**INTRODUCCION**

Los archivos son una forma de almacenar información, el sistema operativo se encarga de gestionarlos mediante el sistema de archivos, hay dos tipos principales:

Directorios (Carpetas)

Archivos normales, que pueden ser de dos tipos:  
 Texto (para almacenar texto plano, requiere codificación de texto, por ejemplo, utf-8)  
 Binarios (no es texto plano, cualquiera que no sea de texto plano es binario y tienen muchos usos, como por ejemplo música, videos, juegos, etc.)

**RUTAS DE ARCHIVO**

En el sistema todos los archivos se almacenan dentro de alguna ruta, esta depende de una estructura de árbol, donde cada directorio representa un nodo.

En sistemas unix, el directorio raíz es /

En sistemas Windows, cada dispositivo de almacenamiento se identifica con una letra, como c, y tiene su propio árbol de directorios

En sistemas tipo unix cada subdirectorio de la ruta se separa con una /

En Windows se puede usar / o la / invertida

En la parte final de la ruta puede venir el nombre de un archivo normal

**DOS TIPOS DE RUTAS**

Rutas absolutas: si la ruta se da desde la raíz

Rutas relativas: no empieza con diagonal, y es relativa al directorio de trabajo actual  
Hay rutas relativas especiales: . (referencia a la ruta actual), .. (Referencia al directorio padre)

**EN PYTHON**

En Python hay una API muy simple y útil para procesar archivos.

Se verán las siguientes operaciones con archivos:

Abrir un archivo existente para lectura, volcar contenido completo de un archivo de lectura, recorrer línea por línea de un archivo de lectura, cerrar archivos, abrir archivos existentes o nuevo para escritura, abrir un archivo para escribir al final y volcar buffer de escritura

**OPERACIONES**

**Abrir archivos:** se usa la función “open”, se le pasan dos parámetros: ruta del archivo y modo de uso del archivo (si es texto, binario, lectura, escritura o Append). La función regresa un objeto especial que representa el archivo y permite manipularlo

Archivo = open(‘/tmp/nuevo.txt’,’tr’ #t quiere decir modo texto, r quiere decir modo lectura

#por defecto, open usa modo tr, por lo que no es necesario ponerlo, aquí una línea equivalente  
archivo = open(‘/tmp/nuevo.txt’)

Una vez abierto el archivo en modo lectura se puede leer todo el contenido y pasarse a cadena mediante el método read del objeto archivo.

Contenido = archivo.read()  
archivo.close() #para cerrar

Solo se debe hacer si el archivo no es demasiado grande ,de otra forma es ineficiente. Se vera a continuación una forma alterna de leer el contenido del archivo sin cargarlo por completo a la RAM

**Recorrer línea por línea de archivo** : mediante for, es posible recorrer un archivo de texto línea por línea, muy común y conveniente en Python, donde en vez de cargarse a la RAM, se procesa en la línea de texto

For línea in open(‘nuevo.txt):  
print(Linea)

**Cerrar archivos:** cuando se abre un archivo se asocia un recurso al proceso, que abre el archivo, este proceso se llama “”descriptor de archivo” cada proceso tiene un numero finito de descriptores, si un proceso se excede de este numero, al querer abrir un nuevo archivo habrá una excepción, es buena practica, para evitar este problema, cerrar el archivo siempre que se termine de usarlo, esto se logra con el método “close”  
ar = open(‘nuevo.txt’)  
contenido = ar.read()  
ar.close()

Tambien se puede lograr de manera indirecta usando for, con una sentencia especial llamada “with”.

with permite crear un contexto especial de ejecución, en el caso de archivos, permite que se cierren de forma robusta al terminar el bloque with

En general se recomienda usar with siempre que se pueda

with open('nuevo.txt') as archivo:

contenido = archivo.read()

**Abrir un archivo existente o nuevo para escritura:** Hay que tener cuidado con esta operación

Si la ruta de archivo que se proporciona a open no existe, entonces se crea un archivo nuevo (siempre y cuando los subdirectorios de la ruta si existan)

Si la ruta es de un archivo existente, éste será sobre escrito, esto es, se perderá su contenido original

archivo = open('/tmp/nuevo.txt', 'tw')

# equivalente, la t está por defecto:

archivo = open('/tmp/nuevo.txt', 'w')

# para escribir, tener cuidado si se quieren saltos de línea:

archivo.write('cadena a agregar\n')

archivo.write('otra línea\n')

arhivo.close()

**Abrir un archivo existente para escribir al final**: Como ya se mencionó, al utilizar la configuración w de open, se sobrescribe en el archivo si es que existe. Para evitar esto, se puede usar la configuración a para activar el modo Append, en este modo no se sobrescribe el archivo, sino que se escribe al final de este.

With open(ruta, ‘a’) as archivo:  
 archivo.write(‘nueva linea’) #se agrega al final

**Volcar buffer de escritura:**Para volcar el contenido a disco se puede hacer como close o con el método flush

**PROCESAMIENTO DE ARCHIVOS BINARIOS**

El procesamiento de archivos binarios es mas complejo que el de texto, es necesario primero familiarizarse con cadenas binarias. Las cadenas binarias son similares a las cadenas de texto e incluso comparten un API similar. Python facilita mucho el manejo de binario gracias a la nocion de cadenas binarias (en muchos otros lenguajes suele ser mas complejo)

Una cadena binaria contiene una serie de bytes, es la unidad mínima de información procesable, y en Python una cadena binaria literal empieza con la letra b

Si se quiere ver cuantos bytes tiene una cadena, se puede usar la función “list”

Se debe establecer el modo r, w, a para binario