Gestores de contexto

Los gestores de contexto son estructuras en Python que te ayudarán muchísimo al trabajar con archivos. Si usas gestores de contexto, no necesitarás recordar cerrar el archivo al final de tu programa y tendrás acceso al archivo solamento en la parte específica del programa que escojas, lo cual disminuye el riesgo de errores (bugs).

Sintaxis

Este es un ejemplo de un gestor de contexto usado para trabajar con archivos:

Dato: El cuerpo de un gestor de contexto debe estar indentado, al igual que indentamos el cuerpo de los ciclos, de las funciones y de las clases. Si el código no está indentado, no se considerará parte del gestor de contexto.
El archivo se cierra automáticamente cuando se completa la ejecución del cuerpo del gestor de contexto:

```
with open("<ruta del archivo>", "<modo>") as <variable>:
    # Trabajando con el archivo...

# ¡El archivo está cerrado aquí!
```

Ejemplo

Aquí tenemos un ejemplo:

```
with open("datos/nombres.txt", "r+") as f:
    print(f.readlines())
```

Este gestor de contexto abre el archivo `nombres.txt` para operaciones de lectura y escritura y asigna el objeto archivo a la variable `f`. Esta variable se usa en el cuerpo del gestor de contexto para trabajar con el objeto archivo.

Intentar leerlo nuevamente

Luego de haber completado la ejecución del cuerpo del gestor de contexto, el archivo se cierra automáticamente, así que no puede ser leído si no se abre nuevamente.

Aquí tenemos una línea de código que intenta leer el archivo luego de cerrarlo:

```
with open("datos/nombres.txt", "r+") as f:
    print(f.readlines())

print(f.readlines()) # Intentar leer el archivo nuevamente, fuera del gestor de
contexto.
```

Veámos qué ocurre:

```
Traceback (most recent call last):
  File "<path>", line 21, in <module>
    print(f.readlines())
ValueError: I/O operation on closed file.
```

Se genera este error porque estamos intentando leer un archivo que ya fue cerrado.

El gestor de contexto hace todo el trabajo pesado por nosotros, es fácil de leer y muy conciso.

Cómo manejar excepciones al trabajar con archivos

Cuando trabajas con archivos pueden ocurrir errores. Quizás porque no tendremos los permisos necesarios para modificar o leer un archivo, o quizás porque el archivo no existe.
Necesitamos predecir estas posibles circunstancias y manejarlas en el programa para evitar cierres o problemas inesperados que pueden afectar la experiencia del usuario.

Veamos algunas de las excepciones (errores durante la ejecución del programa) más comunes que puedes encontrar al trabajar con archivos:

FileNotFoundError

Ocurre cuando un archivo o directorio es solicitado pero no existe.

Texto original en inglés:
Raised when a file or directory is requested but doesn’t exist.

Por ejemplo, si el archivo que intentas abrir no existe en tu directorio actual de trabajo (current working directory):

```
f = open("nombres.txt")
```

Verás el siguiente error:

```
Traceback (most recent call last):
  File "<ruta>", line 8, in <módulo>
```

```
    f = open("nombres.txt")
FileNotFoundError: [Errno 2] No such file or directory: 'nombres.txt'
```

Veamos este error línea por línea:

- `File "<ruta>", line 8, in <módulo>`. Esta línea te dice que el error fue generado cuando el código en el archivo ubicado en `<ruta>` se estaba ejecutando. Específicamente, cuando la línea 8 estaba siendo ejecutada en `<módulo>`.
- `f = open("nombres.txt")`. Esta es la línea que generó el error.
- `FileNotFoundError: [Errno 2] No such file or directory: 'nombres.txt'` . Esta línea dice que ocurrió una excepción `FileNotFoundError` porque el archivo o directorio `nombres.txt` no existe.

Dato: Los mensajes de error son muy descriptivos en Python

PermissionError

Esta es otra excepción común al trabajar con archivos.
Ocurre cuando se intenta ejecutar una operación sin los permisos de acceso adecuados - por ejemplo, permisos del sistema de archivos.

Texto original en inglés:

Raised when trying to run an operation without the adequate access rights - for example filesystem permissions.

La excepción ocurre cuando intentas leer o modificar un archivo pero no tienes los permisos necesarios para acceder a ese archivo.

Si intentas hacerlo, verás el siguiente error:

```
Traceback (most recent call last):
  File "<ruta>", line 8, in <module>
    f = open("<ruta del archivo>")
PermissionError: [Errno 13] Permission denied: 'datos'
```

IsADirectoryError

Ocurre cuando una operación de archivos se intenta ejecutar en un directorio.

Texto original en inglés:

Raised when a file operation is requested on a directory.

Esta excepción en particular ocurre cuando intentas abrir o trabajar con un directorio en lugar de un archivo, así que debes tener mucho cuidado al momento de escoger la ruta que pasarás como argumento a los métodos de objetos archivo.

Cómo manejar excepciones

Para manejar estas excepciones, puedes usar una sentencia **try/except.**

Con esta sentencia, puedes indicarle a tu programa qué hacer en caso de que algo ocurra.
Esta es la sintaxis básica:

```
try:
        # Intenta ejecutar este código.
except <tipo_de_excepción>:
        # Si ocurre una excepción de este tipo, detén el proceso inmediatamente
y salta a este bloque de código.
```

Aquí puedes ver un ejemplo con `FileNotFoundError`:

```
try:
    f = open("nombres.txt")
except FileNotFoundError:
    print("El archivo no existe.")
```

Este código básicamente dice:

- Intenta abrir el archivo `nombres.txt`.
- Si una excepción `FileNotFoundError` ocurre, ¡no colapses! Solo muestrale al usuario un mensaje describiendo lo que ocurrió.

Dato: Puedes escoger cómo manejar la situación escribiendo el código apropiado en el bloque `except`. Quizás puedas crear un archivo nuevo si no existe en la ruta indicada.

Para cerrar el archivo automáticamente luego de la tarea (independientemente de si ocurrió una excepción o no en el bloque **`try`**) puedes agregar la cláusula **`finally`**.

```
try:
        # Intenta ejecutar este código.
except <tipo_de_excepción>:
        # Si ocurre una excepción de este tipo, detén el proceso inmediatamente
y salta a este bloque de código.
finally:
        # Haz esto luego de ejecutar el código, incluso si ocurrió una
excepción.
```

Este es un ejemplo:

```
try:
    f = open("nombres.txt")
except FileNotFoundError:
    print("El archivo no existe.")
finally:
    f.close()
```

Hay muchas formas de personalizar la sentencia try/except/finally e incluso puedes añadir una cláusula `else` para ejecutar un bloque de código solamente si no ocurren excepciones en el bloque `try`.