	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos II</b>	Código:	MADO-20
		Versión:	01
		Página	127/183
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

# Guía Práctica de Estudio 10


## Archivos

Elaborado por:

M.I. Elba Karen Sáenz García

Revisión:

Ing. Laura Sandoval Montaña

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos II</b>	Código:	MADO-20
		Versión:	01
		Página	128/183
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

## Guía Práctica 10

### Estructura de datos y Algoritmos II

### Archivos

**Objetivo:** El estudiante conocerá e identificará aspectos sobre los archivos, como las operaciones, el tipo de acceso y organización lógica.

#### Actividades

Trabajar con operaciones, tipo de acceso y organización lógica de TDA archivo en algún lenguaje de programación.

#### Antecedentes

- Análisis previo del concepto de archivo, operaciones y organización visto en clase teórica.
- Manejo de listas, estructuras de control, funciones en Python 3.
- Conocimientos básicos de la programación orientada a objetos.

#### Introducción


El sistema operativo hace una abstracción de las propiedades físicas de los dispositivos de almacenamiento secundario para definir una unidad lógica de almacenamiento, el archivo.

Un archivo es una colección de información relacionada con un nombre que se guarda en memoria secundaria y comúnmente representan programas y datos. En general es una secuencia de bits, bytes, líneas o registros cuyo significado es definido por el creador y usuario del archivo. Se puede visualizar como espacios de direcciones lógicas contiguas.

#### Atributos de un archivo

Además del nombre un archivo tiene otros atributos que varían de un sistema operativo a otro, en general son:

- Nombre: Nombre simbólico visible al usuario.
- Tipo: Esta información es necesaria cuando se pueden trabajar diferentes tipos.
- Ubicación.: Es un apuntador a un dispositivo y a la ubicación del archivo en el dispositivo

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos II</b>	Código:	MADO-20
		Versión:	01
		Página	129/183
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

- Tamaño. Se tiene el tamaño actual (en bytes, palabras o bloques) y posiblemente el tamaño máximo permitido.
- Protección. Información del control de acceso, que determina quién puede leer, escribir, ejecutar, etc., el archivo
- Hora, fecha e identificación del usuario. Estos datos pueden ser útiles para la protección de seguridad y control de uso.


### Operaciones sobre archivos

Un archivo es un tipo de dato abstracto (TDA) que se manipula por medio de una interfaz ya sea por el SO o por algún programa. Al ser un TDA se tienen operaciones que se realizan sobre él. Las principales son.

- Creación del archivo: Para crear un archivo primero, es necesario encontrar espacio para el mismo
- Escribir en un archivo: Para escribir en un archivo, se debe realizar una llamada al sistema especificando el nombre y la información que se escribirá. Con el nombre se localiza la ubicación así, el sistema mantendrá un apuntador de escritura a la ubicación en el archivo, donde va a tener lugar a la siguiente escritura y este debe actualizarse siempre que ocurra una escritura.
- Lectura del archivo: Para leer un archivo, se realiza una llamada al sistema especificando el nombre y dónde debe colocarse (dentro de la memoria) el siguiente bloque del archivo. Se explora el directorio para hallar la entrada asociada y el sistema necesita mantener un apuntador de lectura que haga referencia a la ubicación dentro del archivo en la que tendrá lugar la siguiente lectura. Una vez que la lectura se completó, se actualiza el apuntador de lectura
- Borrar un archivo: Para borrar un archivo, se explora el directorio en busca del archivo indicado. Una vez encontrada la entrada de directorio asociada, se libera todo el espacio del archivo y se borra la entrada del directorio.

Dado que en las operaciones descritas se realiza primero una búsqueda en el directorio para encontrar la entrada asociada con el archivo, muchos sistemas requieren que se realice una función de apertura (`open()`) antes de utilizar por primera vez el archivo, así cuando se solicita una operación sobre un archivo, se utiliza un índice (descriptor de archivo) en la llamada **tabla de archivos abiertos** que contiene información de todos los archivos abiertos, de manera que no se requiere de una búsqueda. Cuando el archivo deja de utilizarse será cerrado por un proceso y el S. O. eliminará la entrada correspondiente en la tabla de archivos abiertos.

Es importante mencionar que en la función de apertura de un archivo se toma el nombre del mismo y explora el directorio, copiando la entrada del directorio correspondiente en la tabla de archivos abiertos [1]. Esta función además de recibir el nombre del archivo también acepta información acerca del modo de acceso: creación, solo lectura, escritura, lectura-escritura, agregar etc.

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos II</b>	Código:	MADO-20
		Versión:	01
		Página	130/183
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

## Tipos de Archivos

Existen archivos de diferentes tipos: cada uno podría ser un documento de texto, un binario ejecutable, un archivo de audio o video, etc. Una de las estrategias principales para que el sistema operativo reconozca el tipo de un archivo es la extensión. Cada nombre de archivo se divide en dos porciones, empleando como elemento separador al punto: el nombre del archivo y su extensión. hola.txt, programa.exe.

## Estructura interna de los archivos:

Los sistemas de disco normalmente tienen un tamaño de bloque bien definido y determinado por el tamaño de un sector. Todas las operaciones de entrada y salida en disco se realizan en unidades de un bloque (registro físico) y todos los bloques tienen un mismo tamaño. Es poco probable que el tamaño del registro físico corresponda exactamente a la longitud del registro lógico deseado el cual puede variar de longitud. Para resolver este problema se utiliza el llamado empaquetamiento de registros lógicos en bloques físicos [1].

Por ejemplo, el S.O. UNIX define todos los archivos como simples flujos de bytes. Cada byte es direccionable de manera individual, a partir de un desplazamiento con respecto al principio (o al final) del archivo. En este caso el tamaño de cada registro lógico es un byte. El sistema empaqueta y desempaqueta automáticamente los bytes en los bloques físicos del disco (por ejemplo 512 bytes por bloque) según sea necesario [1].


El tamaño del registro lógico, el tamaño del bloque físico y la técnica de empaquetamiento determinan cuántos registros lógicos se almacenan en cada bloque físico.

## Métodos de Acceso

Los archivos almacenan información y cuando se requiere utilizarla es necesario tener acceso a ella y leerla en la memoria de la computadora.

El método más simple de acceso a la información es el **secuencial** donde la información se procesa en orden, un registro después de otro. Una operación de lectura lee la siguiente porción del archivo e incrementa automáticamente un apuntador de archivo, que controla la ubicación de E/S. De forma similar, la operación de escritura añade información al final del archivo y hace que el apuntador avance hasta el final de los datos recién escritos (el nuevo final del archivo). Los archivos pueden reiniciarse para situar el apuntador al principio de los mismos, y en algunos sistemas, puede que el programa sea capaz de saltar hacia adelante o hacia atrás  $n$  registros, para algún cierto valor  $n$ .

Otro método es el acceso **directo, relativo u aleatorio** surge con la llegada de los dispositivos de acceso directo como los discos magnéticos; en este método de acceso el archivo se considera como un conjunto de registros, cada uno de los cuales puede ser un byte. Se puede acceder al mismo desordenadamente moviendo el

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos II</b>	Código:	MADO-20
		Versión:	01
		Página	131/183
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

apuntador de acceso al archivo a uno u otro registro. Esta forma de acceso se basa en un modelo de archivo almacenado en disco, ya que se asume que el dispositivo se puede mover aleatoriamente entre los distintos bloques que componen el archivo.

### Organización.

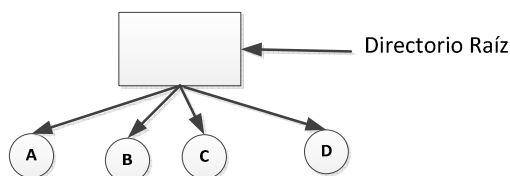
Hay diferentes maneras en las que puede ser organizada la información de los archivos, así como diferentes formas para tener acceso a dicha información.

### Directorios

Algunos sistemas almacenan millones de archivos en terabytes de disco. Para gestionar todos esos datos, se necesitan que se organicen y esa organización implica el uso de directorios.

La forma más simple de un sistema de directorios es tener un directorio que contenga todos los archivos. [2].

En la figura 10.1 se muestra un ejemplo de un sistema con un directorio que contiene cuatro archivos. A menudo se utiliza en dispositivos incrustados simples como teléfonos, cámaras digitales y algunos reproductores de música portátiles.




**Figura 10.1**

Tener un solo nivel es adecuado para aplicaciones dedicadas simples, pero para los usuarios modernos con miles de archivos, sería imposible encontrar algo si todos los archivos estuvieran en un solo directorio.

Lo que se necesita es una jerarquía. Cada directorio puede contener un número arbitrario de archivos, y también puede contener otros directorios (subdirectorios). Los otros directorios pueden contener todavía más archivos y directorios, y así sucesivamente, construyéndose una estructura en árbol en la que un directorio raíz puede contener cualquier número de niveles de otros directorios y archivos. Un ejemplo de este esquema se muestra en la figura 10.2. donde cada uno de los directorios A, B y C contenidos en el directorio raíz pertenecen a un usuario distinto, dos de los cuales han creado subdirectorios para proyectos en los que están trabajando [2].

El uso de directorios hace más fácil organizar los archivos de una manera lógica.

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos II</b>	Código:	MADO-20
		Versión:	01
		Página	132/183
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

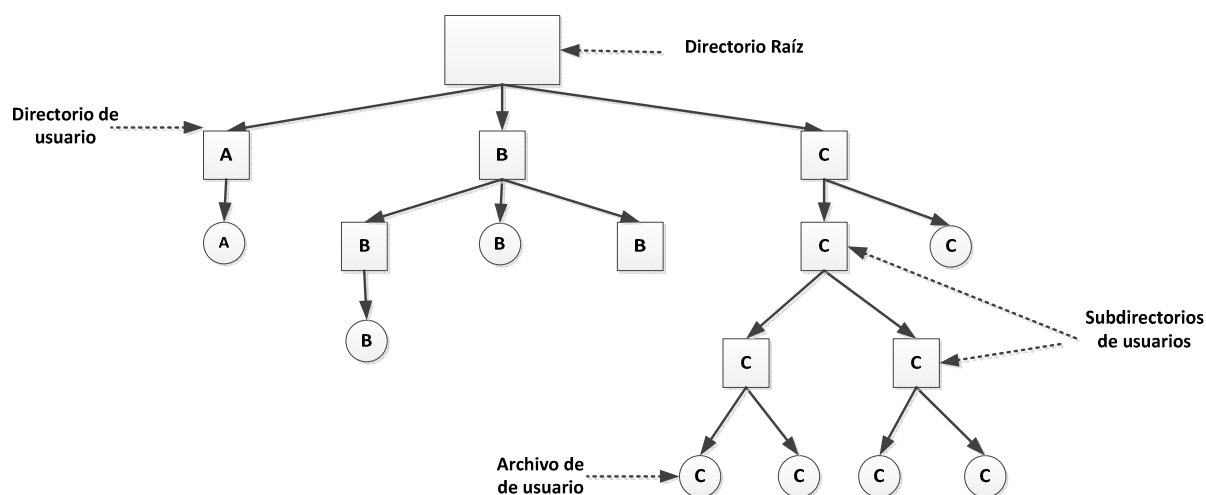


Figura 10.2

Cuando el sistema de archivos está organizado como un árbol de directorios, se necesita cierta forma de especificar los nombres de los archivos. Por lo general se utilizan dos métodos distintos. En el primer método, cada archivo recibe un **nombre de ruta absoluto** que consiste en la ruta desde el directorio raíz al archivo. Por ejemplo, en las siguientes rutas el directorio raíz contiene un subdirectorio llamado *usr* que a su vez contiene un subdirectorio *ast*, el cual contiene el archivo *mailbox* [2].

Windows: `\usr\ast\mailbox`


UNIX: `/usr/ast/mailbox`

MULTICS: `>usr>ast>mailbox`

El otro tipo de nombre es el **nombre de ruta relativa**. Éste se utiliza en conjunto con el concepto del directorio de trabajo (también llamado directorio actual). Un usuario puede designar un directorio como el directorio de trabajo actual, en cuyo caso todos los nombres de las rutas que no empiecen en el directorio raíz se toman en **forma relativa** al directorio de trabajo. Por ejemplo, si el directorio de trabajo actual es `/usr/ast`, entonces el archivo cuya ruta absoluta sea `/usr/ast/mailbox` se puede referenciar simplemente como `mailbox`[2].

### Organización física.

Los datos son arreglados por su adyacencia física, es decir, de acuerdo con el dispositivo de almacenamiento secundario. Los registros son de tamaño fijo o de tamaño variable y pueden organizarse de varias formas para constituir archivos físicos.

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos II</b>	Código:	MADO-20
		Versión:	01
		Página	133/183
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

La organización física de un archivo en el almacenamiento secundario depende de la estrategia de agrupación y de la estrategia de asignación de archivos.

En esta guía no se abordará a profundidad la organización física, se deja a revisión en clase teórica.

## Desarrollo

Para poder realizar actividades en esta parte práctica solo se tratarán algunos aspectos sobre los archivos, como las operaciones, el tipo de acceso y organización lógica.

**Actividad 1:** Leer e implementar los ejemplos que se van describiendo para después poder realizar la actividad 2 propuesta por el profesor.

Antes de poder realizar cualquier operación de lectura/escritura en Python hay que realizar la apertura del archivo, esto se realiza con la función **open()** indicando su ubicación y nombre seguido, opcionalmente, por el modo o tipo de operación a realizar y la codificación que tendrá el archivo. Si no se indica el tipo de operación el archivo se abrirá en modo de lectura y si se omite la codificación se utilizará la codificación actual del sistema. Si no existe la ruta del archivo o se intenta abrir para lectura un archivo inexistente se producirá una excepción del tipo *IOError*.

Uno ejemplos del uso de la función son.

```
archivo=open('C:/Users/pc/Documents/prueba.txt', mode='r', encoding='utf-8')
```

```
archivo=open('C:/Users/pc/Documents/prueba.txt', 'r')
```


Para revisar el formato de codificación del sistema se puede utilizar las siguientes líneas:

```
import locale
print(locale.getpreferredencoding())
```

Una vez que el archivo ya no se utilizará debe cerrarse, eso se consigue con la siguiente línea:

```
archivo.close
```

Entonces una estructura general que considere la apertura, el fallo de apertura y el cierre es la siguiente:

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos II</b>	Código:	MADO-20
		Versión:	01
		Página	134/183
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

```
try:
    archivo=open('C:/Users/pc/Documents/prueba.txt', mode='r', encoding='utf-8')
    #Realizar operaciones con el archivo
except:
    print("error al abrir")
finally:
    if archivo:
        archivo.close
```


En este segmento e código se utiliza una estructura *try* para la captura de las excepciones.

Implícitamente en Python si el archivo no existe, para crearlo hay que indicar en el modo o tipo de operación, que lo haga. En la siguiente tabla se muestran los diferentes modos de apertura que se tienen en Python.

r	Modo de apertura	Ubicación del puntero
r	Solo lectura	Al inicio del archivo
rb	Solo lectura en modo binario	Al inicio del archivo
r+	Lectura y escritura	Al inicio del archivo
rb+	Lectura y escritura en modo binario	Al inicio del archivo
w	Solo escritura. Sobreescibe el archivo si existe. Crea el archivo si no existe	Al inicio del archivo
wb	Solo escritura en modo binario. Sobreescibe el archivo si existe. Crea el archivo si no existe	Al inicio del archivo
w+	Escritura y lectura. Sobreescibe el archivo si existe. Crea el archivo si no existe	Al inicio del archivo
wb+	Escritura y lectura en modo binario. Sobreescibe el archivo si existe. Crea el archivo si no existe	Al inicio del archivo
a	Añadido (agregar contenido). Crea el archivo si éste no existe	Si el archivo existe, al final de éste. Si el archivo no existe, al comienzo
ab	Añadido en modo binario (agregar contenido). Crea el archivo si éste no existe	Si el archivo existe, al final de éste. Si el archivo no existe, al comienzo
a+	Añadido (agregar contenido) y lectura. Crea el archivo si éste no existe.	Si el archivo existe, al final de éste. Si el archivo no existe, al comienzo
ab+	Añadido (agregar contenido) y lectura en modo binario. Crea el archivo si éste no existe	Si el archivo existe, al final de éste. Si el archivo no existe, al comienzo

Ahora para realizar la operación de lectura a un archivo en Python se tienen diferentes funciones, a continuación, se menciona y se muestra un ejemplo de uso de cada una.



	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos II</b>	Código:	MADO-20
		Versión:	01
		Página	135/183
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Con el método **read()** es posible leer un número de bytes determinados. Si no se indica número se leerá todo lo que reste o si se alcanzó el final de archivo devolverá una cadena vacía.

Ejemplo:

```
archivo=open('C:/Users/pc/Documents/prueba.txt', 'r')
c1=a.read(3)
c2=a.read()
print(c1)
#print(c2)
a.close()
```

El método **readline()** lee de un archivo una línea completa. Ejemplo:


```
a3=open('C:/Users/pc/Documents/prueba.txt', 'r')
while True:
    linea=a3.readline()
    if not linea:
        break
    print(linea)
a3.close()
```

El método **readlines()** lee todas las líneas de un archivo como una lista. Si se indica el parámetro de tamaño leerá esa cantidad de bytes del archivo y lo necesario hasta completar la última línea. Ejemplo:

```
archivo=open('C:/Users/pc/Documents/prueba.txt', 'r')
lista = archivo.readlines() #lee todas las lieneas a una lista
numlin = 0
for linea in lista:
    numlin += 1
    print(numlin, linea)
archivo.close()
```

La sentencia **with-as** permite usar los archivos de forma óptima cerrándolos y liberando la memoria al concluir el proceso de lectura.

```
with open('C:/Users/pc/Documents/prueba.txt', 'r') as archivo:
    for linea in archivo: # recorre línea a línea el archivo
        print(linea)
```

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos II</b>	Código:	MADO-20
		Versión:	01
		Página	136/183
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Para escribir en un archivo se utilizan los métodos `write()` y `writelines()`. El método **`write()`** escribe una cadena y el método **`writelines()`** escribe una lista a un archivo. Si en el momento de escribir el archivo no existe se creará uno nuevo. Ejemplo:


```
cadena1 = 'Datos'# declara cadena1
cadena2 = 'Secretos' # declara cadena2
archivo2 = open('C:/Users/pc/Documents/datos2.txt','w') # abre archivo para escribir
print(cadena1 + '\n')
archivo2.write(cadena1 + '\n') # escribe cadena1 añadiendo salto de línea
archivo2.write("hola") # escribe la cadena hola
archivo2.write(cadena2) # escribe cadena2 en archivo
archivo2.close # cierra archivo

lista = ['lunes', 'martes', 'miercoles', 'jueves', 'viernes'] # declara lists
archivo2 = open('datos2.txt','w') # abre archivo en modo escritura
archivo2.writelines(lista) # escribe toda la lista en el archivo
archivo2.close # cierra archivo
```

Para mover el apuntador de archivo se usa el método `seek()` que desplaza el puntero a una posición del archivo. También es posible conocer información sobre su posición para lo que se usa el método `tell()`, que devuelve la posición del puntero en un momento dado (en bytes). Ejemplo:

```
archivo=open('C:/Users/pc/Documents/prueba.txt', 'r')
archivo.seek(5) # mueve puntero al quinto byte
cadena1 = archivo.read(5) # lee los siguientes 5 bytes
print(cadena1) # muestra cadena
print(archivo.tell()) # muestra posición del puntero
archivo.close # cierra archivo
```

Para leer y escribir cualquier tipo de objeto Python se importa el módulo `pickle` y se usan sus métodos `dump()` y `load()` para leer y escribir los datos. Ejemplo:

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos II</b>	Código:	MADO-20
		Versión:	01
		Página	137/183
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			


```
import pickle # importa módulo pickle
lista = ['algoritmos 1', 'algoritmos 2', 'estructuras'] # declara lista
archivo = open('materias.dat', 'wb') # abre archivo binario para escribir
pickle.dump(lista, archivo) # escribe lista en archivo
archivo.close # cierra archivo
del lista # borra de memoria la lista
archivo = open('materias.dat', 'rb') # abre archivo binario para leer
lista = pickle.load(archivo) # carga lista desde archivo
print(lista) # muestra lista
archivo.close # cierra archivo
```

También es posible crear directorios y recorrerlos desde Python. Ejemplo:

```
#crear un directorio
import os
def crear_directorio(ruta):
    try:
        os.makedirs(ruta)
    except OSError:
        pass
    os.chdir(ruta)
crear_directorio('C:/Users/pc/Documents/nuevo')
```

```
#recorrer un directorio
import os
rootDir = 'C:/Users/pc/Documents'
for dirName, subdirList, fileList in os.walk(rootDir):
    print('Directorio encontrado: %s' % dirName)
    for fname in fileList:
        print('\t%s' % fname)
```

Finalmente, algo importante es que se pueden conocer algunos de los atributos de un archivo. Ejemplo:

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos II</b>	Código:	MADO-20
		Versión:	01
		Página	138/183
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

```

archivo=open('C:/Users/pc/Documents/prueba.txt', 'r')
contenido = archivo.read()
nombre = archivo.name
modo = archivo.mode
encoding = archivo.encoding
archivo.close()

if archivo.closed:
    print ("El archivo se ha cerrado correctamente")
else:
    print ("El archivo permanece abierto")
print (contenido)
print (nombre)
print (modo)
print (encoding)

```

## Actividad 2

Ejercicios sugeridos por el profesor

## Referencias

[1] Abraham Silberschatz, Peter Baer Galvin & Greg Gagne  
Fundamentos de Sistemas Operativos  
Séptima Edición  
MacGrall Hill

[2] Andrew S. Tanenbaum  
Sistemas Operativos Modernos  
Tercera Edición

[4] <https://www.python.org/>

[3] <http://python-para-impacientes.blogspot.mx/2014/02/operaciones-con-archivos.html>