MANEJO DE ARCHIVOS EN PYTHON

Existen dos formas básicas de acceder a un archivo, una es utilizarlo como un archivo de texto, que se procesa línea por línea; la otra es tratarlo como un archivo binario, que se procesa byte por byte.

En Python, para abrir un archivo usaremos la función **open(),** que recibe el nombre del archivo a abrir.

```
archivo = open("archivo.txt")
```

Esta función intentará abrir el archivo con el nombre indicado. Si tiene éxito, devolverá una variable que nos permitirá manipular el archivo de diversas maneras.

La operación más sencilla a realizar sobre un archivo es leer su contenido. Para procesarlo línea por línea, es posible hacerlo de la siguiente forma:

```
linea=archivo.readline()
while linea != '':
    # procesar linea
    linea=archivo.readline()
```

Esto funciona ya que cada archivo que se encuentre abierto tiene una posición asociada, que indica el último punto que fue leido.

Cada vez que se lee una línea, avanza esa posición. Es por ello que **readline()** devuelve cada vez una línea distinta y no siempre la misma.

La siguiente estructura es una forma equivalente a la vista en el ejemplo anterior.

```
for linea in archivo:
    # procesar línea
```

De esta manera, la variable línea irá almacenando distintas cadenas correspondientes a cada una de las líneas del archivo.

Es posible, además, obtener todas las líneas del archivo utilizando una sola llamada a función:

```
lineas = archivo.readlines()
```

En este caso, la variable líneas tendrá una lista de cadenas con todas las líneas del archivo.

Al terminar de trabajar con un archivo, es recomendable cerrarlo, por diversos motivos: en algunos sistemas los archivos sólo pueden ser abiertos de a un programa por la vez; en otros, lo que se haya escrito no se guardará realmente hasta no cerrar el archivo; o el limite de cantidad de archivos que puede manejar un programa puede ser bajo, etc.

Para cerrar un archivo simplemente se debe llamar a:

```
archivo.close()
```

Es importante tener en cuenta que cuando se utilizan funciones como **`archivo.readlines(),`** se está cargando en memoria el archivo completo. Siempre que una instrucción cargue un archivo completo en memoria se debe tener cuidado de utilizarla sólo con archivos pequeños, ya que de otro modo podria agotarse la memoria del computadora.

Por ejemplo, para mostrar todas las líneas de un archivo, precedidas por el número de línea, podemos hacerlo como en el siguiente código:

```
archivo = open("archivo.txt")
i = 1
for linea in archivo:
    linea = linea.rstrip("\n")
    print (i, linea)
    i+=1
archivo.close()
```

La llamada a rstrip es necesaria ya que cada línea que se lee del archivo contiene un fin de línea y con la llamada a **`rstrip("\n")`** se remueve.

Los archivos de texto son sencillos de manejar, pero existen por lo menos tres formas distintas de marcar un fin de línea.

En Unix tradicionalmente se usa el caracter **\n** (valor de ASCII 10, definido como nueva línea) para el fin de línea, mientras que en Macintosh el fin de línea se solía representar como un **\r** (valor ASCII 13, definido como retorno de carro) y en Windows se usan ambos caracteres **\r\n**.

Si bien esto es algo que hay que tener en cuenta en una diversidad de casos, en particular en Python por omisión se maneja cualquier tipo de fin de línea como si fuese un **\n**, salvo que se le pida lo contrario.

Otra opción para hacer exactamente lo mismo seria utilizar la función de Python enumerate(secuencia).

```
archivo = open("archivo.txt")
for i, linea in enumerate(archivo):
    linea = linea.rstrip("\n")
    print (i, línea)
archivo.close()
```

MODOS DE APERTURA DE UN ARCHIVO

La función open recibe un parámetro opcional para indicar el modo en que se abrirá el archivo. Los tres modos de apertura que se pueden especificar son:

Modo de sólo lectura **(r).** En este caso no es posible realizar modificaciones sobre el archivo, solamente leer su contenido.

Modo de sólo escritura **(w).** En este caso el archivo es truncado (vaciado) si existe, y se lo crea si no existe.

Modo sólo escritura posicionándose al final del archivo **(a)**. En este caso se crea el archivo, si no existe, pero en caso de que exista se posiciona al final, manteniendo el contenido original.

Por otro lado, en cualquiera de estos modos se puede agregar un **+** para pasar a un modo lectura-escritura. El comportamiento de **r+** y de **w+** no es el mismo, ya que en el primer caso se tiene el archivo completo, y en el segundo caso se trunca el archivo, perdiendo así los datos.

Si un archivo no existe y se lo intenta abrir en modo lectura, se generará un error; en cambio si se lo abre para escritura, Python se encargará de crear el archivo al momento de abrirlo, ya sea con **w**, **a**, **w+** o con **a+).**

En caso de que no se especifique el modo, los archivos serán abiertos en modo sólo lectura **(r).**

Si un archivo existente se abre en modo escritura **(w o w+),** todos los datos anteriores son borrados y reemplazados por lo que se escriba en él.

De la misma forma que para la lectura, existen dos formas distintas de escribir a un archivo.

Mediante cadenas:

```
archivo.write(cadena)
```

O mediante listas de cadenas:

```
archivo.writelines(lista_de_cadenas)
```

Así como la función read devuelve las líneas con los caracteres de fin de línea **(\n**), será necesario agregar los caracteres de fin de línea a las cadenas que se vayan a escribir en el archivo.

```
saludo = open("saludo.txt”,"w")
saludo.write("Hola Mundo\n")
saludo.close()
```

Abrir un archivo en modo agregar al final puede parece raro, pero es bastante útil.

Uno de sus usos es para escribir un archivo de bitácora (o archivo de log), que nos permita ver los distintos eventos que se fueron sucediendo, y así encontrar la secuencia de pasos (no siempre evidente) que hace nuestro programa.

Esta es una forma muy habitual de buscar problemas o hacer un seguimiento de los sucesos.

Para los administradores de sistemas es una herramienta esencial de trabajo.