Clase 009 — Manejo de excepciones y context managers

Parte: 0 — Prerrequisitos · Fuente: Python Tutorial cap. 8 (Errors and Exceptions) · Ramalho, Fluent Python 2e — cap. 18 (Context Managers). · Duración estimada: 75 min.

Clase 009 — Manejo de excepciones y context managers

> *Parte: 0 — Prerrequisitos · Fuente: Python Tutorial cap. 8 (Errors and Exceptions) · Ramalho, Fluent Python 2e — cap. 18 (Context Managers). · Duración estimada: 75 min.*

Objetivo

Que el alumno maneje excepciones con criterio (sin except: pass), construya jerarquías de excepciones propias cuando aporta, y use context managers (with) — tanto los built-in como propios con @contextmanager — para garantizar limpieza de recursos. Sin esto, el código de carga de datos es una bomba de relojería.

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Diferenciar los 3 tipos de errores (Syntax, runtime exceptions, logical) y dónde se manejan.
2. Capturar excepciones específicas (except ValueError, no except:) y propagar las que no sabes manejar.
3. Crear una excepción propia heredando de la jerarquía estándar (class DatasetCorruptoError(Exception)).
4. Usar with para archivos, sesiones HTTP, transacciones DB.
5. Escribir un context manager propio con @contextmanager (timer, supress, change_dir).

Temas

#	Tema	Por qué importa
1	Jerarquía de excepciones built-in	BaseException → Exception → ValueError/Key
2	try/except/else/finally	Cada bloque tiene un rol específico.
3	Capturar específico, no genérico	except: esconde bugs.
4	Excepciones propias	Comunican intención en vez de cargar mensa
5	Context managers: protocolo __enter__/__ex	Garantiza cleanup.
6	@contextmanager de contextlib	Crear cms con función + yield.

Definiciones y características

Excepción

: Objeto que se 'lanza' (raise) cuando algo anómalo ocurre. Sube por el stack hasta que un except lo captura, o termina el programa. Todas heredan de BaseException; las que debes capturar heredan de Exception.

try/except/else/finally

: try: código riesgoso. except: maneja una excepción específica. else: corre si try no lanzó (raro pero útil). finally: cleanup garantizado, lanzó o no.

Jerarquía de excepciones

: BaseException → SystemExit / KeyboardInterrupt / Exception → ValueError / TypeError / LookupError (→ KeyError, IndexError) / OSError / ArithmeticError (→ ZeroDivisionError). Captura siempre la más específica que sepas manejar.

Excepción propia (custom)

: Subclase de Exception (o subclase específica). Permite tipificar errores de tu dominio (class DatasetCorruptoError(Exception): ...) en vez de strings. El caller puede except DatasetCorruptoError con precisión.

Context manager

: Objeto con __enter__ y __exit__ que se usa con with. Garantiza setup/teardown (abrir/cerrar archivo, conectar/desconectar BD, lock/unlock). El __exit__ corre incluso si hay excepción.

@contextmanager

: Decorador de contextlib que convierte una función con yield en context manager. Pre-yield = __enter__, post-yield = __exit__. Mucho más corto que escribir la clase completa.

Dataset / recursos

Archivo temporal generado en el notebook. Sin descarga.

Ejercicios

1. Captura específica. Escribe una función parse_int_safe(s, default=0) que use try/except solo para ValueError. Demuestra que no esconde otros errores (ej. TypeError si pasas un dict).

2. Excepción propia. Define class DatasetCorruptoError(Exception) con un atributo linea. Lanzala desde una función cargar_csv cuando una línea no tenga el número correcto de columnas.

3. with para archivo. Lee un archivo línea por línea contando palabras. Compara con la versión sin with (manual open/close) y muestra qué pasa si hay excepción a mitad.

4. Context manager propio: timer. Con @contextmanager, escribe with timer("carga"): que imprima cuánto duró el bloque.

5. Context manager: change_dir. with cd("/tmp"): cambia de directorio al entrar y vuelve al salir — incluso si hay excepción.

Homework verificable

Notebook con: (a) parse_int_safe con tests de los 3 casos (válido, inválido, otro tipo); (b) DatasetCorruptoError usada en una función cargar_csv que valida #columnas; (c) decorador-context manager timer aplicado a 2 operaciones; (d) cd context manager.

Criterio de aceptación: Excepciones se capturan solo donde sabes manejarlas. timer reporta segundos correctamente.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar

except: o except Exception: ciego esconde	Cualquier error queda silenciado (incluso
finally con return traga la excepción	Si finally ejecuta return, la excepción de
with open(f) as f, open(g) as g: falla por	Esa sintaxis requiere Python 3.10+. Fix: e
Capturé KeyError pero el código sigue romp	Tu except está más arriba del try, o el er
Excepción dentro de generator no se captur	Las excepciones en generators son tricky;

Preguntas frecuentes

¿except Exception o except?

Casi siempre except Exception — más explícito. except: desnudo captura también KeyboardInterrupt y SystemExit, lo cual rompe Ctrl+C y sys.exit().

¿Cuándo creo una excepción propia?

Cuando el caller necesita diferenciar este error de otros. raise ValueError('CSV corrupto') obliga al caller a parsear strings; raise DatasetCorruptoError(linea=42) le da un tipo y un atributo concreto.

¿with solo para archivos?

No — para CUALQUIER recurso que necesite cleanup. Files (open), conexiones DB (engine.connect()), locks (threading.Lock), sesiones HTTP (requests.Session), transacciones (db.atomic()), timing (with timer(...)).

¿Debo capturar y re-lanzar para añadir contexto?

Sí, con raise ExceptionNueva(...) from e — preserva el stacktrace original como '__cause__'. El log muestra ambos errores en cadena.

¿pass en except es siempre malo?

Casi siempre sí. Si tienes que silenciar, al menos log.warning(...). Solo OK cuando la excepción es esperada (except FileNotFoundError: pass al limpiar archivos opcionales).

Referencias

- Python Tutorial — Errors and Exceptions
- Ramalho, Fluent Python 2e — cap. 18 Context Managers and else Blocks.
- contextlib docs

Siguiente clase

Clase 010 — OOP básico, dataclasses, herencia

Apéndice: notebook (primer bloque)

Primera celda ejecutable del notebook de la clase.

```
import os, time, tempfile
from contextlib import contextmanager
from pathlib import Path
```

Archivos complementarios

- notebook.ipynb