id_mineral: int, cantidad_minima: int) -> List[int]:
```

Recibe un **generador** con instancias de `PlanetaMineral`, el id del mineral y un entero con la cantidad mínima de mineral a extraer.

Retorna una **lista** con los id de `Planetas` que cumplen la cantidad neta mínima del mineral especificado.

Para calcular la cantidad neta de mineral extraíble de un planeta, debes multiplicar la cantidad disponible por la pureza de dicho planeta.

Modificar
```
def naves_astronautas_rango(generador_tripulacion: Generator,
    rango: int, minimo_astronautas: int) -> Generator:
```

Recibe un **generador** con instancias de `Tripulacion`, un `int` con un rango de astronauta y un entero con el mínimo de astronautas requeridos.

Retorna un **generador** de tuplas compuestas por dos elemento. El primer elemento corresponde a la patente de la nave, mientras que el segundo es un generador que contiene los `ids` de los astronautas en dicha nave que poseen un rango rango igual o mayor al especificado.

Todas las tuplas retornadas deben poseer al menos mínimo de astronautas indicados. Si la cantidad indicada es menor a 1, entonces**todas** las naves cumplen la condición de rango.

Modificar
```
def cambiar_rango_astronauta(generador_tripulacion: Generator,
    id_astronauta: int, rango: int) -> Generator:
```

Recibe un **generador** con instancias de `Tripulacion`, un entero con el id de un `Astronauta`, y un `int` con el nuevo rango del astronauta.

Retorna el *generador* de tripulación modificado, donde el astronauta con `id_astronauta` presente el rango recibido.

Hint: Para hacer el cambio correspondiente en el generador, investigue el método `namedtuple._replace()`.

Nota: Puede asumir que el astronauta pertenece a una única tripulación.

Modificar
```
def encontrar_planetas_cercanos(generador_planetas: Generator, x1: int,
    y1: int, x2: int, y2: int, cantidad: int | None=None) -> Generator:
```

Recibe un **generador** de `Planetas`, dos pares de **coordenadas** ($x1, y1$ y $x2, y2$) que representan un sector dentro del espacio, y un **parámetro opcional** (`cantidad`) que indica la cantidad máxima de planetas que debe retornar la consulta.

Retorna un **generador** con todos los planetas cuyas coordenadas se encuentren entre $(x1, y1)$ y $(x2, y2)$.

Cuando el parámetro opcional es distinto de `None`, se retornan los primeros `cantidad` de `Planetas` (los con menor `id_planeta`) que cumplan con las coordenadas.

Nota: Puede asumir que $x1 \leq x2$ y que $y1 \leq y2$.

4.3.2. Consultas de 2 generadores

Modificar
```
def disponibilidad_por_planeta(generador_planeta_minerales: Generator,
    generador_planetas: Generator, id_mineral: int) -> Generator:
```

Recibe dos **generadores**: uno con con instancias de `PlanetaMineral`, y otro con instancias de `Planetas`, además de un `int` llamado `id_mineral`.

Retorna un **generador** de tuplas compuesta por tres elementos, donde cada tupla tiene la forma:

(`nombre_planeta`, `id_planeta`, `cantidad_de_mineral_disponible`)

donde:

- `nombre_planeta` es un `str` con el `nombre` del planeta.
- `id_planeta` es un `int` con el `id_planeta` del planeta.
- `cantidad_de_mineral_disponible` es un `float` con la `cantidad_disponible` de mineral que coincide con el mineral entregado por `id_mineral`. Si no hay mineral en el planeta la cantidad es 0.0.

Modificar
```
def misiones_por_tipo_planeta(generador_misiones: Generator,
    generador_planetas: Generator, tipo: str) -> Generator:
```

Recibe dos **generadores**: uno con con instancias de `Mision`, y otro con instancias de `Planetas`, además de un `str` **tipo** que indica el tipo de planeta.

Retorna un **generador** de instancias de `Mision` que hayan sido realizadas y cuyo `planeta` asociado posee un `tipo` que coincide con el **tipo** entregado. Por ejemplo, si el **tipo** entregado fuera `Rocoso`, el generador retorna todas las misiones que fueron realizadas en planetas de tipo `Rocoso`.

Modificar
```
def naves_pueden_llevar(generador_naves: Generator,
    generador_planeta_mineral: Generator, id_planeta: int
    ) -> Generator[tuple[str, int, float]]:
```

Recibe un **generador** con instancias de `Naves`, otro con instancias de `PlanetaMineral` y un entero con el id de un planeta.

Retorna un **generador** de tuplas de tres elementos:

(`patente_nave`, `id_mineral`, `porcentaje`)

donde:

- `patente_nave` es un `str` con la patente de la nave.
- `id_mineral` es un `int` con el id del mineral que la nave puede llevar, considerando su capacidad en ese momento.
- `porcentaje` es un `float` con el porcentaje de dicho mineral que se puede llevar la nave del planeta. Al momento de realizar el calculo debes considerar la cantidad impura de mineral y la cantidad mineral en el planeta.

El generador posee una tupla por cada combinación de patente nave y mineral del planeta. No se consideran los minerales que no estén en `PlanetaMineral`.

4.3.3. Consultas de 3 generadores

Modificar
```
def planetas_por_estadisticas(generador_mineral: Generator,
    generador_planeta_mineral: Generator, generador_planeta: Generator,
    moles_elemento_min: int, concentracion_molar_min: int,
    densidad_min: int) -> Generator
```

Recibe tres **generadores**: uno con con instancias de `Mineral`, otro con instancias de `PlanetaMineral`, y un último con instancias de `Planetas`. Además, puede recibir los **valores opcionales**: moles mínimos del mineral (en mol), concentración molar mínima $\left(\text{en } \frac{mol}{km^3}\right)$ y densidad mínima $\left(\text{en } \frac{g}{km^3}\right)$.

Retorna un **generador** de instancias de `Planetas` cuyas estadísticas cumplan con los requisitos mínimos entregados para al menos un `Mineral`.

Para fines de esta función, estos son los cálculos:

$$moles\_elemento\,(mol) = \frac{cantidad\_disponible\,(ton)}{masa\_atomica\,\left(\frac{g}{mol}\right)}$$

$$concentracin\_molar\left(\frac{mol}{km^3}\right) = \frac{cantidad\_disponible\,(ton)}{volumen\_planeta\,(km^3) \cdot masa\_atomica\,\left(\frac{g}{mol}\right)}$$

$$densidad\left(\frac{g}{km^3}\right) = \frac{cantidad\_disponible\,(ton)}{volumen\_planeta\,(km^3)}$$

Recuerda que la **cantidad** está dada en toneladas (t) y la masa atómica en $\frac{g}{mol}$.

Para completar esta función, se les otorga el módulo **`utilidades.pyc`**, el cual contiene la función `radio_planeta` la cual recibe un `id_planeta` y el `str` `tamano_planeta` y retorna su radio, en **km**, por lo que deben usar la siguiente fórmula para calcular su volumen:

$$volumen = \frac{4}{3} \cdot \pi \cdot r^3$$

Deben usar **`math.pi`** para realizar los cálculos que usen π.

Modificar
```
def ganancias_potenciales_por_planeta(generador_minerales: Generator,
    generador_planeta_mineral: Generator, generador_planetas: Generator,
    precios: dict) -> dict:
```

Recibe tres **generadores**: uno con instancias de `Mineral`, otro con instancias de `PlanetaMineral` y un último con instancias de `Planetas`. Además, se entrega un diccionario **precios** de la forma:

{**nombre_mineral**: **valor**}

donde **mineral** es un `str` con el nombre de algún `mineral` en específico, y **valor** es un `float` con el valor de este por unidad (en toneladas).

Retorna un diccionario de todos los planetas de la forma:

{**planeta**: **valor_potencial**}

donde **planeta** es un `int` con el `id_planeta` de algún planeta y **valor_potencial** es un `float` con el valor potencial de ese planeta en especifico.

El valor potencial de un planeta se consigue al sumar el valor total de cada mineral que haya en ese planeta. El valor total de un mineral determinado, corresponde al precio del mineral multiplicado por la cantidad pura del mismo que hay en el planeta.

Puedes asumir que si dos minerales comparten el mismo nombre, entonces tienen el mismo precio.

Modificar
```
def planetas_visitados_por_nave(generador_planetas: Generator,
    generador_misiones: Generator, generador_tripulaciones: Generator)
     -> Generator:
```

Recibe tres **generadores**: uno con instancias de `Planetas`, otro con instancias de `Mision` y un último con instancias de `Tripulacion`.

Retorna un **generador** de tuplas de la forma:

(`patente`, `nombre_planeta`, `id_planeta`)

donde:

- `patente` es la patente de la nave que se está consultando.
- `nombre_planeta` es el nombre del planeta que visitó la nave.
- `id_planeta` es el id del planeta visitado.

El generador no puede tener tuplas repetidas. Si la nave a consultar no ha visitado ningún planeta, se retorna una tupla de la forma (patente, `None`, `None`)

Modificar
```
def mineral_por_nave(generador_tripulaciones: Generator,
    generador_misiones: Generator, generador_misiones_mineral: Generator)
     -> Generator:
```

Recibe tres **generadores**: uno con instancias de `Tripulacion`, otro con instancias de `Mision` y un último con instancias de `MisionMineral`.

Retorna un **generador** de tuplas de la forma:

(`patente_nave`, `cantidad_de_mineral_recolectada`)

donde:

- `patente_nave` es un `str` con la patente de la nave utilizada por alguna tripulación.
- `cantidad_de_mineral_recolectada` es un `float` que corresponde a la suma de todo el mineral obtenido en sus misiones. Para que alguna cantidad de mineral sea considerado como recolectado, la misión debe haber sido lograda.

Se retorna únicamente una tupla por nave dentro del generador.

Modificar
```
def porcentaje_extraccion(generador_tripulacion: Generator,
    generador_mision_mineral: Generator,
    generador_planeta_mineral: Generator,
    Mision : namedtuple) -> tuple[float, float]:
```

Recibe tres **generadores**: uno con instancias de `Tripulacion`, otro con instancias de `MisionMineral` y un último con instancias de `PlanetaMineral`. Además, recibe una instancia de `Mision`.

Retorna una tupla de la forma:

(`porcentaje_extraído_del_planeta`, `porcentaje_por_tripulante`).

Donde `porcentaje_extraído_del_planeta` corresponde a cuánto mineral extrajo la misión con respecto al total del planeta. `Porcentaje_por_tripulante` corresponde a `porcentaje_extraído_del_planeta` dividido en el número de tripulantes de la misión.

Puedes asumir que **todos** los tripulantes participaron en igual medida en la recolección, y que la capacidad de la nave es suficiente para llevarse todos los minerales de la misión.

Debes considerar que la misión solo usa como cantidad el mineral puro, mientras que el planeta usa el mineral impuro, por lo que deberás calcular el porcentaje usando valores puros.

Si la misión no está marcada como lograda, se considera como que no se recolectaron recursos.

4.3.4. Consulta de 4 generadores

Modificar
```
def resultado_mision(mision: Mision, generador_naves: Generator,
    generador_mision_mineral: Generator,
    generador_planeta_mineral: Generator,
    generador_tripulacion: Generator) -> Mision
```

Recibe una instancia de `Mision`, un generador con instancias de `Naves`, un generador con instancias de `MisionMineral`, un generador con instancias de `PlanetaMineral`, y un generador con instancias de `Tripulacion`.

Retorna una instancia de `Mision` con el campo `lograda` en `True` si ocurre alguno de los siguientes casos:

- Si los minerales están disponibles en el planeta, y alcanza la capacidad de la nave para dicho mineral.
- Si la cantidad de "gatochico de Platino" (`Mineral` de `id` 1) requerida alcanza para almacenarla en la nave sin considerar los otros minerales de la misión.

En otro caso, se usa `False` para el campo `lograda`.

Nota 1: Debes corroborar que `PlanetaMineral` tenga suficiente mineral, y además este debe caber en la nave.
Nota 2: El *Mineral 1* (mineral cuyo `id_mineral` es 1) se ha bautizado coloquialmente por los astronautas como *"gatochico de Platino"*, por lo que debes buscar las ocurrencias de este para el segundo caso.

5. Interfaz Gráfica e Interacción

Con el fin de administrar correctamente toda la información sin necesidad de conocimientos previos, se ha pedido diseñar una interfaz gráfica que facilite la interacción y muestre resultados deseados.

5.1. Modelación del programa

En esta sección se evaluarán, entre otros, los siguientes aspectos:

- Correcta **modularización** del programa, esto quiere decir que se debe respetar y seguir una adecuada estructuración entre ***frontend*** y ***backend***, con un **diseño cohesivo** y de **bajo acoplamiento.**
- Correcto uso de señales entre ***backend*** y ***frontend***.
- Un flujo prolijo a lo largo del programa. Esto quiere decir que el usuario puede navegar sin problemas entre las distintas partes que componen *Departamento de las Colecciones del Cosmos* solo ejecutando una vez el programa. En otras palabras, un usuario nunca debe quedarse atorado en alguna parte del programa que lo obliga a cerrar *Departamento de las Colecciones del Cosmos* y volver a ejecutarlo. A modo de ejemplo: si un usuario ingresa a una sección y no se puede mover a otra sección, esto es considerado como una mala implementación.
- Una interfaz interactiva integrada con las funcionalidades pedidas. Esto quiere decir que se espera que la comunicación y el comportamiento del programa sea visible y controlable desde la interfaz. **No se asignará puntaje** si las funcionalidades solicitadas no son visibles en la interfaz ó si es que estas solo se pueden comprobar en la consola o en el código, a menos que se indique explícitamente lo contrario.
- Generación de las ventanas mediante código programado por el estudiante. Es decir, **no se permite** la creación de ventanas con el apoyo de herramientas como `QtDesigner`, `QML`, entre otros.

5.2. Ventanas

Para la correcta implementación de *Departamento de las Colecciones del Cosmos*, se espera la creación de **tres ventanas**, cada una con elementos mínimos de interfaz que deben estar presentes, los cuales serán detallados a continuación.

Importante:

Los esquemas y diseños que serán expuestos son únicamente referenciales. No es necesario que tu tarea sea una copia exacta de estos; por lo que **lo único que será evaluado es que los elementos mínimos estén presentes y funcionen del modo que se exige**.

Los elementos mínimos de cada ventana serán explicitados en la sección correspondiente; pero en ningún caso se les exige cosas relacionadas a la estética de la interfaz gráfica. Por lo tanto, una ventana que sólo tenga los elementos expuestos en los esquemas, tiene el mismo nivel de validez que otra llena de decoraciones y efectos interactivos, esto, suponiendo que ambas tengan el comportamiento deseado.

En caso de que lo consideres necesario puedes ubicar los elementos en otras posiciones; agregar creatividad inventando botones nuevos; nuevas interacciones, diseños y hasta animaciones. Cabe destacar que si se desea desarrollar funcionalidades extras, estas serán evaluadas bajo los mismos criterios generales mencionados anteriormente. Esto quiere decir que, si el programa falla debido a una funcionalidad extra, se aplicara el descuento en el *item* correspondiente.

5.2.1. Ventana de Entrada

Esta ventana existe antes de llegar a la ventana principal, y se muestra al iniciar el programa. Se muestra un mensaje de bienvenida, con un botón para pasar a la ventana siguiente.

En la Figura 2a se muestra un ejemplo de cómo se puede estructurar esta ventana.

5.2.2. Ventana Principal

Esta ventana sirve para elegir de manera interactiva las consultas del ***backend*** que serán ejecutadas y, mostrar los resultados de dichas consultas.

Dado que la gran mayoría de las consultas retornan un iterable, el formato de presentación de los resultados consistirá en mostrar un texto, donde cada línea corresponda a un elemento del iterable.

Para esto, se debe crear una ventana que contenga como mínimo los siguientes elementos (los nombres son referenciales, puedes elegir el que quieras):

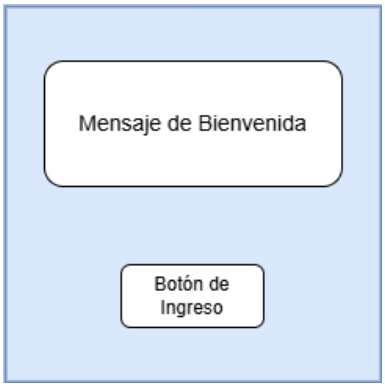
- ***QFileDialog*** que entrega el archivo desde el cual se cargarán los datos.
- ***Input* de texto** donde se pueda ingresar la entidad a cargar y un filtro. Por ejemplo `"Astronauta"` mostrará todos los astronautas. `"Mineral,id_mineral=1"` mostrará todos los minerales con id_mineral = 1. `"Nave,material=Aluminio,capacidad_astronautas=6"` mostrará todas las naves de aluminio con capacidad_astronauta igual a 6. En caso de que no coincida la entidad ingresada con el archivo entregado en el ***QFileDialog***, se debe mostrar un mensaje de error.
- **Botón "Ejecutar Consulta"** que se encarga de cargar los datos indicados en el *input* de texto que recibe el *path* del archivo (obtenido con ***QFileDialog***) y cargar la entidad indicada con el filtro correspondiente.
- **Área de texto** que posea un *scroll* vertical para ver todos los resultados de la consulta.
- **Botón "Mapa"** que visualiza los planetas dentro de un espacio determinado. El objetivo es utilizar la función `encontrar_planetas_cercanos` en este ítem.

En la Figura 2b se muestra un ejemplo de cómo puede verse la ventana principal, con los elementos previamente mencionados.

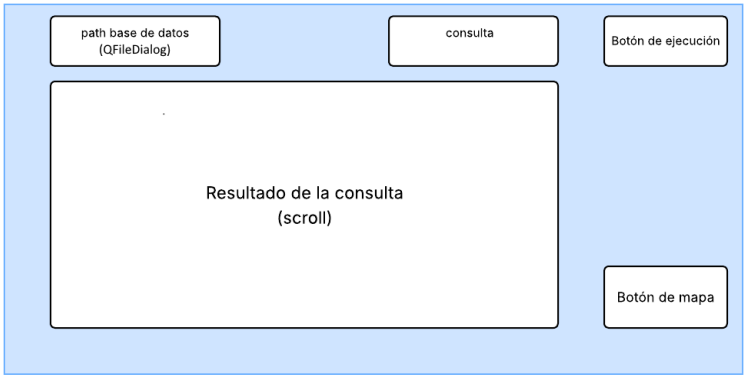
5.2.3. Ventana Mapa

Con esta ventana, se podrá visualizar el mapa formado por los planetas, aprovechando las coordenadas de los planetas. Para esto, es necesario que llames a la función `encontrar_planetas_cercanos` para obtener los planetas que se encuentren dentro de un sector.

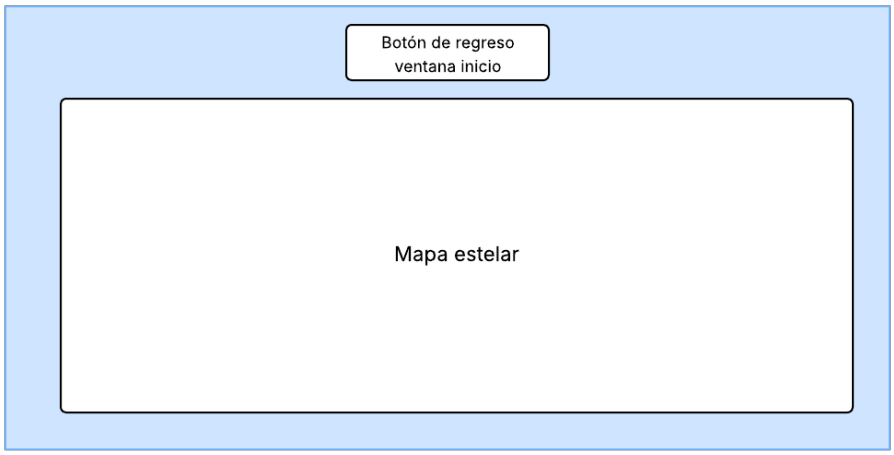
Es necesario que crees el archivo **parametros.py** que asigne los valores $x1$, $y1$, $x2$, $y2$ y la **cantidad** de planetas a mostrar (recuerda dejarlo como `None` si es que se quiere visualizar todos los planetas). Además, para poder representar con fidelidad los tamaños de los planetas, debes considerar sus radios y multiplicarlo por un **factor**, almacenado también en **parametros.py**. Las dimensiones de cada eje, los cuales representan el porte del contenedor del mapa, también deben ser agregados a **parametros.py**. Es importante considerar que los valores de las variables de **parametros.py** pueden ser cambiados en el momento de la corrección.



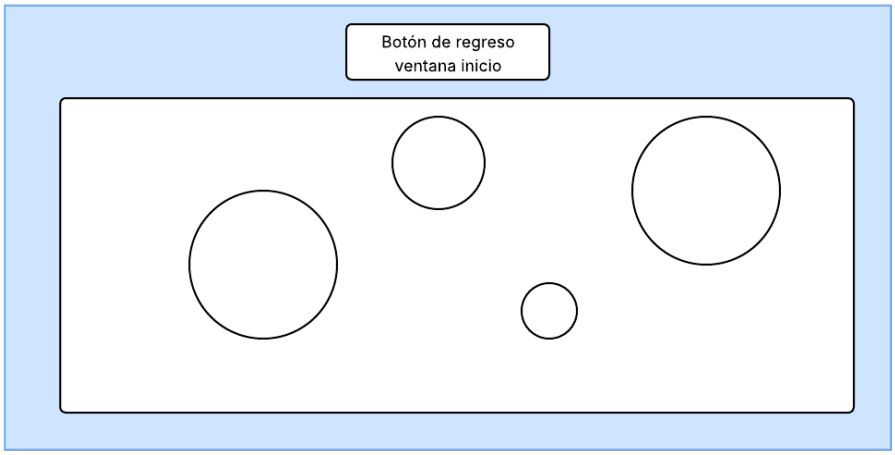
(a) Ventana de entrada



(b) Ventana principal



(c) Ventana mapa



(d) Ejemplo mapa

Figura 2: *Mock-up* de las ventanas

6. *Tests*

El objetivo principal de la implementación de los *tests* es la eficiencia. Cada test posee un tiempo límite de ejecución permitido. Estos tiempos pueden ser distintos para distintos *tests*. **Si no se cumple con el tiempo límite, no se otorgará puntaje.**

Los *tests* presentes en esta evaluación se dividirán en dos conjuntos:

- **Correctitud:** Evaluará el comportamiento de la solución implementada con respecto a posibles casos bordes. En estos casos, se probará la función con generadores ya cargados por el programa.
- **Carga de datos:** Evaluará el comportamiento de la solución implementada con respecto a la eficiencia del código y su capacidad de trabajar con archivos de diversos tamaños.

Para lograr este objetivo de esta evaluación, es esencial un correcto uso de **generadores** y **programación funcional**. El no aplicar correctamente los contenidos anteriormente mencionados, puede provocar que las funciones no terminen en el tiempo esperado.

Reiterando lo indicado en el Flujo del programa: Es sumamente importante que **no realices la tarea basándote en los tests públicos**. Te debes basar en el enunciado y sólo puedes usar los tests públicos para apoyo. Debido a esto, es importante destacar que **si el alumno tiene los test públicos correctos, pero los tests privados incorrectos, no se otorgará puntaje**.

Adicionalmente, para los *tests* de Carga de datos de la sección Carga de datos de `namedtuples` se evalúa el uso de memoria máxima de la función para comprobar el uso de programación funcional. Para tener puntaje en esta se requiere que el test haya pasado su equivalente de los *tests* de Correctitud de la misma sección.

6.1. Ejecución de *tests*

Para la corrección automática se entregarán varios archivos `.py` los cuales contienen diferentes *tests* que ayudan a validar el desarrollo de la tarea. Para ejecutar estos *tests*, primero debes posicionar tu terminal/consola en la carpeta de la tarea `Tareas/T3/`. Luego, desde esta misma, debes escribir el siguiente comando para ejecutar todos los *tests*:

- `python3 -B -m unittest discover tests_publicos -v -b`

En cambio, si deseas ejecutar un subconjunto de *tests*, puedes hacerlo escribiendo lo siguiente:

- `python3 -B -m unittest -v -b tests_publicos.<test_N>`
  Reemplazando `<test_N>` por el nombre del archivo de test que desees probar.

  Por ejemplo, si quisieras probar si realizaste correctamente la función "`misiones_por_fecha`", deberás escribir lo siguiente:

  `python3 -B -m unittest -v -b tests_publicos.test_01_misiones_desde_fecha_correctitud`

Importante: Recuerda que para ejecutar Python debes usar el comando específico de tu computador, este puede ser: `py`, `python`, `py3`, `python3` o `python3.12`.

7. *utilidades.pyc*

Se entregara un archivo **.pyc**, el cual contendrá las `namedtuples` y funciones que deben utilizar para el desarrollo de la tarea. Estas `namedtuples` y funciones solo deben ser importadas y utilizadas, sin la posibilidad de ser modificadas o ver su funcionamiento.

7.1. `Namedtuples`

Tal como se indica en la Subsección 4.1: Datos, cada uno de los datos entregados en los archivos CSV presenta una `namedtuples` que lo representa. El detalle de estas `namedtuples` se encuentra en la Subsección 4.1.

7.2. Funciones Auxiliares

Adicionalmente, para que puedas implementar algunas consultas se te entregan la siguiente función auxiliar:

- No modificar `def radio_planeta(id_planeta: int, tamano_planeta: int) -> float:`

  Recibe el id de un planeta (`int`) y el tamaño de un planeta (`str`), y retorna un `float` que es el radio del planeta en *km*.

8. `.gitignore`

Para esta tarea **deberás utilizar un** `.gitignore` para ignorar los archivos indicados, este deberá estar dentro de tu carpeta `Tareas/T3/`.

Los elementos que no debes subir y **debes ignorar mediante el archivo** `.gitignore` para esta tarea son:

- El enunciado.
- La carpeta `data/` y los archivos `csv` y `zip` correspondientes a los datos.
- La carpeta `test_publicos/` y los archivos zip relacionados a dicha carpeta.
- Cualquier archivo `utilidades.pyc` o algún otro que posea dicha extensión.

Recuerda **no ignorar archivos vitales de tu tarea como los que tú creas o modificas, o tu tarea no podrá ser revisada**.

El correcto uso del archivo `.gitignore`, implica que los archivos **deben** no subirse al repositorio debido al uso archivo `.gitignore` y no debido a otros medios.

Importante **Debes asegurarte de que ni los archivos de datos ni los archivos de los *tests* no sean subidos a tu repositorio personal, en caso contrario, se aplicarán 10 décimas de descuento de formato en la corrección de la evaluación.** Dado que en esta evaluación presenta archivos de gran tamaño, junto a los archivos base de la tarea se incluye un archivo `.gitignore` que ignora todos los archivos `csv` y `zip`.

Finalmente, en caso hacer *commit* de la carpeta "`data/`" o "`tests_publicos/`", debido al tamaño de dichos archivos, Git impedirá que dicho *commit* y los siguientes puedan ser subidos al repositorio remoto, por lo que no podrán hacer entregas parciales. En caso de que lleguen a enfrentarse a este problema, deben:

1. Hacer un respaldo de su solución.

2. Volver a clonar su repositorio personal.
3. Agregar al nuevo repositorio los cambios respaldados.
4. Hacer *commit* y *push* de los cambios, teniendo consideración de no agregar los archivos de datos.

9. Importante: Corrección de la tarea

En el **siguiente enlace** se encuentra la distribución de puntajes. Para esta tarea, el carácter funcional del programa será el pilar de la corrección, es decir, **sólo se corrigen tareas que puedan ejecutar**. Por lo tanto, se recomienda hacer periódicamente pruebas de ejecución de su tarea y *push* en sus repositorios, corroborando que cada *test* que les pasamos para cada consulta corra en el tiempo indicado (Sección 6: *Tests*), en caso contrario se asumirá un resultado incorrecto.

Importante: Todo ítem corregido automáticamente será evaluado de forma ternaria: puntaje completo si pasa todos los *tests* de dicho ítem, medio punto para quienes pasan más del 65 % de los *test* de dicho ítem, y 0 puntos para quienes no superan el 65 % de los *tests* en dicho ítem. Finalmente, todos los descuentos serán asignados automáticamente por el cuerpo docente.

La corrección se realizará en función del último *commit* realizado antes de la fecha oficial de entrega (lunes 27 de octubre a las 20:00). Si se desea continuar con la evaluación en el periodo de entrega atrasado, es decir, realizar un nuevo *commit* después de la fecha de entrega, **es imperante responder el formulario de entrega atrasada** sin importar si se utilizará o no cupones. Responder este formulario es el mecanismo que el curso dispone para identificar las entregas atrasadas. El enlace al formulario está en la primera hoja de este enunciado y estará disponible para responder hasta el miércoles 29 de octubre las 23:59.

Para terminar, si durante la realización de tu tarea se te presenta algún problema o situación que pueda afectar tu rendimiento, no dudes en contactar al ayudante de Bienestar de tu sección. El correo está en el **siguiente enlace**.

10. Restricciones y alcances

- Esta tarea es **estrictamente individual**, y está regida por el **Código de honor de Ingeniería**.
- Tu programa debe ser desarrollado en Python 3.12.X con X mayor o igual a 0.
- Tu programa debe estar compuesto por uno o más archivos de extensión `.py` que estén correctamente ordenados por carpeta. **No se revisará archivos en otra extensión como `.ipynb`**.
- Todo el código entregado debe estar contenido en la carpeta y rama (*branch*) indicadas al inicio del enunciado. Ante cualquier problema relacionado a esto, es decir, una carpeta distinta a `Tareas/T3/` o una rama distinta a `main`, se recomienda preguntar en las ***issues* del foro**.
- Si no se encuentra especificado en el enunciado, supón que el uso de cualquier librería Python está prohibida. Pregunta en la *issue* especial del **foro** si es que es posible utilizar alguna librería en particular.
- Debes adjuntar un **único archivo markdown**, llamado `README.md`, **conciso y claro**, donde describas las referencias a código externo. El no incluir este archivo, incluir un readme vacío o el subir más de un archivo `.md`, conllevará un **descuento** en tu nota.
- Esta tarea se debe desarrollar **exclusivamente** con los contenidos liberados al momento de publicar el enunciado. No se permitirá utilizar contenidos que se vean posterior a la publicación de esta evaluación.
- Se encuentra estrictamente prohibido citar código que haya sido publicado **después de la liberación del enunciado**. En otras palabras, solo se permite citar contenido que ya exista previo a la publicación del enunciado. Además, se encuentra estrictamente prohibido el uso de herramientas generadoras de código para el apoyo de la evaluación.
- Cualquier aspecto no especificado queda a tu criterio, siempre que no pase por sobre otro que sí sea especificado por enunciado.

Las tareas que no cumplan con las restricciones del enunciado obtendrán la calificación mínima (1,0)**.**