

# CLASE 14

## Excepciones en Python

*Cuando las cosas salen mal... y cómo manejarlo*





# EL PROBLEMA - Demostración en Vivo

## Escenario 1: Conversión inválida

```
edad = int("hola")  
# ValueError: invalid literal for int()
```

## Escenario 3: Archivo inexistente

```
archivo = open("datos.json")  
# FileNotFoundError: No such file
```

## Escenario 2: División entre cero

```
resultado = 10 / 0  
# ZeroDivisionError: division by zero
```



El programa MUERE.  
Sin avisar. Sin guardar. Sin oportunidad de recuperarse.

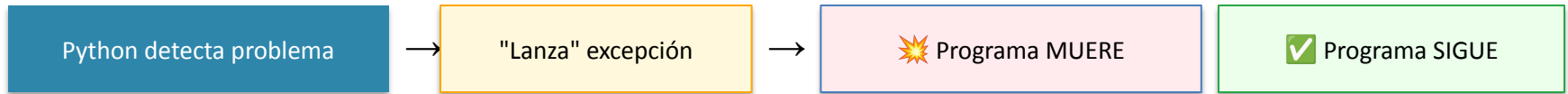


¿Es aceptable que un programa profesional simplemente... muera?

# ¿Qué es una Excepción?

EXCEPCIÓN = Error que ocurre DURANTE la ejecución del programa


## Flujo de una excepción:



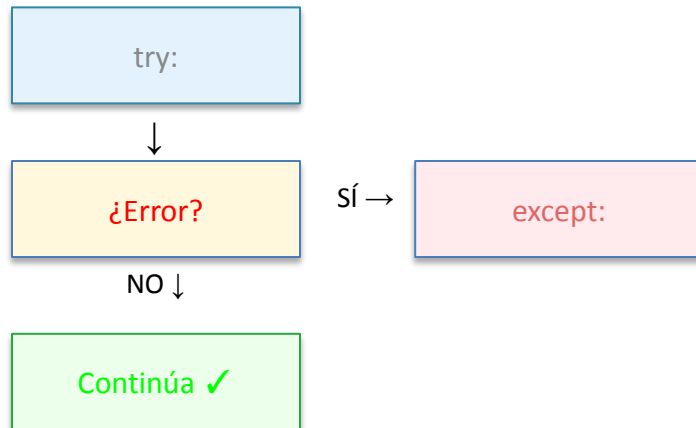
# TRY / EXCEPT - Atrapar errores

## Sintaxis básica:

```
try:  
    # Código que PODRÍA fallar  
    resultado = 10 / 0  
except:  
    # Qué hacer SI falla  
    print("¡Algo salió mal!")
```

 try = "intentar"  
except = "excepto si hay error"

## Flujo:



El programa NO murió. Atrapó el error y SIGUIÓ. ✓

# EXCEPT Específico

## Problema con except: solo (sin tipo):

```
# MAL - except genérico
try:
    numero = int("hola")
except:
    print("Algo salió mal")
# ¿Qué salió mal? NO SABEMOS
```



Atrapa TODO → No sabes qué pasó

## Mejor - except con tipo específico:

```
# BIEN - except específico
try:
    numero = int("hola")
except ValueError:
    print("No es un número válido")
# Sabemos EXACTAMENTE qué pasó
```



Solo atrapa ESE error → Sabes qué pasó



Es como ir al doctor:

"Me siento mal" vs "Me duele la rodilla derecha"

¿Cuál ayuda más al doctor a diagnosticar?



# EJERCICIO 1: Input Seguro

## MISIÓN:

1. Pedir al usuario un número con `input()`
2. Si escribe algo que NO es número → mostrar error
3. Si es número válido → mostrar el doble



### PISTAS:

- Usa `try/except`
- El error cuando `int()` falla es: `ValueError`
- Recuerda: `int(input("..." ))`

## Ejemplo de uso:

Escribe un número: 5  
El doble es: 10

Escribe un número: hola  
✗ Eso no es un número válido



Equipos de 2-3



Archivo: `ejercicio1_input_seguro.py`

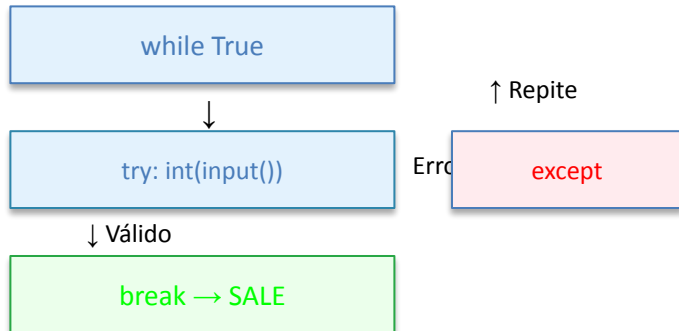


## BONUS: Pedir hasta que sea válido

```
while True:
    try:
        numero = int(input("Escribe un número: "))
        break # Si llegó aquí, es válido → salir del loop
    except ValueError:
        print("❌ Eso no es un número. Intenta de nuevo.")

print(f"El doble es: {numero * 2}")
```

### Flujo:



PATRÓN MUY COMÚN:

while True + try/except + break = "Sigue pidiendo hasta que esté bien"

Este patrón lo van a usar en TODOS sus programas interactivos.



# BREAK

10 MINUTOS



# Errores Comunes en Python

<b>ValueError</b>	Valor incorrecto	<code>int("hola")</code>
<b>TypeError</b>	Tipo incorrecto	<code>"hola" + 5</code>
<b>KeyError</b>	Clave no existe en dict	<code>mi_dict["clave_falsa"]</code>
<b>IndexError</b>	Índice fuera de rango	<code>lista[99]</code> (con 3 items)
<b>FileNotFoundError</b>	Archivo no existe	<code>open("no_existe.json")</code>
<b>ZeroDivisionError</b>	División entre cero	<code>10 / 0</code>



No necesitan memorizar. El NOMBRE del error les dice qué pasó:  
ValueError = el VALOR no sirve | KeyError = la CLAVE no existe



# Ejercicio Rápido: Identificar Errores

## Caso A:

```
datos = {"nombre": "Ana"}  
print(datos["telefono"])
```

## Caso B:

```
numeros = [1, 2, 3]  
print(numeros[10])
```

## Caso C:

```
edad = int("veinte")
```

## Caso D:

```
with open("config.json") as f:  
    datos = f.read()
```

## Caso E:

```
resultado = "10" + 5
```

**?** Para cada caso: ¿Qué excepción? ¿Por qué?  
(Respuestas en grupo - 5 minutos)



Pista: A veces el error dice que tu CÓDIGO está mal.  
No siempre la solución es try/except. A veces es ARREGLAR EL BUG.

# Múltiples EXCEPT

```
def dividir_numeros():  
    try:  
        a = int(input("Primer número: "))  
        b = int(input("Segundo número: "))  
        resultado = a / b  
        print(f"Resultado: {resultado}")  
    except ValueError:  
        print("✗ Debes escribir un número")  
    except ZeroDivisionError:  
        print("✗ No puedes dividir entre cero")
```

## Flujo:

Python revisa cada except EN ORDEN hasta encontrar uno que coincida.



Puedes tener tantos except como necesites

## Capturar el mensaje con 'as e':

```
except ValueError as e:  
    print(f"Error: {e}")  
  
# Output:  
# Error: invalid literal for  
# int() with base 10
```



'as e' guarda el error en la variable e.  
Útil para debugging y logs.



## EJERCICIO 2: Calculadora Robusta

### MISIÓN: Crear función calculadora() que:

1. Pida dos números y una operación (+, -, \*, /)
2. Maneje ValueError (números inválidos)
3. Maneje ZeroDivisionError (división entre cero)
4. Use while para repetir hasta "salir"



#### PISTAS:

- Usa el patrón while True + break
- float() en vez de int() para decimales
- continue para volver al inicio del while



Equipos de 2-3



Archivo: ejercicio2\_calculadora.py

### Ejemplo de uso:

Primer número: hola  
✗ Debes escribir números válidos

Primer número: 10  
Segundo número: 0  
Operación: /

✗ No puedes dividir entre cero

Primer número: 10  
Segundo número: 3  
Operación: +  
✓ Resultado: 13.0



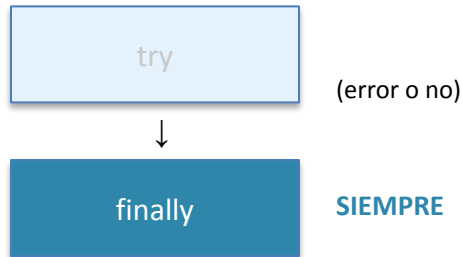
# FINALLY - Se ejecuta SIEMPRE

```
try:
    archivo = open("datos.txt", "r")
    contenido = archivo.read()
    # Procesar contenido...
except FileNotFoundError:
    print("❌ Archivo no encontrado")
finally:
    print("🔒 Esto se ejecuta SIEMPRE")
# Cerrar conexiones, limpiar recursos, etc.
```



with open() ya hace el "finally" por ti.  
Por eso usamos with para archivos.

## Flujo:



## Usos comunes de finally:

- Cerrar archivos
- Cerrar conexiones a BD
- Limpiar recursos temporales
- Guardar estado



finally es OPCIONAL pero muy útil para limpieza

# ✨ ELSE - Solo si NO hubo error

```
try:
    numero = int(input("Número: "))
except ValueError:
    print("✗ No es un número")
else:
    print(f"✓ Perfecto, escribiste {numero}")
finally:
    print("Fin del intento")
```

💡 REGLA: try solo lo MÍNIMO que puede fallar.  
else para el código que DEPENDE del try pero no puede fallar.  
No es obligatorio, pero es buena práctica.

## Estructura completa:

try: → Código que puede fallar

except: → Si HUBO error

else: → Si NO hubo error

finally: → SIEMPRE (opcional)

✓ try → except → else → finally (en ese orden)



# Buenas Prácticas vs Antipatrones



## HACER:

- Siempre especificar el tipo de excepción
  - Usar "as e" para obtener detalles
  - Manejar cada tipo de error diferente
- try solo alrededor del código que puede fallar
- Retornar valores por defecto cuando tenga sentido



## EVITAR:

- except: solo (sin tipo) → atrapa TODO
- try: con 50 líneas → difícil saber qué falló
  - Usar excepciones para flujo normal
  - Ignorar errores: except: pass



## NUNCA:

try:  
# 100 líneas de código  
except:  
pass

Es como tapan el check engine del auto con cinta adhesiva 



REGLA DE ORO: "Atrapa solo los errores que SABES manejar."



# Mejorando cargar\_tareas()

## ANTES (Clase 13):

```
def cargar_tareas(archivo="tareas.json"):
    with open(archivo) as f:
        return json.load(f)

# 💥 Si no existe → CRASH
# 💥 Si está corrupto → CRASH
```



2 líneas extra de try/except y su programa  
NUNCA se rompe.  
No agregamos funcionalidad. Agregamos  
RESILIENCIA.

## DESPUÉS (Clase 14):

```
def cargar_tareas(archivo="tareas.json"):
    try:
        with open(archivo) as f:
            return json.load(f)
    except FileNotFoundError:
        print("📁 Primera vez. Creando...")
        return []
    except json.JSONDecodeError:
        print("⚠️ Archivo corrupto.")
        return []

# ✅ NUNCA se rompe
```



## RESULTADO:

- Si el archivo no existe → Empieza vacío
- Si el archivo está corrupto → Empieza vacío
- El usuario NUNCA ve un crash



# Mejorando el Menú

## ANTES:

```
opcion = int(input("Opción: "))  
  
# 💥 CRASH si escribe "hola"
```

## DESPUÉS - Función reutilizable:

```
def pedir_opcion(mensaje="Opción: ",  
                 minimo=0, maximo=5):  
    while True:  
        try:  
            opcion = int(input(mensaje))  
            if minimo <= opcion <= maximo:  
                return opcion  
            else:  
                print(f"❌ Entre {minimo} y {maximo}")  
        except ValueError:  
            print("❌ Escribe un número válido")
```

## Uso:

```
# En el menú principal:  
opcion = pedir_opcion(  
    "Elige (1-5): ",  
    minimo=1,  
    maximo=5  
)  
  
# ✅ NUNCA falla
```



Esta función es **REUTILIZABLE** en cualquier proyecto.

Combina:

- Validación de TIPO (try/except ValueError)
- Validación de RANGO (if minimo <= opcion <= maximo)



TAREA: Agreguen estas mejoras a su proyecto de Clase 11/13

# TODO SE CONECTA

## El viaje hasta aquí:

Clase 3: input() → "¿Y si escriben mal?"

Clase 7: diccionarios → "¿Y si la clave no existe?"

Clase 8: funciones → "¿Y si los argumentos son inválidos?"

Clase 13: archivos/JSON → "¿Y si el archivo no existe?"

Clase 14: EXCEPCIONES → "Ahora saben manejarlo TODO"

## Próxima clase:

Clase 15: Excepciones II + Debugging

- raise - lanzar excepciones propias
  - Excepciones personalizadas
  - Debugging con PEP 8

## Lo que aprendieron hoy:

- ✓ try/except básico
- ✓ Excepciones específicas (ValueError, etc.)
  - ✓ Múltiple except
  - ✓ finally (siempre se ejecuta)
  - ✓ else (solo si no hubo error)
- ✓ Patrón while + try/except + break



TAREA PARA LA PRÓXIMA:

Revisar proyecto de Clase 11/13 y agregar:

- cargar\_tareas() con try/except
- Menú con validación robusta

"Hasta Clase 13, sus programas eran FRÁGILES.  
Hoy aprendieron a manejar cuando las cosas salen MAL."