UNIVERSIDAD DE MURCIA

Tecnología de la Programación

Ejercicios adicionales

Tema 5: Excepciones y Entrada/Salida

Autores: Alejandro Buitrago López (alejandro.buitragol@um.es), Sergio López Bernal (slopez@um.es), Javier Pastor Galindo (javierpg@um.es), Jesús Damián Jiménez Re (jesusjr@um.es) y Gregorio Martínez Pérez (gregorio@um.es)

Dpto. de Ingeniería de la Información y las Comunicaciones

Curso 2025-2026

Última modificación:
26 de noviembre de 2025

Tema 5: Ejercicios adicionales

Índice

Estos ejercicios adicionales están diseñados para trabajar de manera complementaria y profunda los conceptos trabajados durante el **Tema 5**, como son las **excepciones** y la **entrada y salida de datos**.

5.1 Gestor de usuarios y configuraciones

Diseña un sistema en Python que implemente un gestor de usuarios y configuraciones de una aplicación. El programa debe permitir registrar usuarios, gestionar sus configuraciones personalizadas y guardar o cargar esta información desde archivos locales.

En primer lugar, se debe ofrecer la funcionalidad de registrar usuarios ingresando su nombre y edad desde la entrada estándar. Si el usuario proporciona una edad no válida, como un valor no numérico o fuera de un rango permitido (por ejemplo, entre 18 y 99 años), el programa debe lanzar una excepción personalizada. Estas excepciones deben ser capturadas para mostrar mensajes claros y permitir que el usuario reingrese los datos.

Cada usuario registrado debe tener configuraciones personalizadas, como el idioma preferido o la preferencia por un tema oscuro o ligero. Estas configuraciones deben almacenarse en un archivo JSON para su persistencia y poder cargarse nuevamente en futuras ejecuciones del programa. Si el archivo JSON no existe o está corrupto, el programa debe capturar las excepciones correspondientes y generar configuraciones por defecto. Antes de guardar y recuperar datos, el sistema debe verificar si el directorio donde se almacenarán los archivos existe y, si no, crearlo dinámicamente.

5.2 Gestor de Pokémon

Diseña un sistema en Python que gestione información sobre Pokémon descargada desde una API pública gratuita. El sistema debe permitir al usuario consultar información de distintos Pokémon, descargarla desde la API, almacenarla localmente en un archivo binario y mostrarla en un formato legible.

El programa debe conectarse a la API pública de PokeAPI para obtener información de los Pokémon. La URL base a utilizar será `https://pokeapi.co/api/v2/pokemon/id`, donde `id` debe ser reemplazado por el ID del Pokémon que se desea consultar (por ejemplo, 1 para Bulbasaur, 25 para Pikachu, 150 para Mewtwo, etc.). La información descargada debe incluir al menos el nombre del Pokémon, su altura y su peso. Si la conexión falla o el ID proporcionado no corresponde a un Pokémon válido, el programa debe capturar estas excepciones y notificar al usuario con mensajes claros.

La información de los Pokémon descargados debe almacenarse localmente en archivos binarios (*pickle*). Antes de guardar los datos, el programa debe verificar si ya existe un archivo binario con datos previos. Si el archivo existe, los datos deben cargarse y combinarse con los nuevos, evitando duplicados. Si el archivo no existe, el programa debe crearlo dinámicamente.

Finalmente, el sistema también debe permitir visualizar un listado de todos los Pokémon registrados, mostrando su nombre, altura y peso en un formato legible. Además, debe asegurar que los datos guardados y cargados sean consistentes y válidos. Para lograr esto, se deben manejar posibles errores de conexión, formato de datos incorrecto o problemas de lectura/escritura con el archivo binario.

5.3 Open Library

Diseña un sistema en Python que permita gestionar información sobre libros utilizando datos obtenidos desde una API pública gratuita. El sistema debe permitir al usuario buscar información de libros mediante la API de Open Library y guardar los resultados en un archivo de texto local para su posterior consulta. Si la conexión falla o el título proporcionado no corresponde a un libro válido, el sistema debe capturar estas excepciones y notificar al usuario con mensajes claros.

El sistema debe conectarse a la API de Open Library a través de la URL base `https://openlibrary.org/search.json?title=titulo`, donde `titulo` será el título del libro proporcionado por el usuario. Los resultados de la búsqueda deben incluir información relevante como el título del libro, el autor principal y el año de publicación, si están disponibles. Si el título ingresado tiene una longitud menor a 2 caracteres, el sistema debe notificar al usuario de manera clara.

Los libros obtenidos deben guardarse en un archivo de texto llamado con extensión `txt`, donde cada libro se almacenará en una línea con el formato:

```
Título | Autor | Año
```

Antes de guardar un libro, el sistema debe verificar si ya existe en el archivo para evitar duplicados. Esto asegura que la información almacenada sea única y consistente.

Finalmente, al iniciar, el sistema debe verificar si el archivo *txt* existe. Si lo encuentra, debe cargar y mostrar los datos almacenados previamente. Si el archivo no existe, el sistema debe inicializar una lista vacía y crear el archivo al guardar el primer libro. Además, debe proporcionar al usuario la opción de listar todos los libros registrados, mostrando la información en un formato legible.

5.4 Gestión de tareas

Diseña un sistema en Python que implemente un gestor de tareas para una aplicación. El sistema debe permitir registrar tareas, gestionar su estado (pendiente o completada) y guardar o cargar esta información desde un archivo JSON.

En primer lugar, el sistema debe permitir al usuario registrar nuevas tareas ingresando un título y una descripción desde la entrada estándar. Si el usuario deja el título vacío o la descripción es demasiado breve (por ejemplo, menos de 10 caracteres), el sistema debe lanzar una excepción personalizada. Estas excepciones deben ser capturadas para mostrar mensajes claros y permitir que el usuario reingrese los datos correctamente.

Cada tarea registrada debe incluir un estado, que será “pendiente” por defecto. El sistema debe ofrecer la opción de marcar tareas como completadas. La información sobre las tareas debe almacenarse en un archivo JSON para su persistencia. Si el archivo JSON no existe, el sistema debe crearlo automáticamente. Si el archivo está corrupto o no tiene un formato válido, el sistema debe capturar las excepciones correspondientes y notificar al usuario, inicializando una lista vacía de tareas como medida de recuperación.

Finalmente, el sistema debe cargar las tareas registradas previamente desde el archivo JSON y mostrar la lista de tareas existentes, si las hay. Además, debe garantizar que el directorio donde se almacenará el archivo JSON exista, creándolo dinámicamente si no está presente.