### Manejo de la Excepción FileNotFoundError

Un problema común cuando se trabaja con archivos es la gestión de archivos perdidos. El archivo que buscas puede estar en una ubicación diferente, el nombre del archivo puede estar mal escrito o puede no existir en absoluto. Puedes manejar todas estas situaciones con un bloque try-except.

Intentemos leer un archivo que no existe. El siguiente programa intenta leer el contenido de *Alice in Wonderland*, pero no he guardado el archivo *alice.txt* en el mismo directorio que *alice.py*:

**alice.py**

from pathlib import Path  
  
path = Path('alice.txt')  
contents = path.read\_text(encoding='utf-8')

Ten en cuenta que aquí estamos utilizando read\_text() de una forma ligeramente distinta a la que has visto antes. El argumento encoding es necesario cuando la codificación por defecto de tu sistema no coincide con la codificación del archivo que se está leyendo. Lo más probable es que esto ocurra al leer de un archivo que no se creó en tu sistema.

Python no puede leer de un archivo que falta, por lo que lanza una excepción:

Traceback (most recent call last):  
❶ File "alice.py", line 4, in <module>  
❷ contents = path.read\_text(encoding='utf-8')  
 ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^  
 File "/.../pathlib.py", line 1056, in read\_text  
 with self.open(mode='r', encoding=encoding, errors=errors) as f:  
 ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^  
 File "/.../pathlib.py", line 1042, in open  
 return io.open(self, mode, buffering, encoding, errors, newline)  
 ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^  
❸ FileNotFoundError: [Errno 2] No such file or directory: 'alice.txt'

Éste es un rastreo más largo que los que hemos visto anteriormente, así que veamos cómo dar sentido a rastreos más complejos. A menudo es mejor empezar por el final del rastreo. En la última línea, podemos ver que se lanzó una excepción FileNotFoundError ❸. Esto es importante porque nos dice qué tipo de excepción utilizar en el bloque except que vamos a escribir.

Mirando hacia atrás, cerca del principio del rastreo ❶, podemos ver que el error se produjo en la línea 4 del archivo *alice.py*. La siguiente línea muestra la línea de código que causó el error ❷. El resto del rastreo muestra código de las bibliotecas que intervienen en la apertura y lectura de archivos. Normalmente no es necesario que leas o comprendas todas estas líneas en un rastreo.

Para tratar el error que se está produciendo, el bloque try comenzará con la línea que se identificó como problemática en el rastreo. En nuestro ejemplo, es la línea que contiene read\_text():

from pathlib import Path  
  
path = Path('alice.txt')  
try:  
 contents = path.read\_text(encoding='utf-8')  
❶ except FileNotFoundError:  
 print(f"Sorry, the file {path} does not exist.")

En este ejemplo, el código del bloque try produce un FileNotFoundError, así que escribimos un bloque except que coincide con ese error ❶. Python ejecuta entonces el código de ese bloque cuando no encuentra el archivo, y el resultado es un mensaje de error amigable en lugar de un rastreo:

Sorry, the file alice.txt does not exist.

El programa no tiene nada más que hacer si el archivo no existe, así que ésta es toda la salida que vemos. Continuemos con este ejemplo y veamos cómo puede ayudar el manejo de excepciones cuando trabajas con más de un archivo.

[anterior](c10_24.html)[Subtema 25 de 37: (Ver todo)](c10.html)[siguiente](c10_26.html)