Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación
IIC2233 — Programación Avanzada 2021-1

15 de Abril de 2021
Actividad Formativa

Actividad Formativa 2

Levantamiento y manejo de excepciones

Entrega

- **Lugar:** En su repositorio privado de GitHub, en la **carpeta** `Actividades/AF2/`
- **Hora del *push*:** 16:30

Importante: Antes de comenzar, comprueba que Git este funcionando correctamente en tu repositorio privado. Para esto, **sube los archivos base de la actividad de inmediato** (*add*, *commit*, *push*). Se espera que en esta actividad (así como en las demás actividades y tareas) utilices Git a lo largo de **todo tu desarrollo** como una herramienta, no sólo como un método de entrega. Es por esto que recomendamos enfáticamente que vayas subiendo tus cambios constantemente (***push***), ya que **problemas de último minuto** relacionados con la entrega y Git **no serán considerados**.

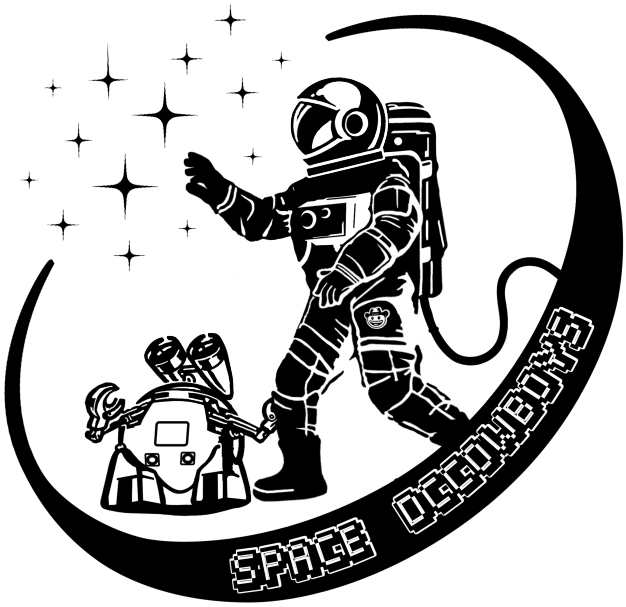
Introducción

Estás tranquilo programando la AF1 en la estación espacial de Marte junto a tu amigo robot **PER1GEE** y te das cuenta que te llega un mensaje de DCConecta2 desde la Tierra. Para tu sorpresa es de prioridad nivel 1 enviado por la NASA el cual dice lo siguiente:

Estimade,

Te escribo porque un objeto volador no identificado colisionó uno de nuestros satélites desviándolo de su ruta hacia la galaxia Andrómeda. Con el impacto, éste envió información errónea alterando todos los datos de estrellas y cuerpos celestes que recopiló **PER1GEE** *en su estancia en Marte. Me comuniqué con personas expertas en el área y me dijeron que con tus conocimientos de programación podremos recuperar los datos y calcular una ruta segura para ir en busca del satélite a tiempo. Quedas al mando de esta nueva misión* **Space DCCowboys.**

Buena suerte y buen viaje vaquero espacial.



Flujo del programa

En esta actividad tendrás que poner a prueba todos tus conocimientos de excepciones para lograr completar tu misión interestelar. Luego de leer el mensaje de DCConecta2 inmediatamente conectas **PER1GEE** a tu computador para limpiar datos recopilados. En esta primera parte deberás verificar y corregir los datos de las estrellas que se encuentran en tu camino a recuperar el satélite, además de confirmar la existencia de estrellas cercanas. Luego, te das cuenta que no puedes aproximarte demasiado a estrellas muy luminosas porque no te dejarán ver, ni a estrellas muy grandes porque puedes chocar y tampoco a estrellas de elevada temperatura porque corres el riesgo de quedar carbonizado. Por esto en la segunda parte de esta actividad debes crear una excepción personalizada que te permita armar una ruta que te lleve sano y salvo a tu destino.

Suerte, ¡que toda la fuerza esté contigo!

Archivos

- `estrellas.csv`: Este archivo contiene los datos de todas las estrellas registradas. ¡Este archivo fue dañado! Por lo que pueden haber estrellas con datos erróneos. **No debes modificar este archivo.** El formato del archivo es:

  `Nombre,Alias,Magnitud,Distancia,Temperatura,Radio,Luminosidad`

  donde `Alias` es un código para identificar a la estrella. Un ejemplo de línea puede ser:

  `Enif,86ZM,2.38,670,4460,150,6700`

- `estrellas_cercanas.txt`: Este archivo contiene los nombres de todas las estrellas que están en ruta. ¡Este archivo fue dañado! Por lo que puede ser que algunos nombres no se encuentren en el archivo anterior. **No debes modificar este archivo.**

- `cargar_datos.py`: Este archivo contiene dos funciones para cargar los datos necesarios para la actividad (de `estrellas.csv` y `estrellas_cercanas.txt`). **No debes modificar este archivo.**

- `estrellas.py`: Este archivo contiene a la clase **Estrella** que se explica a continuación. **No debes modificar este archivo.**
- `verificar_estrellas.py`: Este archivo contiene las funciones para verificar y corregir los datos de las estrellas, también para identificar las estrellas cercanas. **Deberás trabajar en este archivo** durante la *Parte 1: Levantar y capturar excepciones*.
- `excepciones_estrellas.py`: Este archivo contiene la clase `RutaPeligrosa`, que se explicará en la *Parte 2: Excepción personalizada*. **Deberás trabajar con este archivo**.
- `calcular_ruta.py`: Este archivo genera la ruta por las estrellas, verificando que cumplan las condiciones para un viaje seguro. Se debe utilizar la excepción `RutaPeligrosa` para calcular una ruta segura para el astronauta. **Deberás trabajar con este archivo**.
- `main.py`: Este es el archivo principal a ejecutar. El objetivo de este código es cargar los datos y corregir los errores de las estrellas para luego agregar las estrellas verificadas a tu ruta. **No debes modificar este archivo.**

Clase Estrella

Tendrás que trabajar con objetos de la clase `Estrella`, que se encuentra definida en `estrellas.py`. Esta clase **ya está implementada** y tiene los siguientes atributos, algunos de los cuales deberás revisar y posiblemente corregir mediante excepciones en la parte I.

- `nombre` : Un `str` que corresponde al nombre de la estrella.
- `alias` : Un `str` que corresponde al nombre de la estrella en letras y números. **Debe ser revisado**.
- `magnitud` : Un `str` que corresponde a la magnitud de la estrella. **Debe ser revisado**.
- `distancia` : un `float` que corresponde la distancia desde la Tierra en Años Luz. **Debe ser revisado**.
- `temperatura` : Un `int` que corresponde a la temperatura de la estrella en Kelvin.
- `radio` : Un `int` que corresponde al radio de la estrella en Kilómetros.
- `luminosidad` : Un `int` que corresponde a la luminosidad de la estrella en Lux.

Parte I: Levantar y Capturar excepciones

En esta parte tendrás la misión de manejar excepciones en algunas funciones para realizar con éxito la actividad **Space DCCowboys**. Deberás completar las siguientes funciones, las cuales se encuentran en el archivo `verificar_estrellas.py` y utilizar `try/except` o levantar excepciones con `raise` donde corresponda.

Importante: Para esta parte no se considerará correcto el uso de `except` sin argumentos o `except Exception`. Para comprobar el código puedes descomentar las líneas que están en la sección principal de `verificar_estrellas.py` y ejecutar este archivo de manera individual.

- `def verificar_alias_estrella(estrella: Estrella)`: Recibe una instancia de la clase `Estrella` y deberás usarla para verificar que el alias de la estrella está en el formato correcto, el cual consiste en dos letras mayúsculas y dos números. Los pares siempre estarán juntos, entonces el alias puede venir como `AB12` o `12AB` pero nunca como `1A2B`. Siempre será de cuatro caracteres exactos, por lo que no necesitas verificar eso. En particular, debes asegurarte que se cumplan los siguientes requisitos:

- Las letras deben ir antes de los números.
- Las letras no deben contener la letra `'F'`.

Si no se cumple al menos uno de estos (pueden ocurrir ambos a la misma vez) debes **levantar** un `ValueError`, y el mensaje del error debe ser algo así:

```
"Error: El alias de la estrella es incorrecto."
```

- `def corregir_alias_estrella(estrella: Estrella):` Recibe un objeto de clase `Estrella`. Usando la función anterior, debes verificar que el alias de la estrella tenga el formato correcto. Si **capturas** un error debes imprimirlo y corregir el problema. En el caso de que el error fue ocasionado porque tenía los números antes de las letras deberás ordenar el `str`. Si por el contrario, surgió el error porque había una `'F'`, deberás reemplazarla por una `'T'`. En caso de haber corregido el valor, debes imprimir un mensaje avisándole al usuario que el error ha sido corregido.

```
f"El alias de {estrella.nombre} fue correctamente corregido.\n"
```

- `def verificar_distancia_estrella(estrella: Estrella):` Recibe un objeto de la clase `Estrella` y revisa que su atributo `distancia` sea positivo. Si este es negativo entonces **se levanta** un `ValueError` con un mensaje similar al siguiente:

```
"Error: Distancia negativa."
```

- `def corregir_distancia_estrella(estrella: Estrella):` Recibe una instancia de la clase `Estrella`. Verifica que la distancia tenga el valor correcto usando la función anterior. Si **detecta** un error, lo imprime y luego cambia el signo de la distancia. Debes `capturar el error` y modificar el valor del atributo. En caso de haber corregido el valor debes imprimir un mensaje para avisarle al usuario, como este:

```
f"La distancia de la estrella {estrella.nombre} fue corregida.\n"
```

- `def verificar_magnitud_estrella(estrella: Estrella):` Esta función recibe una instancia de la clase `Estrella`. Debes revisar si el atributo `magnitud` de la estrella está en formato `float`. Si no, se debe **levantar** un `TypeError` con un mensaje parecido a este:

```
"Error: Magnitud no es del tipo correcto."
```

- `def corregir_magnitud_estrella(estrella: Estrella):` Recibe una instancia de la clase `Estrella`. Usando la función anterior, deberás **verificar** que la magnitud de la estrella tenga el formato correcto. En caso de que haya un error, deberás imprimirlo y corregir lo siguiente: revisar si tiene un punto y coma (`';'`), el cual reemplazarás por un punto (`'.'`) y luego deberás **convertir** el `str` a un `float`. Si ya viene con un punto (`'.'`), deberás solamente hacer la conversión de tipos. Tu tienes que implementar esta función. Al terminar de arreglarlo, debes imprimir un mensaje para avisarle al usuario.

```
f"La magnitud de la estrella {estrella.nombre} fue corregida.\n"
```

Después de completar estos métodos **puedes correr el archivo `verificar_estrellas.py` para comprobar que todo está funcionando correctamente.** Deberías poder ver al menos los siguientes mensajes, y otros más.

```
"Revisando posibles errores en las estrellas..."

"Error: El alias de la estrella es incorrecto."
"El alias de Acamar fue correctamente corregido."

"Error: Distancia negativa."
"La distancia de la estrella Acamar fue corregida."

"Error: Magnitud no es del tipo correcto."
"La magnitud de la estrella Acamar fue corregida."

"Error: Magnitud no es del tipo correcto."
"La magnitud de la estrella Acrux fue corregida."

"Error: El alias de la estrella es incorrecto."
"El alias de Aldhafera fue correctamente corregido."

"Error: Magnitud no es del tipo correcto."
"La magnitud de la estrella Aldhafera fue corregida."

# ... ... más mensajes de este estilo
```

Por último, deberás completar la siguiente función:

`def dar_alerta_estrella_cercana(nombre_estrella: str, diccionario_estrellas: dict)`: Recibe un `str` que representa el nombre de una estrella del archivo `estrellas_cercanas.txt` y un `dict` tal que sus llaves son nombres de las estrellas del archivo `estrellas.csv` y sus valores son las instancias de las estrellas. Deberás notificar si la estrella `nombre_estrella` está en el `diccionario_estrellas` **utilizando excepciones**. Debes intentar obtener la instancia haciendo `diccionario_estrellas[nombre_estrella]`, **capturar** un posible `KeyError` en caso de que el diccionario no tenga `nombre_estrella` en sus llaves, e imprimir algo como lo siguiente:

```
f"Estrella {nombre_estrella} NO está en nuestra base de datos."
"¡Alerta, puede ser una trampa de algún extraterrestre!"
```

En el caso de que `nombre_estrella` sí esté en el diccionario, deberás imprimir un mensaje parecido al siguiente:

```
f"Estrella {nombre_estrella} está en nuestra base de datos."
f"Su alias es {alias_estrella}."
```

Después de completar esta función **puedes correr el archivo `verificar_estrellas.py` para comprobar que todo está funcionando correctamente.** Deberías poder ver al menos los siguientes mensajes, y muchos más.

```
"Revisando estrellas inexistentes..."

"Estrella Acamar está en nuestra base de datos. Su alias es WT68."
"Estrella Acrux está en nuestra base de datos. Su alias es QK55."
"Estrella Aldhafera está en nuestra base de datos. Su alias es 01BN."
```

```
"Estrella Baten Kaitos está en nuestra base de datos. Su alias es BS83."
"Estrella Bellatrix está en nuestra base de datos. Su alias es 93WH."
"Estrella Sirio NO está en nuestra base de datos."
"Alerta!, puede ser una trampa de algún extraterrestre!"
# ... ... más mensajes de este estilo
```

Parte II: Excepción Personalizada

En esta parte deberás calcular la ruta de tu viaje interestelar. Para esto debes **completar una excepción personalizada** llamada `RutaPeligrosa`, la cual está en el archivo `excepciones_estrellas.py`, y que será llamado al recorrer la base de datos de estrellas corregidas. **El recorrido de la base de datos viene implementado**. Esta excepción se encargará de que se descarten de tu ruta las estrellas muy luminosas, muy grandes o con mucha temperatura.

Clase RutaPeligrosa

La clase `RutaPeligrosa` representa una excepción personalizada que simula un sistema de detección de peligros y tiene un método `dar_alerta_de_peligro` con el fin de imprimir un mensaje dependiendo del tipo de peligro. Debes asegurarte que `RutaPeligrosa` sea una excepción.

- `def __init__(self, tipo_peligro: str, nombre_estrella: str):` Recibe dos `str`. El primero de ellos es el tipo de peligro y el segundo el nombre de una estrella. Debes asignar estos parámetros como atributos de la clase. Además deberá imprimir el mensaje:

```
print('¡Alto ahí viajero! Hay una amenaza en tu ruta...')
```

- `def dar_alerta_de_peligro(self):` En este método deberás imprimir un mensaje dependiendo del tipo de peligro. Los tipos de peligro junto a sus alertas son:
  - Exceso de luz: (`self.tipo_peligro` será igual a `"luz"`)

  ```
  print("¡Ten cuidado, que con tanta luz no podrás ver :(!")
  ```

  - Exceso de tamaño: (`self.tipo_peligro` será igual a `"tamaño"`)

  ```
  print("¡Ooops! Esa estrella es demasiado grande...")
  ```

  - Exceso de calor: (`self.tipo_peligro` será igual a `"calor"`)

  ```
  print("¡Alerta! ¡Alerta! ¡Peligro inminente de quedar carbonizados!")
  ```

  Y luego de eso tienes que imprimir un mensaje de qué estrella ha quedado fuera de tu ruta de la siguiente forma:

```
print(f"La Estrella {self.nombre_estrella} ha quedado fuera de tu ruta.\n")
```

Por último, en el archivo `calcular_ruta.py` se encuentran las siguientes funciones:

`def verificar_condiciones_estrellas(estrella: Estrella)`: Esta función recibe una instancia de la clase `Estrella` y tiene el objetivo de asegurar que la estrella entregada cumpla ciertos requisitos

(luminosidad, magnitud y temperatura) para que la ruta del satélite sea segura. En caso de que la estrella no cumpla alguno de los requisitos se va a generar una excepción del tipo `RutaPeligrosa`. **Esta función ya viene implementada.**

`def generar_ruta_estrellas(estrellas: list) -> list:` Recibe una lista de estrellas, en la cual cada elemento es una instancia de la clase `Estrella` y retorna una lista en la cual cada elemento es el nombre de una estrella aceptada para la ruta. Esta función se encarga de crear y retornar una ruta segura de estrellas, revisando que cada una cumpla con las condiciones necesarias para formar parte de la ruta (luminosidad, magnitud y temperatura). En esta función deberás iterar por cada estrella en la lista `estrellas` y para cada una hacer un `try/except` para capturar la excepción `RutaPeligrosa`. Deberás utilizar la función `verificar_condiciones_estrella` para asegurarte que cada estrella sea segura. Si no hay error de `RutaPeligrosa` debes agregar la estrella a una lista que va a ser tu ruta e imprimir el siguiente mensaje:

```
print(f'¡La estrella {estrella.nombre} se ha agregado a tu ruta!' + u'\x02' + '\n')
```

En caso de que para una estrella, `verificar_condiciones_estrella` levante la excepción `RutaPeligrosa`, deberás capturarla, imprimir el error y llamar el método `dar_alerta_peligro` de la clase `RutaPeligrosa`. Cabe destacar que las estrellas que estén en ruta peligrosa (las que entran en la excepción) no se deben agregar a la ruta final. Finalmente debes retornar una lista con las estrellas que forman parte de tu ruta.

Notas

- Esta actividad esta diseñada para ser implementada en el orden que se presenta en este enunciado.
- Recuerda que debes hacer `git push` en la rama principal de tu repositorio
- Para ver los mensajes escritos en toda su grandeza, corre el programa desde la línea de comandos (`cmd`, `Terminal`, etc).

Objetivos

- Conocer los tipos de errores más relevantes para el curso.
- Ser capaces de levantar y capturar excepciones.
- Ser capaces de definir excepciones personalizadas.