Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación
IIC2233 — Programación Avanzada 2020-1

16 de abril de 2020
Actividad Formativa

Actividad Formativa 03

Excepciones

Entrega

- **Lugar:** En su repositorio privado de GitHub, en la **carpeta** `Actividades/AF03/`
- **Hora del *push*:** 16:50

Introducción

Debido a la crisis sanitaria que vive el país, se ha decretado cuarentena total en algunas comunas de Chile. En la práctica, esto significa que en dichos sectores se prohíbe el libre tránsito de las personas para disminuir la tasa de contagios. Así por ejemplo, muchos estudiantes de Programación Avanzada quedan confinados en sus casas, teniendo como única diversión en las noches hacer ~~fiestas virtuales con intenso de consumo de alcohol~~ los ejercicios propuestos para las actividades.

Sin embargo, existe la posibilidad de que la gente salga de sus casas por motivos especiales como para ir al supermercado o pasear a sus mascotas, pero primero necesitan sacar un permiso. Es por ello que se ha implementado un sistema llamado **DCComisaria Virtual** donde se pueden emitir los permisos, pero cuenta con muchos errores porque se tuvo muy poco tiempo para implementarlo. Te han pedido que ocupes tus conocimientos sobre **excepciones** para que el sistema funcione correctamente.

DCComisaria Virtual

En esta actividad, ayudarás al encargado del sistema a ver qué solicitudes de un permiso específico fueron aprobadas o rechazadas en un día. Al cargar el programa, el encargado podrá seleccionar qué permiso quiere revisar y el programa deberá aprobar o rechazar cada solicitud dependiendo de los requerimientos del permiso en cuestión.

Como en muchas de tus tareas, el encargado selecciona qué permiso quiere revisar a través de un menú, donde el usuario puede ingresar las siguientes opciones:

- 1: Permisos de Clave Única
- 2: Permisos de Asistencia Médica
- 3: Permisos de Servicios Básicos
- 0: Salir

Actualmente, el programa ejecuta lo pedido si se ingresa cualquiera de las opciones {0, 1, 2, 3}. En la Parte 4, deberás hacer que el menú sea a prueba de errores.

Archivos

Se te entregarán los siguientes archivos para la realización de la actividad:

- `datos.csv`: Contiene los datos de todas las personas **registradas** en la DCComisaría Virtual. Cada línea está en el formato:

  `rut,nombre,clave_unica,domicilio`

- `cuarentena.csv`: Contiene todas las comunas que están en cuarentena a lo largo de Chile. Cada fila contiene únicamente el nombre de una comuna.
- `ruts_clave_unica.csv`: Contiene todos los identificadores RUT de las personas que quieren solicitar la clave única. Cada línea contiene únicamente el RUT de una persona.
- `permiso_hora.csv`: Contiene las horas de todas las personas que están solicitando el permiso de asistencia médica. Cada línea está en el formato:

  `rut,hora`

- `permiso_supermercado.csv`: Contiene los datos de todas las personas que solicitan el permiso de servicios básicos. Cada línea está en el formato:

  `rut,lugar_salida,lugar_llegada`

- `cargar_datos.py`: Contiene las funciones que permitirán cargar los datos de los archivos nombrados anteriormente.
- `permisos.py`: Contiene los métodos que deberás utilizar para levantar las excepciones.
- `main.py`: Es el archivo principal a ejecutar, en este se llaman a todas las funciones a implementar.

Parte I: Reparar código

El primer paso de **DCComisaria Virtual** es cargar los datos necesarios para utilizar la plataforma. Alguien comenzó a implementar este sistema, sin embargo, entró en desesperación porque su código tenía

múltiples errores. En este parte debes tomar este código, encontrar los errores y corregirlos para que los datos pueden ser cargados.

Debes ingresar al archivo `cargar_datos.py` y corregir los errores modificando la línea en donde se encuentran. Te recomendamos ejecutar repetidamente el código de `cargar_datos.py` para identificar los errores. Existe **un** error de cada uno de los siguientes tipos:

- `SyntaxError`
- `IndexError`
- `NameError`
- `ValueError`

Para corregirlos debes cambiar únicamente la línea en donde se encuentra el error. Puedes agregar o quitar letras, comas, dos puntos (`:`), o bien, comentar la línea si es que en el código se indica que esa línea no es necesaria.

El archivo `cargar_datos.py` contiene las siguientes funciones:

- `def cargar_comunas_en_cuarentena(path):` Retorna una **lista** de todas las comunas que se encuentran en cuarentena.
- `def cargar_clave_unica(path):` Retorna una **lista** con los todos los RUT de las personas que están solicitando la clave única.
- `def cargar_datos(path):` Retorna un diccionario donde la llave es el RUT de la persona y el valor es la *namedtuple* `Persona` con sus datos (`rut`, `nombre`, `clave` y `domicilio`).
- `def cargar_permiso_hora(path):` Retorna un diccionario donde la llave es el RUT de la persona y el valor es una *namedtuple* con los datos del permiso (`rut`, `hora`).
- `def cargar_permiso_supermercado(path):` Retorna un diccionario donde la llave es el RUT de la persona y el valor es una *namedtuple* con los datos del permiso (`rut`, `salida`, `llegada`).

Parte II: Levantar excepciones

Dado que la gente ya está comenzando a desesperarse en la cuarentena, **DCComisaria Virtual** quiere implementar un mecanismo para que no se puedan sacar más permisos que los permitidos al día y así evitar que la gente se aproveche de este sistema. En esta parte debes **levantar excepciones** en los siguientes métodos encontrados en el archivo `permisos.py`. **Atención:** En esta parte **no** debes utilizar `try`/`except`.

`def verificar_rut(rut, datos_registrados):` recibe el `rut` de una persona y un diccionario de nombre `datos_registrados`, cuya llave es el RUT de la persona y el valor es una *namedtuple* `Persona` (cargada en `def cargar_datos`). Deberás verificar si el RUT está escrito de la forma `XXXXXXXX-Y`. Si no es así, por ejemplo, si presenta puntos o no contiene el guión debes **levantar una excepción** del tipo `ValueError` con el siguiente mensaje:

```
1    'RUT viene con puntos o sin guión'
```

La función retorna una *namedtuple* con la información de la persona con ese RUT, si es que esta se encuentra en los datos registrados. **Esto último ya viene implementado.**

`def permiso_clave_unica(rut, datos_registrados):` en este método deberás verificar si el usuario que está pidiendo la clave única se encuentra registrado en el sistema. Esta información se encuentra en el

diccionario `datos_registrados` donde las llaves son los RUTs de las personas registradas y habilitadas para obtener la clave única y el valor es una instancia de la *namedtuple* `Persona`. En caso de que una persona no registrada trate de solicitar una clave deberás **levantar una excepción** del tipo `KeyError` con el siguiente mensaje:

```
'No puedes solicitar clave única. Impostor!'
```

`def permiso_asistencia_medica(hora):` Esta función recibe la hora a la que se debe asistir al recinto médico, pero el ministerio de salud ha establecido que usará la forma militar de nombrar la hora. En lugar de decir `12:00`, debe decir `1200`. Debes verificar que la hora recibida no contenga el separador `:` entre las horas y los minutos. Si esto no se cumple, deberás **levantar** una excepción del tipo `TypeError` con el siguiente mensaje:

```
'El formato de la hora es incorrecto.'
```

`def permiso_servicios_basicos(persona, solicitud, comunas_cuarentena):` Recibe una instancia de la *namedtuple* `Persona`, y debe verificar que el lugar de origen o destino esté en cuarentena. La lista `comunas_cuarentena` contiene las comunas que están bajo el decreto. Si esto no se cumple debes **levantar una excepción** del tipo `ValueError` con el siguiente mensaje:

```
'La comuna no está en cuarentena'
```

Ahora, si la comuna de origen o destino están efectivamente en cuarentena pero el lugar de origen no coincide con el domicilio de la persona se debe levantar **otra excepción** del tipo `ValueError` con el siguiente mensaje:

```
'No está saliendo desde su domicilio.'
```

Parte III: Capturar errores - try/except

En esta sección deberás manejar **las excepciones levantadas en la parte anterior** para todas las personas que decidieron sacar sus permisos. Debes utilizar el archivo `main.py` y completar los métodos:

```
def clave_unica(datos_clave_unica, datos_registrados)
def asistencia_medica(datos_horas, datos_registrados)
def asistencia_servicios_basicos(datos_permiso_supermercado, datos_registrados,
comunas_cuarentena)
```

Para cada permiso que se quiera solicitar debes verificar, **en primera instancia**, que su RUT este bien escrito, y después verificar que la solicitud sea válida. Si la solicitud es válida se debe imprimir las siguientes frases según sea el caso:

```
'Clave única obtenida con éxito'
'Permiso de Servicio de asistencia médica obtenido con éxito'
'Permiso de Servicios básicos obtenido con éxito'
```

Si no es así se deben capturar las excepciones correspondientes e imprimir el mensaje descrito en la sección anterior, según sea el caso.

A continuación veremos un ejemplo de todos los posibles *outputs* que se pueden imprimir, dependiendo de la función que se utilice:

```
# clave_unica
'Error: RUT viene con puntos o sin guión'
'Error: No puedes solicitar clave única. Impostor!'
'Clave única obtenida con éxito'
```

```
# asistencia_medica
'Error: RUT viene con puntos o sin guión'
'Error: El formato de la hora es incorrecto.'
'Permiso de Servicio de asistencia médica obtenido con éxito'
```

```
# asistencia_servicios_basicos
'Error: RUT viene con puntos o sin guión'
'La comuna no está en cuarentena'
'No está saliendo desde su domicilio.'
'Permiso de Servicios básicos obtenido con éxito'
```

Parte IV: Menú DCComisaria Virtual a prueba de errores

En esta última parte, deberás hacer que el programa sea a prueba de errores si no se ingresa una opción válida. Para ello, deberás crear una excepción personalizada llamada `ErrorPermiso`, que recibe el input del usuario `opcion` y `dic_opciones`. La clase entrega como mensaje:

```
'\n¡Error de Permiso!'
```

Además, se debe implementar el método `mostrar_opciones_validas` que imprime:

```
'La opción {opcion} no es válida'
'Las opciones válidas son: {string_opciones}'
```

donde `opcion` corresponde al *string* ingresado por el usuario y `string_opciones` es un *string* que separa a todas las llaves de `dic_opciones` por un `", "`, tal que quede `"1, 2, 3, 0"`. Es importante que trabajes con `dic_opciones` para hacer `string_opciones`, de caso contrario, no obtendrás puntaje en esta parte. Además debes terminar de implementar el `try`/`except`.

Requerimientos

- (1.00 pts) Parte I: reparar código.
  - (0.25 pts) Corregir `SyntaxError`.
  - (0.25 pts) Corregir `IndexError`.
  - (0.25 pts) Corregir `NameError`.
  - (0.25 pts) Corregir `ValueError`.
- (2.00 pts) Parte II: Levantar excepciones.
  - (0.5 pts) Implementar `def verificar_rut`.

    - (0.5 pts) Implementar `def permiso_clave_unica`.
    - (0.5 pts) Implementar `def permiso_asistencia_medica`.
    - (0.5 pts) Implementar `def permiso_servicios_basicos`.
- (2.00 pts) Parte III: Capturar errores excepciones.
    - (0.5 pts) Capturar excepción en `def clave_unica`.
    - (0.5 pts) Capturar excepción en `def asistencia_medica`.
    - (0.5 pts) Capturar excepción en `def asistencia_servicios_basicos`.
    - (0.5 pts) Capturar excepción para el *input* del menú.
- (1.00 pts) Parte IV: Menú a prueba de errores.