Manejo de Excepciones en Python

Aprende a controlar los errores en tus programas y crear código más robusto y profesional. El manejo adecuado de excepciones es fundamental para desarrollar aplicaciones confiables.

¿Qué es una excepción?

En Python, una **excepción** es un error que ocurre durante la ejecución de un programa. Cuando Python encuentra una situación que no puede manejar, "lanza" una excepción.

Si no manejamos esta excepción, el programa se detendrá abruptamente y mostrará un mensaje de error. Esto puede ser frustrante tanto para el programador como para el usuario final.

Las excepciones no siempre significan que el código está mal escrito. A veces representan situaciones imprevistas, como cuando un usuario ingresa datos incorrectos o un archivo no existe.

Ejemplo común

```
print(10 / 0)
```

Resultado:

```
ZeroDivisionError: division by zero
```

¿Por qué manejar excepciones?

Evitar caídas del programa

Tu aplicación continúa funcionando incluso cuando algo sale mal, mejorando la experiencia del usuario.

Mensajes amigables

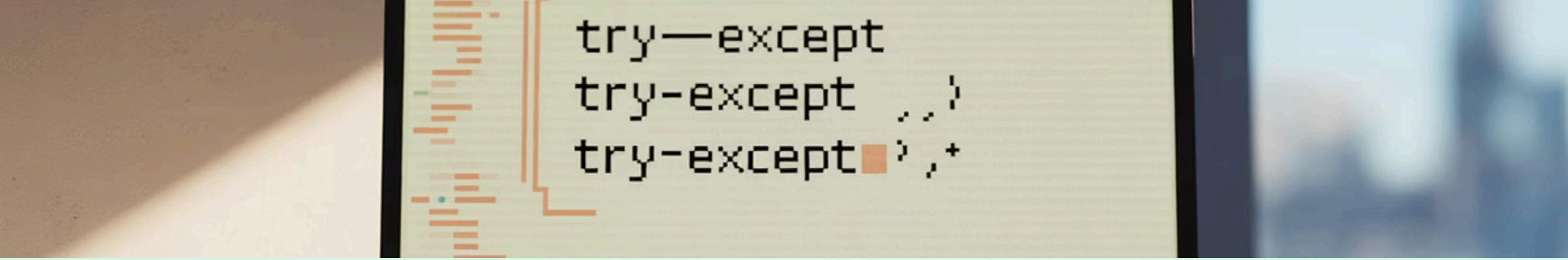
Puedes mostrar mensajes claros y útiles en lugar de errores técnicos confusos.

Facilitar depuración

Puedes registrar errores para analizarlos después y mejorar tu código continuamente.

Código profesional

El manejo de excepciones es señal de un programador experimentado que piensa en todos los escenarios posibles.



La estructura try-except básica

Para manejar excepciones, Python nos ofrece el bloque `try-except`. Es como decirle al programa: "intenta hacer esto, pero si falla, haz esto otro en su lugar".

try

Aquí va el código que *podría* generar un error. Python intentará ejecutarlo normalmente.

except

Si ocurre un error en el bloque try, Python salta aquí y ejecuta este código alternativo.

Primer ejemplo práctico

Veamos cómo convertir texto a número de forma segura. Este es uno de los errores más comunes en programación.

❌ Sin manejo de errores

```
numero = int("hola")
print(numero)
```

✅ Con manejo de errores

```
try:
    numero = int("hola")
    print(numero)
except ValueError:
    print("Error: No se puede convertir texto a número")
```

Resultado: El programa continúa y muestra un mensaje claro y útil.

Ampliando el control: else y finally

Python nos permite añadir dos bloques adicionales para tener un control total sobre el flujo del programa cuando manejamos excepciones.

01

try

Código que puede generar excepciones

02

except

Se ejecuta **solo si** hubo un error en try

03

else

Se ejecuta **solo si no** hubo ningún error

04

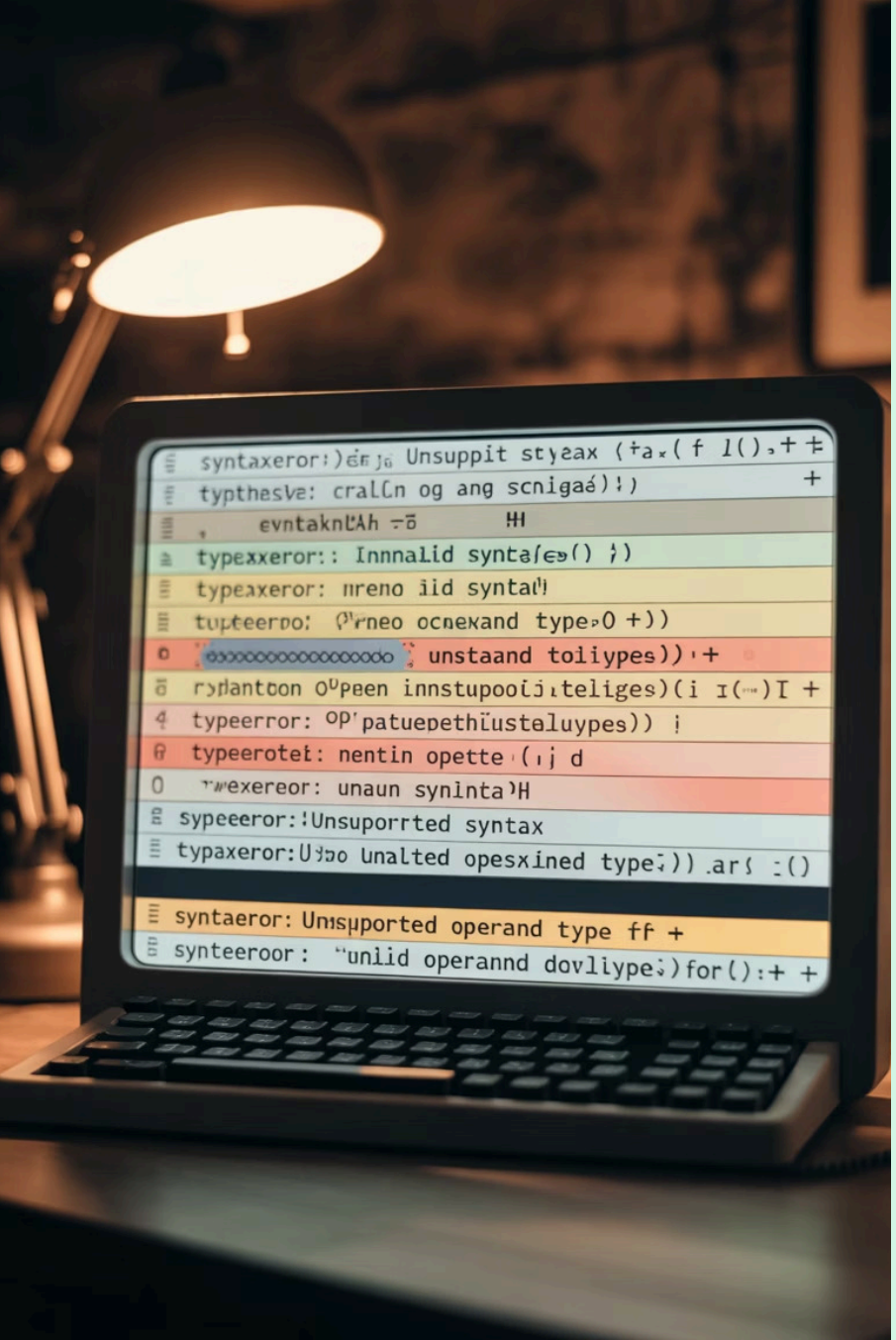
finally

Se ejecuta **siempre**, haya o no haya habido error

El bloque `finally` es especialmente útil para tareas de limpieza, como cerrar archivos o conexiones a bases de datos, que deben realizarse siempre.

Ejemplo completo con todos los bloques

```
try:
    n = int(input("Ingresa un número: "))
    print("El doble es:", n*2)
except ValueError:
    print("Error: Debes ingresar un número válido")
else:
    print("Todo salió bien. No hubo excepciones.")
finally:
    print("La ejecución del bloque try-except ha terminado.")
```



Las excepciones más comunes

Conocer los tipos de excepciones te ayudará a anticipar problemas y escribir código más defensivo.

Catálogo de excepciones frecuentes

ValueError

Ocurre cuando una función recibe un argumento con el tipo correcto pero un valor inapropiado.

```
int("abc")  # No se puede convertir
```

ZeroDivisionError

Se produce al intentar dividir un número entre cero, una operación matemática imposible.

```
resultado = 10 / 0
```

IndexError

Sucede al intentar acceder a una posición que no existe en una lista o tupla.

```
lista = [1, 2, 3]
print(lista[10]) # Fuera de rango
```

Más excepciones importantes

KeyError

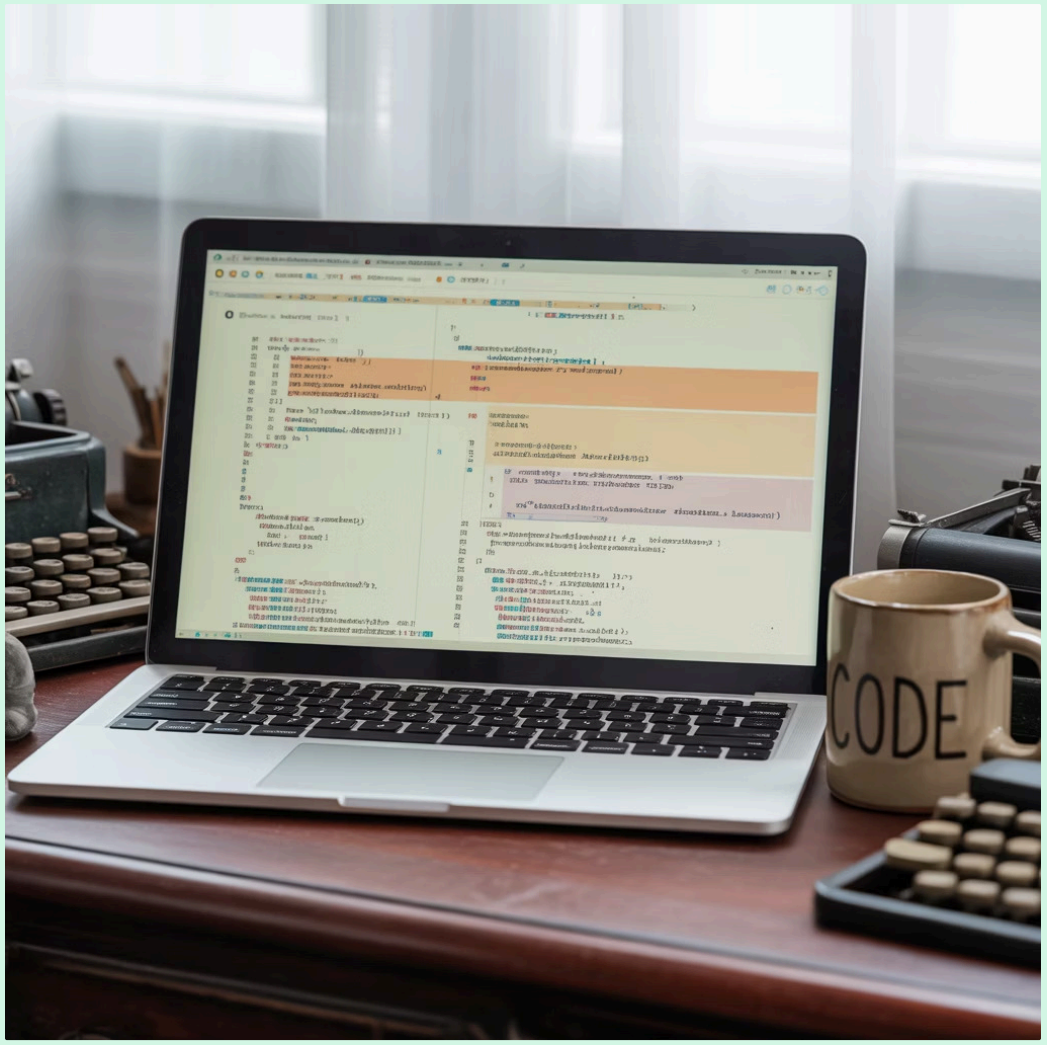
Intentar acceder a una clave que no existe en un diccionario.

```
persona = {"nombre": "Ana"}
print(persona["edad"])
```

TypeError

Realizar operaciones entre tipos de datos incompatibles.

```
resultado = "5" + 3  # str + int
```



Cada tipo de excepción nos da información valiosa sobre qué salió mal. Aprender a identificarlas rápidamente mejorará tu capacidad de depuración.

Buenas prácticas en el manejo de excepciones

- [ ] **Sé específico con las excepciones**

  Captura solo las excepciones que esperas y sabes cómo manejar. Usar `except:` sin especificar el tipo puede ocultar errores inesperados.

- [ ] **No abuses del try-except**

  El manejo de excepciones no debe usarse para controlar el flujo normal del programa. Úsalo solo para situaciones excepcionales.

- [ ] **Proporciona mensajes útiles**

  Los mensajes de error deben ser claros y orientar al usuario sobre qué hacer para solucionarlo.

- [ ] **Mantén los bloques try pequeños**

  Solo incluye en el bloque try el código que realmente puede generar la excepción, facilitando la identificación del problema.

Ejercicio práctico

Ha llegado el momento de poner en práctica lo aprendido con un ejercicio guiado.

📌 Desafío: Calculadora de cuadrados

> **Objetivo:** Escribe un programa que pida al usuario un número y muestre su cuadrado. El programa debe manejar correctamente el caso en que el usuario ingrese texto en lugar de un número.

Solución

```
try:
    numero = int(input("Ingresa un número: "))
    print("El cuadrado es:", numero ** 2)
except ValueError:
    print("Entrada inválida. Por favor, ingresa un número entero.")
```

Ejemplos de ejecución del ejercicio

✅ Ejecución exitosa

```
Ingresa un número: 5
El cuadrado es: 25
```

El usuario ingresó un número válido, por lo que el programa calculó correctamente $5^2 = 25$.

⚠️ Manejo de error

```
Ingresa un número: hola
Entrada inválida. Por favor, ingresa un número entero.
```

El usuario ingresó texto en lugar de un número, por lo que el programa capturó el `ValueError` y mostró el mensaje de error correspondiente.

Resumen y próximos pasos

Has aprendido

- Qué son las excepciones y por qué ocurren
- Cómo usar try-except para manejar errores
- Los bloques else y finally para mayor control
- Las excepciones más comunes en Python
- Buenas prácticas para código robusto

Continúa practicando

Intenta mejorar el ejercicio añadiendo validación para números negativos, o crea un programa que maneje múltiples tipos de excepciones. La práctica constante es clave para dominar el manejo de excepciones.

Recuerda: Un buen programador no solo escribe código que funciona cuando todo va bien, sino que también anticipa y maneja elegantemente los casos en que las cosas pueden salir mal.