Programación II

Archivos

Lic. Federico Severino Guimpel
LCC - FCEIA - UNR

Archivos

En Python se puede abrir o crear un archivo usando la función `open`. Esta función retorna un objeto archivo. A partir de él vamos a poder leer o escribir.

Las acciones a realizar van a depender de la forma en la que el archivo se abra.

A continuación vamos a ver un par de ejemplos para ilustrar su uso.

Si escribimos:

```
>>> f = open("ejemplo.txt")
```

probablemente el resultado que vean sea:

```
>>> f = open("ejemplo.txt")
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    f = open("ejemplo.txt")
FileNotFoundError: [Errno 2] No such file or directory: 'ejemplo.txt'
```

Es decir, se generó una excepción. ¿Por qué sucede esto?

Esto pasa por el comportamiento de la función open.

```
>>> f = open("ejemplo.txt")
```

Al usarla de esta forma, se están diciendo dos cosas:

1. El archivo que se quiere abrir está en el directorio por defecto, es decir, el directorio de instalación.
2. Al no indicarse el modo de apertura del archivo, se asume que el mismo es para lectura.

Como consecuencia de que el archivo no existe en el directorio por defecto no se lo puede abrir para leer, dando lugar a la excepción. Para resolver esto habría que poner el path completo del archivo a leer como aparece posteriormente.

Se mencionó el modo de apertura de un archivo para explicar el por qué de la excepción anterior. Ahora bien, nos deberíamos preguntar:

a. ¿Qué tipos de modos de apertura hay?
b. ¿Cómo se indica el modo de apertura de un archivo?

A continuación, vamos a responder cada una de esas preguntas.

a. ¿Qué tipos de modos de apertura hay?

Hay varios. A continuación vamos a enumerar los más significativos, para el uso que le vamos a dar:

i. `'r'`: el archivo es abierto para lectura (modo por defecto).

ii. `'w'`: el archivo es abierto para escritura. Si ya existe, se va a sobreescribir. Si no existe, lo crea.

iii. `'r+'` ó `'w+'`: el archivo es abierto para lectura y escritura.

iv. `'a'`: el archivo es abierto para escribir al final del mismo. No pisa la información existente.

b. ¿Cómo se indica el modo de apertura de un archivo?

Es muy sencillo, al momento de usar la función open se pasa otro argumento que indica el modo. Por ejemplo:

```
>>> f = open("d:\ejemplo.txt", 'r+')
>>> f = open("d:\ejemplo.txt", 'a')
>>> f = open("d:\ejemplo.txt", 'w')
```

Archivos

Parecería que si abrimos un archivo para escritura no debería haber problemas. Sin embargo, nos puede pasar esto:

```
>>> f = open("ejemplo.txt", 'w')
Traceback (most recent call last):
  File "<pyshell#9>", line 1, in <module>
    f = open("ejemplo.txt", 'w')
PermissionError: [Errno 13] Permission denied: 'ejemplo.txt'
```

En este caso, la excepción nos indica que no se cuenta con permisos para crear un archivo en ese directorio. Esto sucede porque cada sistema operativo cuenta con un sistema de usuarios, archivos y permisos.
Para resolver esto habría que seleccionar otro lugar donde escribir el archivo y que se cuente con esos permisos.

Algo que debemos tener en cuenta es que luego de manipular el archivo debe cerrarse. Esto se hace usando la función `close`.

```
>>> f.close()
```

Si esta operación no se realiza, los cambios realizados podrían no aplicarse y, además podría no poder abrirse nuevamente (dependiendo del modo de apertura) hasta que se cierre.

Antes de ver con qué funciones se cuentan para poder leer y escribir en un archivo es preciso hacer un comentario previo para entender cómo funcionan estas operaciones.

Al momento de abrir un archivo se cuenta con lo que podríamos imaginar como una flecha que nos indica en qué lugar del archivo estamos. Si el archivo se abre con cualquier formato que no sea `'a'`, esa flecha hace referencia al inicio del archivo. Si es `'a'` hace referencia al final.

En la medida que vamos leyendo o escribiendo, esa flecha se desplaza hasta llegar al final del archivo, denominado EOF (End Of File). Más adelante vamos a ver ejemplos de este funcionamiento.

Archivos

Para escribir en un archivo se dispone de la función `write`. Esta función nos va a retornar un número; es la cantidad de caracteres que se escribieron en el archivo.

```
>>> f = open("d:\ejemplo.txt", 'w+')
>>> f.write("la primera linea\n")
17
>>> f.write("este archivo\n")
13
>>> f.write("tiene tres lineas\n")
18
>>> f.close()
```

El código anterior realiza los siguientes pasos:

1. Abre un archivo (si no existe lo crea y si existe lo sobreescribe)
2. Escribe tres líneas en él, una debajo de la otra. Es importante marcar que cada ‘\n’ genera un salto de línea. Como podemos ver, este comportamiento es parte de lo que habíamos adelantado en el slide 10.
3. Lo cierra.

```
>>> f = open("d:\ejemplo.txt", 'w+')
>>> f.write("la primera linea\n")
17
>>> f.write("este archivo\n")
13
>>> f.write("tiene tres lineas\n")
18
>>> f.close()
```

Archivos

Ahora que ya tenemos un archivo creado, con datos, vamos a leerlo. Disponemos de diferentes funciones para hacer esto. La primera es la función `readlines`.

```
>>> f = open("d:\ejemplo.txt", 'r')
>>> f.readlines()
['la primera linea\n', 'este archivo\n', 'tiene tres lineas\n']
```

Esta función lee el archivo completo y retorna una lista formada por las líneas encontradas.

Como podemos ver, al usar la función `readlines` y leer todo el archivo, nos quedamos posicionados al final del mismo, como habíamos comentado anteriormente. Por ese motivo, si queremos leer algo más vamos a obtener una lista vacía (en el caso de esta función).

```
>>> f = open("d:\ejemplo.txt", 'r')
>>> f.readlines()
['la primera linea\n', 'este archivo\n', 'tiene tres lineas\n']
>>> f.readlines()
[]
```

Archivos

Otras dos funciones que se pueden usar para leer son:

- la función `readline`, lee una única línea
- la función `read` puede:
  - tomar como argumento la cantidad de caracteres que quiere leer
  - no tomar argumentos y devolver el contenido completo del archivo.

```
>>> f.read()
'la primera linea\neste archivo\ntiene tres lineas\n'
```

Se puede ver el uso de estas funciones con un ejemplo:

```
>>> f.readline()
'la primera linea\n'
>>> f.read(5)
'este '
>>> f.read(10)
'archivo\nti'
>>> f.read()
'ene tres lineas\n'
```

Cuando estamos al final del archivo y usamos `readline` o `read` el resultado es el string vacío. Esto se puede usar como condición para saber que se llegó al final del archivo, como vamos a ver más adelante.

```
>>> f.read()
''
>>> f.readline()
''
```

A partir de esto, nos debería surgir una pregunta:

¿hay alguna forma de “retroceder” en el archivo?

La respuesta es que sí. Existe una función `seek` para desplazarse a la que tenemos que indicarle cuántos caracteres queremos movernos desde el inicio:

- en este caso, se posiciona al inicio del archivo

```
>>> f.seek(0)
```

- aquí se encuentra en el décimo caracter del archivo

```
>>> f.seek(10)
```

Archivos

Para finalizar vamos a ver cómo imprimir el contenido de un archivo. La primera opción es usar la función `readlines` e iterar sobre la lista que nos retorna. Recordemos que hay que estar posicionados en el inicio del archivo.

```
>>> for x in f.readlines():
        print(x)


la primera linea

este archivo

tiene tres lineas
```

Archivos

La otra opción es iterar, directamente sobre el propio archivo.

```
>>> for x in f:
        print(x)

la primera linea

este archivo

tiene tres lineas
```

También se debe estar posicionado al inicio del mismo.

Una última posibilidad es usar la función `readline` con un bucle `while`. La condición del bucle, como se mencionó anteriormente, nos permite iterar mientras no se llegue al final del archivo.

```
>>> linea = f.readline()
>>> while linea != '':
        print(linea)
        linea = f.readline()


la primera linea

este archivo

tiene tres lineas
```