**PROCESAMIENTO DE ARCHIVOS**

handles = un tipo de puntero inteligente

streams = una especie de canal

la libertad sobre los streams depende de las caracteristicas de permisos del fichero

la manera en la que trabaja el stream depende del modo en el que abrimos el stream

los streams se los puede analogar a una grabadora

OPERACIONES CON EL STREAM

lectura: se recuperan los datos del fichero y se lo guarda en un espacio de memoria administrada por el programa

escritura : porsiones de datos de la memoria administrada por el programa se transfieren al fichero



MODOS PARA ABRIR EL STREAM

modo lectura : solo se permite la operación de lectura

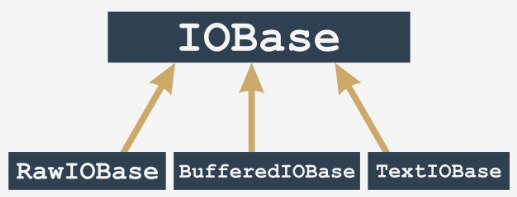
en caso de hacer una operación distinta salta la excepcion

UnsupportedOperation :

modo escritura : solo se permite la operación de escritura

en caso de hacer otra operación salta la anterior excepcion

modo actualizar : este modo permite tanto lectura como escritura



estas son las clases que se encargan de abrir los streams

CONTENIDO DE LOS STREAMS

puede ser de tipo texto : esta contenido por lineas de caracteries tipograficos

estos datos se leen carácter por carácter, o linea por linea

el final de linea en python se ldica con LF (\n)

puede ser binario : esta lleno de bytes

pueden ser ejecutables, imagenes ,audio, video, o una base de datos

ya que no tiene lineas, se leen byte a byte, o bloque a bloque

PROBLEMA DE PORTABILIDAD

en sustemas windows el fin de linea se hace con \r\n y en sistemas linux es con \n

si se quiere leer estos archivos en otro OS este se vovlera inutil

SOLUCION

en sistemas linux no pasa naa, pero en windows se produce una traduccion de caracteres de nueva linea

cambia \r\n por \n y viceversa en el caso necesario

**ABRIR STREAMS**

stream = opne(file , mode = ‘r’ , encoding = None)

devuelve **un objeto stream** en caso de que el stream se abra correctamente de lo contrario manda una excepcion

el primer argumento, es el nombre del fichero

el segundo argumento es el modo de apertura del stream

‘r’ : es modo de lectura

el fichero **tiene que existir**

‘w’ : es modo de escritura

si el fichero no existe es creara en este modo

**si el fichero si existe lo BORRA**

‘a’ : es el modo adjuntar

si no existe el fichero lo creara

si el fichero existe el “cabezal de la grabadora” ira el final del fichero

el contenido anterior queda intacto

‘r+’ : es el modo leer y actualizar

**el fichero debe existir y tiene que se escribible**

permite la lectura y excritura en el stream

‘w+’ : es el modo escribir y actualizar

**si no existe el fichero lo creara**

el contenido anterior queda intacto

permite lectura y escritura en el stream

al final de cada uno de los modos poner ‘b’ o ‘t’ para especificar si es en modo binario o texto

cuando estamos en modo texto (‘t’) este archivo puede **tener SOLO texto**

los valores por defecto es decir si no se le pasa argumentos es

apertura en modo lectura de modo texto

la codificacion depende de la plataforma utilizada

**ver ejemplo /programas\_3/codigo1.py**

STREAMS PREDEFINIDOS

from sys import stdin , stdout , stderr

stdin : es un stream que se asocia con el teclado

la funcion input() lee datos de este stream

stdout : normalmente asociado con la pantalla

la funcion print() envia datos al stream stdout

stderr : se asocia normalmente con la pantalla

este es el stream donde se envian los errores que el programa en ejecucion a encontrado

MANIPULAR LOS STREAMS

print(stream.read())

la funcion read() sinningun argumento lee todo el contenido del archivo y lo devuelve como una cadena

print(stream.read(1))

lee solo 1 carácter del archivo y **avanza el “cabezal de lectura del stream”** hasta despues del carácter leido

podemos analogar este cabezal como un puntero

print(stream,readline())

lee toda una linea del archivo

mueve el puntero hasta despues de la linea leida

print(stream.readlines(numero\_lineas))

esta funcion devuelve una lista con las lineas en forma de cadena del archivo en cada indice de la lista

si se le pasa argumento especifica el numero de lineas que leera

lleva el puntero hasta el final de su lectura

si no se le pasa argumentos leera todas las filas, y delvolvera una lasta con cadenas en cada indice

**ver ejemplo en /programas\_3/codigo2.py**

OPEN() COMO ITERABLE

la funcion open() puede trabajar como un iterador que se lo puede meter dentro un iterador

dentro un for por ejemplo

open como un iterable devulve linea por linea del archivo

**ESCRIBIR EN ARCHIVOS**

tenemos que abrir previamente el archivo

se tiene que abrir el archivo en modo lectura

open(‘nombre\_archivo’ , ‘wt’)

al momento de abrir tenemos que especificar la ruta donde queremos que cree el archivo

si abrimos el stream de la anterior manera **creara el archivo especificado**

**en caso de ya existir el archivo borrara lo que habia y sobre escribira**

cadena = ‘hola soy un programador\n’

stream.write(cadena)

para agregar salto de linea tenemos que especificar en la cadena que andemos con \n

**ARCHIVOS BINARIOS**

los archivos binarios son solo datos amorfos de binarios

es decir no estan ordenados en filas o caracteres, son solo bits

LEER ARCHIVOS BINARIOS

para leer un binario tenemos que crear un espacio de memoria binario

data = bytearray(n\_bytes)

es un **tipo de dato MUTABLE**

es un iterable

creamos un array con n\_bytes especificado todos llenos de ceros (0)

bytearray son mutables,se accede a el usando indexacion normal

**solo para enteros entre 0 – 255**

de lo contrario dara error

cada valor se lo puede tratar y guardar como un entero

hex(numero\_entero)

esta funcion devuelve la forma hexadecimal del numere entero pasado

para escribir leer en un archivo binario utilizamos un bytearray() del tamano que querramos

aquí se guardaran los datos del archivo para manipularlos en el programa

abrimos el stream en modo lectura y en binario, especificando la ruta y el nombre del archivo que queremos leer

stream = open(‘/ruta/file.bin’ , ‘rb’)

y le pasamos el bytearray donde guardara lo que lea del archivos

stream.readinto(bytearray)

en caso de que el bytearray sea mas pequeño que el archivo (en bytes) leera solo el tamano del bytearray

en caso de que el bytearray sea mas grande que el archivo (en bytes) leera todo el archivo dejando espacios intactos que lleguen a ser usados en el bytearray

en ambos casos el puntero del stream avanzara hasta donde llegue a leer del archivos

otra forma de leer de un archivo binario es con el metodo read()

datos = stream.read() #aquitratara de leer todo el archivo

entrega un objeto tipo ‘byte’

es un iterable

sus valores se los puede tratar como enteros

**es inmutable**

**solo utilizar este metodo cuando sepamos que el archivo es pequeño o que la memoria nos alcanzara**

datos = stream.read(10)

es lo mismo que el anterior ejemplo pero este limita los bytes leidos al valor de su argumetno

ESCRIBIR EN ARCHIVOS BINARIOS

primero tenemos que abrir el stream en modo de escritura y en binario

stream = open(‘/ruta/file.bin’ , ‘wb’)

ahora podemos pasarle los datos que queremos escribir directamente o a travez de un bytearray

stream.write(datos\_binarios)

datos = bytearray(10) #es un bytearray de 10 bytes

stream.write(datos)

**ver ejemplo de escribir y leer de un archivo binario en**

/programas\_3/codigo2\_1.py/

**despues de escribir o leer en un archivo binario o de texto SIEMPRE tenemos que cerrar el stream**

stream.close()

**ver ejemplo de un copiador de archivos binarios en:**

**/**programas\_3/codigo2\_2.py