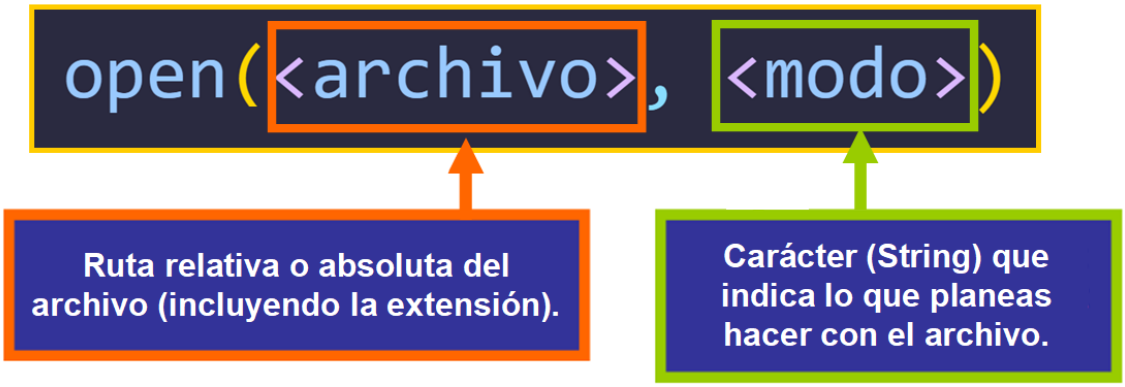
Manejo de archivos

Trabajar con archivos:

Una de las funciones más importantes que necesitarás usar a medida que trabajas con archivos en Python es **`open()`**, una función incorporada (built-in) que abre un archivo y permite que tu programa tenga acceso a él.

Esta es la sintaxis básica:



Dato: estos son los dos argumentos más comúnmente usados para llamar a esta función. Existen seis argumentos adicionales que son opcionales.

Modo:

Carácter	Significado
`'r'`	abierto para lectura (por defecto)
`'w'`	abierto para escritura, truncando primero el fichero
`'x'`	abierto para creación en exclusiva, falla si el fichero ya existe
`'a'`	abierto para escritura, añadiendo al final del fichero si este existe
`'b'`	modo binario
`'t'`	modo texto (por defecto)
`'+'`	abierto para actualizar (lectura y escritura)

Primer parámetro: file (archivo)

El primer parámetro de la función **`open()`** es **`file`** (archivo), la ruta (path) absoluta o relativa del archivo con el cual estás intentando trabajar.
Normalmente usamos la ruta relativa, la cual indica dónde está ubicado el archivo en relación a la ubicación del archivo de Python (script) que llama a la función **`open()`**.
Por ejemplo, la ruta en esta llamada a la función **`open()`**:

```
open("nombres.txt") # La ruta relativa es "nombres.txt"
```

Solo contiene el nombre del archivo.
Si el archivo se encuentra dentro de la carpeta “datos”, y nos ubicamos fuera de ella:
Entonces necesitamos usar una ruta específica para indicarle a la función que el archivo de texto está dentro de otra carpeta:
Por lo tanto, esta sería la ruta para este ejemplo:

```
open("datos/nombres.txt")
```

Atención!! estamos escribiendo datos/ al principio (el nombre de la carpeta seguido de un /) y luego nombres.txt (el nombre del archivo con su extensión).

Segundo parámetro: mode (modo)

El segundo parámetro de la función open() es mode (modo), una cadena de caracteres conformada por un solo carácter. Ese único carácter básicamente le dice a Python lo que planeas hacer con el archivo en tu programa.

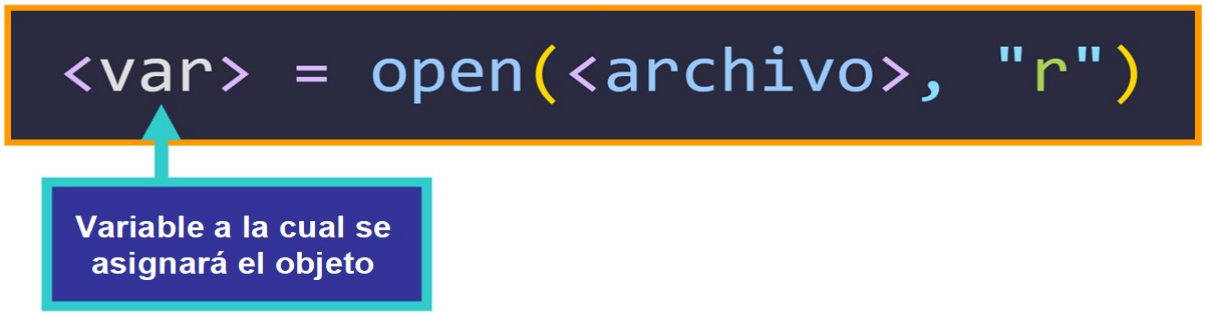
Para usar los modos de texto o binario, debes añadir estos caracteres al modo principal. Por ejemplo: "wb" significa "escribir en modo binario".

Dato: los modos asignados por defecto son read ("r") (leer) y text ("t") (texto), lo cual significa "abrir para leer texto" ("rt"), así que no necesitas especificarlos en open() si deseas usarlos porque se asignan automáticamente. Puedes simplemente escribir open(<archivo>).

¿Por qué modos?
En realidad, tiene sentido que Python solo otorgue ciertos permisos en base a lo que planeas hacer con el archivo

Cómo leer un archivo

Ahora que sabes más sobre los argumentos que recibe la función **`open()`**, veamos cómo puedes abrir un archivo y guardalo en una variable para usarlo en tu programa.
Esta es la sintaxis básica:



Estamos asignando el valor retornado a una variable. Por ejemplo:

```
archivo_nombres = open("data/nombres.txt", "r")
```

Sé que te debes estar preguntando: ¿qué tipo de valor retorna **`open()`**?
La respuesta es.... un objeto archivo.
Los objetos archivo tienen atributos, tales como:

- **name**: el nombre del archivo.
- **closed**: `True` si el archivo está cerrado. `False` si está abierto.

- **mode**: el modo usado para abrir el archivo.

Por ejemplo:
```
f = open("datos/nombres.txt", "a")
print(f.mode) # Resultado: "a"
```
Ahora veamos cómo podemos acceder al contenido de un archivo a través de un objeto archivo.

Métodos para leer un archivo

Para poder trabajar con objetos archivo, necesitamos tener una forma de "interactuar" con ellos en nuestro programa y eso es exactamente lo que hacen los métodos. Veamos algunos de ellos.

Read()

El primer método que debes aprender es **`read()`**, el cual retorna todo el contenido del archivo como una cadena de caracteres.

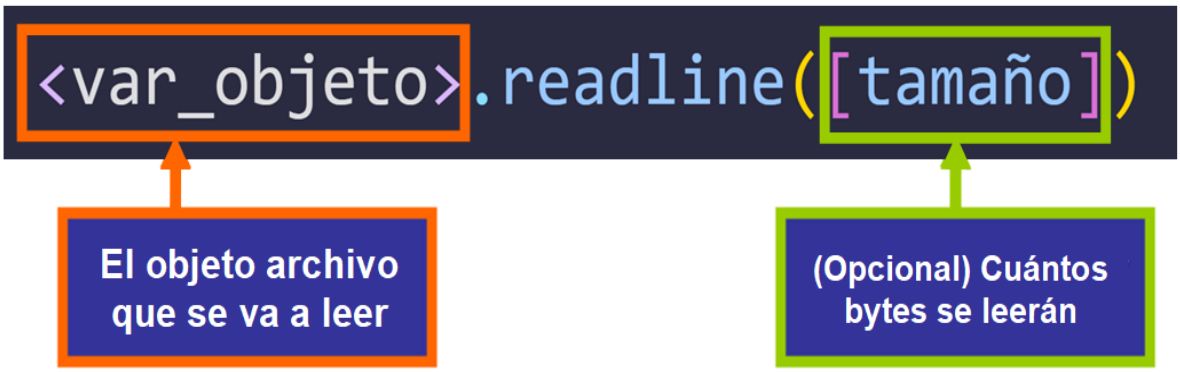
Aquí tenemos un ejemplo:
```
f = open("data/nombres.txt")
print(f.read())
```
El resultado es:
```
Nora
Gino
Timmy
William
```
Puedes usar la función `type()` para confirmar que el valor retornado por `f.read()` es una cadena de caracteres:
```
print(type(f.read()))

# Output
<class 'str'>
```
En este caso, se mostró todo el archivo porque no especificamos un número máximo de bytes, pero también podemos hacerlo:
Aquí tenemos un ejemplo:
```
f = open("data/nombres.txt")
```

```
print(f.read(3))
```
El valor retornado se limitará a este número de caracteres:
```
Nor
```
Importante: debes **cerrar** el archivo luego de que la tarea ha sido completada para liberar los recursos asociados al archivo. Para hacerlo, debes llamar al método **`close()`** de esta forma:

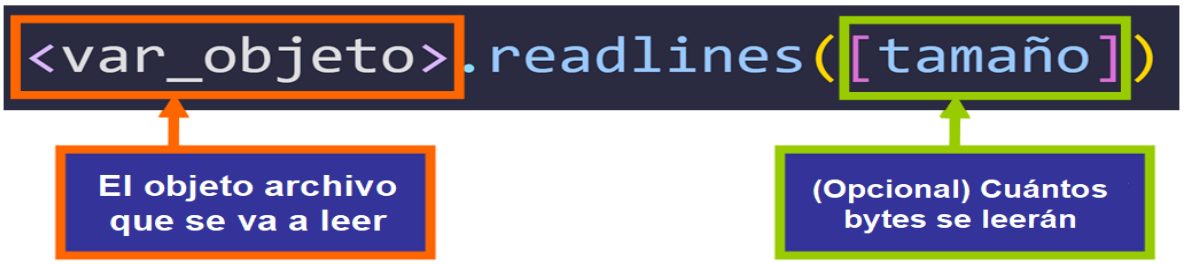
Readline() vs. Readlines()

Puedes leer un archivo línea por línea con estos dos métodos. Son ligeramente diferentes, así que veámoslos en detalle.
`readline()` lee una línea del archivo. Mantiene el carácter de salto de línea (`\n`) al final de la cadena de caracteres.

Dato: Opcionalmente, puedes pasar el tamaño (size), el número máximo de caracteres que deseas incluir en el resultado.



Por ejemplo:
```
f = open("data/nombres.txt")
print(f.readline())
f.close()
```
El resultado es:
```
Nora
```
Esta es la primera línea del archivo.
En cambio, **`readlines()`** retorna una lista que contiene todas las líneas del archivo como elementos individuales de la lista (cadenas de caracteres).
Esta es la sintaxis:



Por ejemplo:
```
f = open("data/nombres.txt")
print(f.readlines())
f.close()
```
El resultado es:
```
['Nora\n', 'Gino\n', 'Timmy\n', 'William']
```
Nota que cada cadena de caracteres termina con un carácter de salto de línea `\n`, excepto la última.

Dato: Puedes obtener la misma lista con `list(f)`.

Puedes trabajar con esta lista en tu programa asignándola a una variable o usando un ciclo:
```
f = open("data/nombres.txt")

for line in f.readlines():
    # Hacer algo con cada línea del archivo.

f.close()
```
También podemos iterar sobre `f` directamente (sobre el objeto archivo) en un ciclo:
```
f = open("data/nombres.txt", "r")

for line in f:
    # Hacer algo con cada línea del archivo.

f.close()
```

Cómo crear un archivo

Si necesitas crear un archivo de forma dinámica usando Python, puedes hacerlo con el modo `"x"`. Veamos cómo. Esta es la sintaxis básica:

Puedes escribir el nombre del archivo para crearlo en el directorio en el que estás actualmente o especificar una ruta para crear el archivo en una carpeta diferente.

Si ejecuto esta línea de código:
```
f = open("archivo_nuevo.txt", "x")
```
Con este modo, puedes crear un archivo y luego añadir contenido de forma dinámica

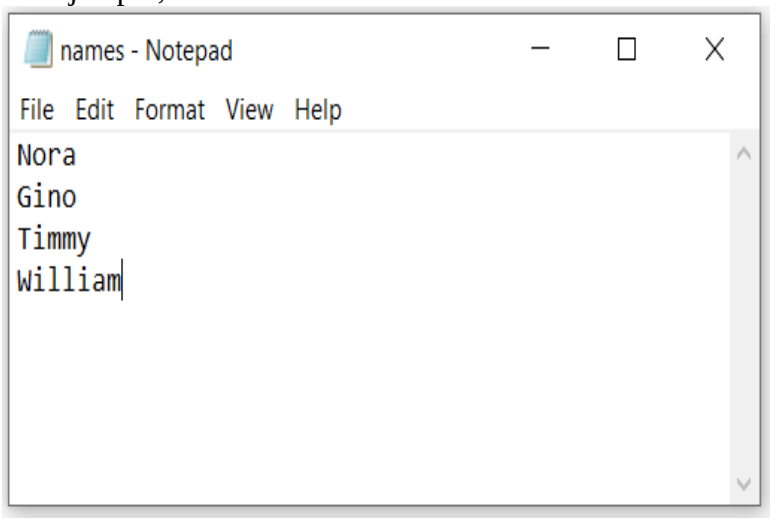
Dato: El archivo inicialmente estará vacío.
Algo curioso es que si intentas ejecutar esta línea de código nuevamente y ya existe un archivo con ese mismo nombre, verás un error:
```
Traceback (most recent call last):
  File "<path>", line 8, in <module>
    f = open("archivo_nuevo.txt", "x")
FileExistsError: [Errno 17] File exists: 'archivo_nuevo.txt'
```

Cómo modificar un archivo

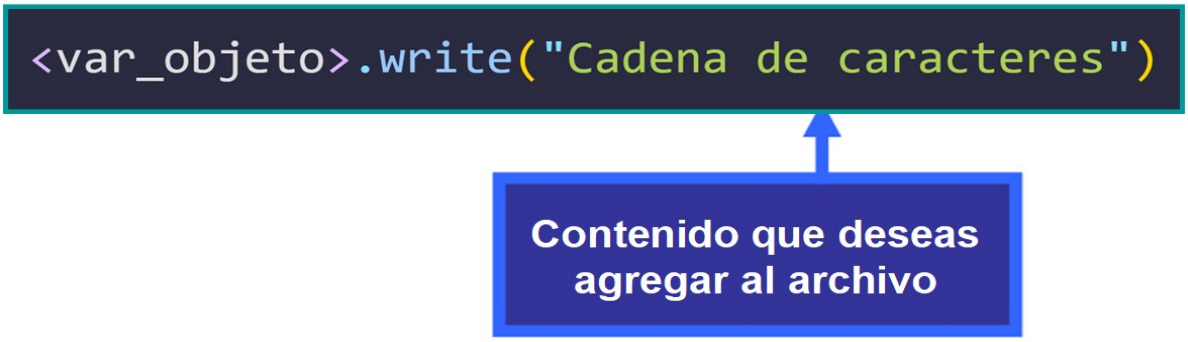
Para modificar (cambiar el contenido) de un archivo, debes usar el método **`write()`**. Existen dos alternativas (append o write) en base al modo que escojas para abrir el archivo. Veámos estas alternativas en detalle.

Append

"Append" significa agregar algo al final de una estructura o valor. El modo `"a"` (append) te permite abrir un archivo para agregar contenido al final del contenido existente.
Por ejemplo, si tenemos este archivo:



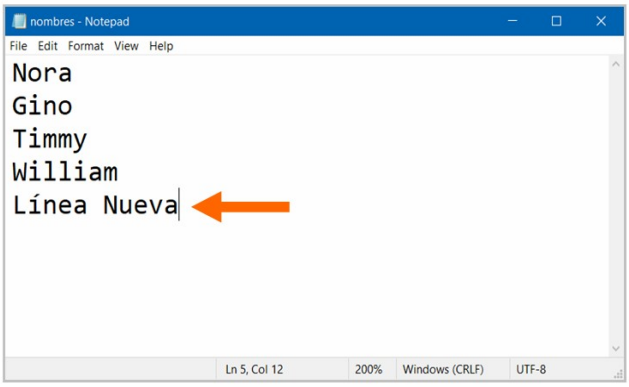
Y queremos agregarle una línea nueva, podemos abrir el archivo usando el modo **`"a"`** (append) y luego llamar al método **`write()`** pasando el contenido que queremos agregar como argumento.
Esta es la sintaxis básica para llamar al método **`write()`**:



Aquí tenemos un ejemplo:

```
f = open("datos/nombres.txt", "a")
f.write("\nLínea Nueva")
f.close()
```

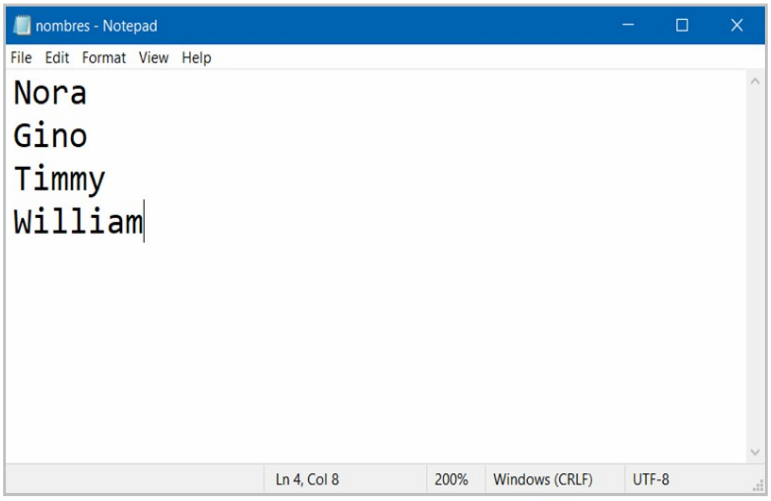
Dato: Nota que estoy agregando `\n` antes de la línea para indicar que mi intención es que la línea nueva se agregue como una línea separada y no como una continuación de la línea actual. Este es el archivo luego de ejecutar el programa:



Dato: Es posible que la línea nueva no aparezca en el archivo hasta que **`f.close()`** se ejecute.

Write

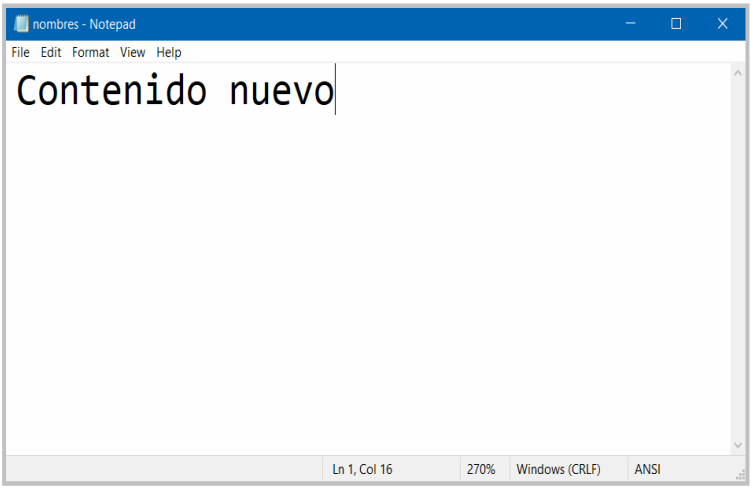
También puedes eliminar todo el contenido de un archivo y reemplazarlo completamente con contenido nuevo. Puedes hacerlo con el método **`write()`** si abres el archivo en el modo **`"w"`** (write).
Aquí tenemos este archivo de texto:



Si ejecuto este programa:

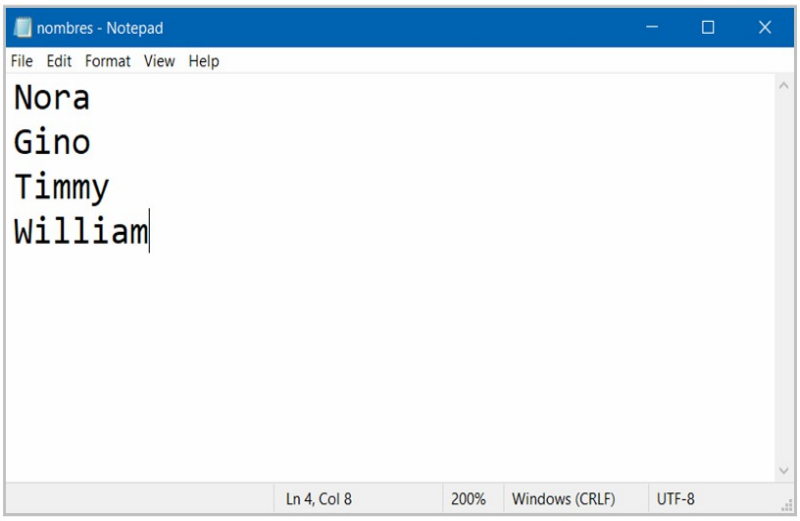
```
f = open("datos/nombres.txt", "w")
f.write("Contenido nuevo")
f.close()
```

Este es el resultado:



Como puedes ver, abrir el archivo con el modo **`"w"`** y luego llamar al método **`write()`** reemplaza el contenido del archivo.

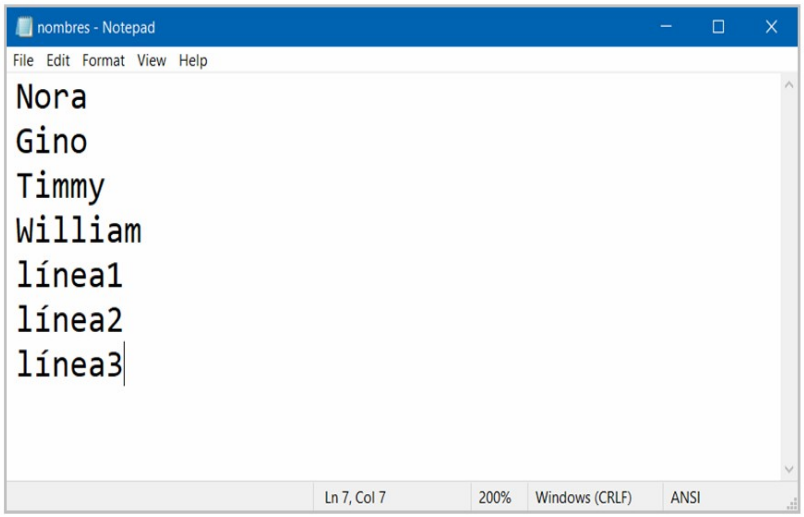
Dato: El método **`write()`** retorna el número de caracteres que fueron agregados al archivo.
Si deseas escribir varias líneas a la vez, puedes llamar al método **`writelines()`**, el cual toma una lista de cadenas de caracteres como argumentos. Cada cadena de caracteres representa una línea que se agregará al archivo.
Aquí tenemos un ejemplo. Este es el estado inicial del archivo:



Si ejecutamos este programa:

```
f = open("datos/nombres.txt", "a")
f.writelines(["\nlínea1", "\nlínea2", "\nlínea3"])
f.close()
```

Las líneas se agregan al final del archivo:



Abrir un archivo para varias operaciones

Ahora ya sabes cómo crear, leer y agregar contenido a un archivo, pero ¿cómo puedes realizar más de una operación con el mismo archivo en el mismo programa? Veamos qué ocurre si intentamos hacerlo con los modos que hemos aprendido hasta el momento.
Si abres un archivo en modo **`"r"`** (leer) y luego tratas de modificarlo:

```
f = open("datos/nombres.txt")
f.write("Contenido Nuevo") # Intentar escribir
```

```
f.close()
```

Verás este error:
```
Traceback (most recent call last):
  File "<path>", line 9, in <module>
    f.write("Contenido Nuevo")
io.UnsupportedOperation: not writable
```

De igual forma, si abres un archivo en modo **`"w"`** mode (escribir) y luego intentas leer su contenido:

```
f = open("datos/nombres.txt", "w")
print(f.readlines()) # Intentar leer
f.write("Contenido Nuevo")
f.close()
```

Verás el siguiente error:
```
Traceback (most recent call last):
  File "<path>", line 14, in <module>
    print(f.readlines())
io.UnsupportedOperation: not readable
```

Lo mismo ocurrirá con el modo **`"a"`** (append).

¿Cómo podemos solucionar este problema?

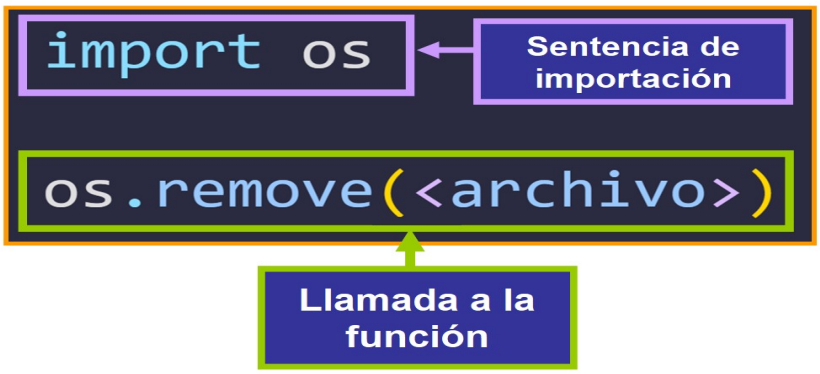
Para leer el archivo y realizar otras operaciones en el mismo programa, debemos agregar el símbolo **`"+"`** al modo, de esta forma:
```
f = open("datos/nombres.txt", "w+") # Leer + Escribir
f = open("datos/nombres.txt", "a+") # Leer + Append (agregar al final)
f = open("datos/nombres.txt", "r+") # Leer + Escribir
```

Muy útil. Esto es lo que normalmente necesitarás usar en tus programas, pero asegúrate de incluir solo los modos que necesitas para evitar problemas o errores (bugs).

Cómo eliminar archivos

Para eliminar un archivo usando Python, debes importar un módulo llamado **`os`** que contiene funciones para interactuar con tu sistema operativo.

Dato: Un módulo es un archivo de Python con variables, funciones y clases relacionadas. Específicamente, necesitas la función **`remove()`**. Esta función toma la ubicación o la ruta (path) del archivo como argumento y elimina el archivo automáticamente.



Veamos un ejemplo. Queremos eliminar el archivo llamado `archivo_demo.txt`.

Para hacerlo, escribimos el siguiente código:

```
import os
os.remove("archivo_demo.txt")
```

- La primera línea: `import os` se denomina una "sentencia de importación". Esta sentencia se escribe al inicio del archivo y te permite tener acceso a las funciones y otros elementos definidos en el módulo `os`.
- La segunda línea: `os.remove("archivo_demo.txt")` elimina el archivo especificado por el argumento.

Dato: Puedes usar una ruta (path) absoluta o relativa.