TEXT CLASS REVIEW

TEMAS A TRATAR EN EL CUE

- ¿Por qué necesitamos manejar archivos?
- ¿Qué es un File Descriptor?
- Apertura de archivos
- Distintos modos de apertura: sólo lectura, sólo escritura, y lectura y escritura

¿POR QUÉ NECESITAMOS MANEJAR ARCHIVOS?

Un archivo es la información o datos que se recopilan en los distintos dispositivos de almacenamiento de un ordenador. Pueden ser música, video, texto, entre otros.

En la programación, cuando se generan las diversas aplicaciones para el desarrollo de servicios web, aplicaciones de escritorio, algoritmos de automatización, y proyectos de aprendizaje automático, se necesitan manejar archivos ya que permiten ahorrar tiempo y ser más prácticos; por ende, ser mucho más eficiente con el manejo de datos.

En general, cuando se trabaja con Python dividimos los archivos en dos categorías: texto y binarios. Mientras que los archivos de texto son texto simple, los archivos binarios contienen datos que solo pueden ser interpretados por un ordenador. Python ofrece formas simples de manipular ambos tipos.

¿QUÉ ES UN FILE DESCRIPTOR?

Un descriptor de archivo (FD por sus siglas en inglés), es un pequeño número entero no negativo que ayuda a identificar un archivo abierto dentro de un proceso, mientras se utilizan recursos de entrada / salida. En cierto modo, puede considerarse como una tabla de índice de archivos abiertos. Cuando hay operaciones de lectura, escritura o cierre de archivos, uno de los parámetros de entrada considerados es el descriptor de archivo. Estos forman un componente importante de la interfaz de programación de aplicaciones, y proporcionan una interfaz primitiva de bajo nivel para las operaciones de entrada o salida.

Desde la perspectiva de la programación de aplicaciones, los descriptores de archivos deben usarse si hay operaciones de entrada o salida en modos especiales, incluidas las entradas sin bloqueo. A diferencia de las secuencias que proporcionan funciones altas para el control, una interfaz de descriptor de archivo proporciona funciones simples para la transferencia de bloques de caracteres. Las operaciones de bajo nivel se pueden realizar directamente en el descriptor de archivo.

APERTURA DE ARCHIVOS

Existen dos formas básicas de acceder a un archivo. Una es utilizarlo como un archivo de texto, que procesaremos línea por línea; y la otra es tratarlo como un archivo binario, que procesaremos 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")
```

Se utiliza para abrir el archivo, y devolver un objeto de archivo equivalente. La función `open()` toma principalmente dos parámetros: `filename` y `mode`.

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 para realizar sobre un archivo es leer su contenido. Para procesarlo línea por línea, es posible hacerlo de la siguiente forma:

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

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

```
líneas = archivo.readlines()
```

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

DISTINTOS MODOS DE APERTURA

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.

MODO R EN LA APERTURA DE ARCHIVOS PYTHON

El modo r se utiliza cuando queremos abrir el archivo para su lectura. El puntero de archivo en este modo se coloca en el punto de inicio del archivo.

El modo r se puede utilizar en la función `open()` de la siguiente manera:

```
f1 = open("god.txt", "r")
```

MODO R+ EN LA APERTURA DE ARCHIVOS PYTHON

El modo r+ se utiliza para abrir un archivo tanto para lectura, como para escritura. Al igual que en el modo anterior, el puntero de archivo en este modo también se coloca en el punto de inicio del archivo.

El modo r+ se puede utilizar en la función `open()` de la siguiente manera:

```
f1 = open("god.txt", "r+")
```

Para abrir el archivo, escribir y leer en formato binario, podemos usar el modo rb+.

```
f1 = open("god.txt", "rb+")
```

MODO W EN LA APERTURA DE ARCHIVOS PYTHON

El modo w se utiliza para abrir un archivo con el único propósito de escribir. Si el archivo ya existe, trunca el archivo a una longitud cero y, de lo contrario, crea uno nuevo. El puntero de archivo en este modo se coloca en el punto de inicio del archivo.

El modo w se puede utilizar en la función `open()` de la siguiente manera:

```
f1 = open("god.txt", "w")
```

MODO W+ EN LA APERTURA DE ARCHIVOS PYTHON

El modo w+ abre el archivo para lectura y escritura. Si el archivo ya existe, se trunca y, de lo contrario, se crea uno nuevo. El puntero de archivo en este modo se coloca en el punto de inicio del archivo.

El modo w+ se puede utilizar en la función `open()` de la siguiente manera.

```
f1 = open("god.txt", "w+")
```

Para abrir el archivo en formato binario, podemos usar el modo wb+.

```
f1 = open("god.txt", "wb+")
```

MODO A EN LA APERTURA DE ARCHIVOS PYTHON

El modo a abre el archivo con el propósito de agregarlo. El puntero de archivo en este modo se coloca al final del archivo si ya existe en el sistema. Si el archivo no existe, se crea para escritura.

El modo a se puede utilizar en la función `open()` de la siguiente manera.

```
f1 = open("god.txt", "a")
```

MODO A+ EN LA APERTURA DE ARCHIVOS PYTHON

El modo a+ abre el archivo para leerlo y agregarlo. El puntero de archivo en este modo se coloca al final del archivo si ya existe en el sistema. El archivo se abre en el modo anexar. Si el archivo no existe, se crea para escritura.

El modo a+ se puede utilizar en la función `open()` de la siguiente manera.

```
f1 = open("god.txt", "a+")
```

Para abrir el archivo en modo binario, podemos agregar el modo ab+.

```
f1 = open("god.txt", "ab+")
```

MODO X EN LA APERTURA DE ARCHIVOS PYTHON

Este modo está disponible para las versiones Python 3, y superiores. Abre el archivo para creación exclusiva, fallando si el archivo con ese nombre ya existe. Cuando se especifica creación exclusiva, significa que este modo no creará uno nuevo si el archivo con el nombre especificado ya existe. En el modo x, el archivo solo se puede escribir, pero en el modo x+, el archivo se abre como legible y escribible.

Este modo es significativo, y es útil cuando no queremos truncar accidentalmente un archivo ya existente con los modos a o w.

El modo x se puede utilizar en la función `open()` de la siguiente manera.

```
f1 = open("god.txt", "x")
```