

Manejo de Archivos en Python

Existen dos formas básicas de acceder los datos de un archivo: una es utilizarlo como un archivo de texto y acceder a los datos de forma secuencial (línea por línea) y la otra es tratarlo como un archivo binario para acceder a los datos byte por byte.

1. Archivos de Texto con acceso Secuencial.

Los archivos así denominados son aquellos en donde su contenido corresponde a cadenas de caracteres de texto libre y que cada línea finaliza con un retorno de carro (ASCII 13) y un salto de línea (ASCII 10).

Para su lectura, de tal forma que sea posible la recuperación de los datos individuales, es necesario que estén almacenados de una manera apropiada, generalmente con longitudes fijas para cada dato o mediante un símbolo que delimite un dato de otro.

Algunos ejemplos de archivos de texto para ser accedido en forma secuencial:

Ejemplo 1: Datos separados por un carácter delimitador (;)

10765433-3; ROSA ESPINOZA; 43; AV. LIBERTAD 135; F
12345837-2; ALAN BRITO DELGADO; 22; 1 NORTE 666; M
19654233-K; YAN CLOCK VANDAM; 8; CALLE COVANDONGA 123; M

Ejemplo 2: Datos con longitud fija.

10765433-3	ROSA ESPINOZA	43	AV. LIBERTAD 135	F
12345837-2	ALAN BRITO DELGADO	221	NORTE 666	M
19654233-K	YAN CLOCK VANDAM	8	CALLE COVANDONGA 123M	

Cada línea, en los ejemplos anteriores, finaliza con unos caracteres de control (0D0A en hexadecimal o 13 y 10 en decimal según la tabla ASCII)

New file

Open file

Reload

Export

Undo

Redo

Tools

Settings

Help

Go To

Current Address

0x0000005E

Memo

Last Address

0x00000094

Go to

File Information

File Name

x.txt

File Size

149 bytes

Data Inspector (Little-endian)

Type	Unsigned (+)	Signed (+)
8-bit Integer	10	10
16-bit Integer	12554	12554
24-bit Integer	3748106	3748106
32-bit Integer	909717770	909717770
64-bit Integer (+)	3689068447917486346	
64-bit Integer (+)	3689068447917486346	
16-bit Float. P	0,1574707	
32-bit Float. P	0,0000027595693	
64-bit Float. P	4,425131943097653e-62	
MS-DOS DateTime	2007-01-25 06:08:20 Local	
OLE 2.0 DateTime	1899-12-30 00:00:00.000 UTC	
UNIX DateTime	1998-10-30 03:22:50 UTC	
Macintosh HFS	1932-10-28 23:22:50 Local	

00000000

31 30 37 36 35 34 33 33

2D 3B 38 52 4F 53 41 20

10765433-3; ROSA.

00000010

45 53 50 49 4E 4F 5A 41

38 34 33 3B 41 56 2E 4C

ESPINOZA;43;AV..L

00000020

49 42 45 52 54 41 44 20

31 33 35 3B 46 00 0A 31

IBERTAD.135;F..1

00000030

32 33 34 35 38 33 37 2D

32 3B 41 4C 41 4E 20 42

2345837-2;ALAN.B

00000040

52 49 54 4F 20 44 45 4C

47 41 44 4F 3B 32 32 3B

RITO.DELGADO;22;

00000050

31 20 4E 4F 52 54 45 20

36 36 36 3B 4D 00 0A 31

1.NORTE.666;M..1

00000060

39 36 35 34 32 33 33 2D

4B 38 59 41 4E 20 43 4C

9654233-K;YAN.CL

00000070

4F 43 4B 20 56 41 4E 44

41 4D 3B 38 3B 43 41 4C

OCK.VANDAM;8;CAL

00000080

4C 45 20 43 4F 56 41 4E

44 4F 4E 47 41 20 31 32

LE.COVANDONGA.12

00000090

33 3B 4D 00 0A +

3;M..

Search

Search for

Data Type

☐ 8-bit Integer
 ☐ 16-bit Integer
 ☐ 24-bit Integer
 ☐ 32-bit Integer
 ☐ 64-bit Integer
 ☐ 16-bit Floating Point
 ☐ 32-bit Floating Point
 ☐ 64-bit Floating Point
 ☐ Hexadecimal Values
 ☐ Text

Byte Order

☒ Little-endian
 ☐ Big-endian

En la figura anterior, se puede visualizar los datos del ejemplo 1 almacenados en un archivo. En la parte central se aprecia la representación de cada carácter en hexadecimal y al lado derecho en formato ASCII. Observe que después de cada registro le siguen los caracteres 0D y 0A, correspondientes a los caracteres de control Retorno de Carro y Fin de Línea. Estos caracteres provocan que en un editor de texto cada registro se vea en líneas diferentes. De esta forma, la lectura se puede realizar línea por línea.

Para hacer uso de un archivo de texto, que contenga datos estructurados, se debe realizar las siguientes acciones: Abrir el archivo (Apertura), Realizar operaciones con los registros (Lectura/Escritura a modo de adicción), Cerrar el Archivo (Cierre).

Cabe señalar que, en estricto rigor, solamente se puede leer los registros de un archivo o añadir (al final) nuevos registros. No se puede modificar los registros ya escritos.

1.1. Apertura.

Para abrir un archivo, se emplea el método **open**. Su sintaxis es la siguiente:

Sintaxis: `VariableArchivo = open("Nombre_del_archivo" [,Modo_de_apertura])`

Por ejemplo: `Archivo = open("PERSONAS.TXT")`

— **VariableArchivo:** Es el nombre de una variable que estará asociada al archivo y que a través de ella se realizarán las operaciones sobre el archivo.

— **Nombre_del_Archivo (Obligatorio):**

Corresponde al nombre que tiene el archivo en el disco. Se puede señalar la ruta para acceder al archivo, por ejemplo: `Archivo = open("C:\TRABAJOS\PERSONAS.TXT")`

Si se omite la ruta se asume la carpeta de trabajo.

— **Modo_de_apertura (Opcional):**

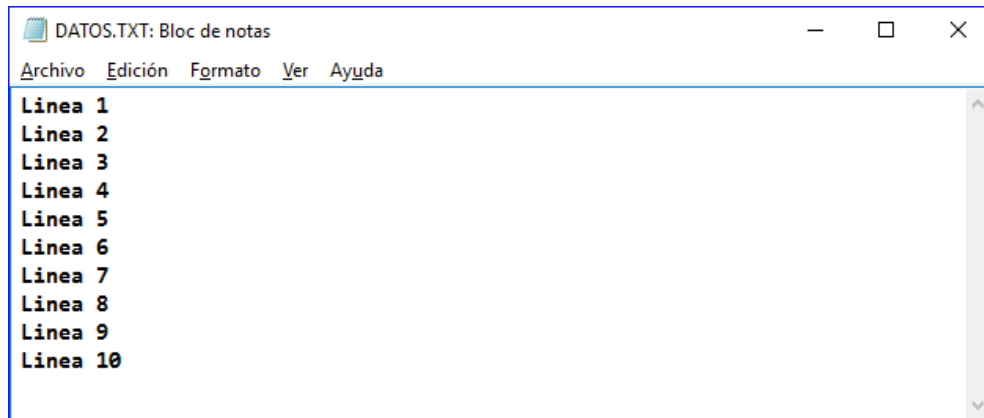
- Modo de **sólo lectura** (r). En este caso no es posible realizar modificaciones sobre el archivo, solamente leer su contenido.
- Modo de **sólo escritura** (w). En este caso el archivo es truncado (vaciado) si existe, y se lo crea si no existe.
- Modo **adicción** (a). En este caso se crea el archivo, si no existe, pero en caso de que exista se posiciona al final, manteniendo el contenido original, y la escritura se realiza a continuación de la última línea.

— **Observaciones.**

- 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, todos los datos anteriores son borrados y reemplazados por lo que se escriba en él.

1.2. Lectura.

Existen varias formas de realizar la lectura desde el archivo. Los siguientes ejemplos emplean el archivo DATOS.TXT el que contiene el siguiente texto:



1.2.1. Forma 1. Empleando el método read.

Sintaxis: Variable = VariableArchivo.read()

Algoritmo 1: En este ejemplo se emplea el método `read`, el cual lee, de una vez, todas las líneas del archivo.

```
01 # -*- coding: utf-8 -*-
02
03 Archivo = open("DATOS.TXT", "r")
04 Datos = Archivo.read()
05 print(Datos)
06 Archivo.close()
07 input("Presione Enter para finalizar")
```

El resultado en pantalla es el siguiente:

```
Linea 1
Linea 2
Linea 3
Linea 4
Linea 5
Linea 6
Linea 7
Linea 8
Linea 9
Linea 10

Presione Enter para finalizar
```

1.2.2. Forma 2. Empleando el método readline.

Sintaxis: Variable = VariableArchivo.readline()

Algoritmo 2: En este ejemplo se emplea el método `readline`, el cual lee solamente una línea del archivo (en realidad lee todo el texto hasta encontrarse con los caracteres de control de Retorno de Carro y Salto de Línea).

```
01 # -*- coding: utf-8 -*-
02
03 Archivo = open("DATOS.TXT", "r")
04 Linea = Archivo.readline()
05 print(Linea)
06 Linea = Archivo.readline()
07 print(Linea)
08 Linea = Archivo.readline()
09 print(Linea)
10 Archivo.close()
11 input("Presione Enter para finalizar")
```

El resultado en pantalla es el siguiente:

```
Linea 1
Linea 2
Linea 3
Presione Enter para finalizar
```

Por lo tanto, para leer todas las líneas del archivo se debería repetir la sentencia de lectura hasta que ya no haya más líneas que leer.

Algoritmo 3: Lectura de todas las líneas (registros) de un archivo.

```
01 # -*- coding: utf-8 -*-
02
03 Archivo = open("DATOS.TXT", "r")
04 Contador = 0
05 Linea = Archivo.readline()
06 while Linea != "":
07     Contador = Contador + 1
08     print(str(Contador) + ". " + Linea)
09     Linea = Archivo.readline()
10 Archivo.close()
11 input("Presione Enter para finalizar")
```

El resultado en pantalla es el siguiente:

```
1. Línea 1
2. Línea 2
3. Línea 3
4. Línea 4
5. Línea 5
6. Línea 6
7. Línea 7
8. Línea 8
9. Línea 9
10. Línea 10

Presione Enter para finalizar
```

Otra forma de leer el archivo línea por línea, obteniendo el mismo resultado anterior, sería el siguiente:

Algoritmo 4: Lectura más eficiente de todos los registros de un archivo.

```
01 # -*- coding: utf-8 -*-
02
03 Archivo = open('DATOS.TXT','r')
04 Contador = 0
05 for Línea in Archivo:
06     Contador = Contador + 1
07     print(str(Contador) + '. ' + Línea)
08 Archivo.close()
09 input('Presione Enter para Finalizar')
```

Observación:

Note que al momento de escribir en pantalla, se despliega una línea en blanco entre cada línea leída desde el archivo. Esto se debe a que la lectura, empleando `readline`, considera los caracteres de control Fin del Línea y Retorno de Carro como parte del texto leído. Para evitar esto, se debe modificar la línea 07 añadiendo el parámetro `end = ""`, el resultado sería el siguiente:

```
1. Línea 1
2. Línea 2
3. Línea 3
4. Línea 4
5. Línea 5
6. Línea 6
7. Línea 7
8. Línea 8
9. Línea 9
10. Línea 10
Presione Enter para Finalizar
```

Otra forma sería empleando el método `strip`.

1.2.3. Forma 3. Empleando el método readlines.

Sintaxis: Variable = VariableArchivo.readlines()

En este ejemplo se emplea el método **readlines**, el cual lee todo el archivo generando una lista o vector conteniendo todos los registros leídos.

Algoritmo 5: Lectura de todos los registros de un archivo, quedando almacenados en una lista o vector.

```
01 # -*- coding: utf-8 -*-
02
03 Archivo = open("DATOS.TXT", "r")
04 Linea = Archivo.readlines()
05 print(Linea)
06 Archivo.close()
07 input("Presione Enter para finalizar")
```

El resultado obtenido es una lista con los valores de cada línea.

```
['Línea 1\n', 'Línea 2\n', 'Línea 3\n', 'Línea 4\n', 'Línea 5\n', 'Línea 6\n', 'Línea 7\n', 'Línea 8\n', 'Línea 9\n', 'Línea 10\n']
Presione Enter para finalizar
```

En definitiva, las formas de lectura 1 y 3 se pueden utilizar cuando la cantidad de datos a leer no sean muchos. De esta forma, el método más recomendable es el 2, por lo que emplearemos los algoritmos 3 y 4 cuando debamos recorrer un archivo.

1.3. Escritura.

Básicamente existen 2 formas de realizar escritura en un archivo de texto: empleando el método **write** o el método **writelines**. Dependiendo del modo de apertura, la cadena será almacenada al comienzo (apertura en modo “w”) o a continuación de la última línea (apertura en modo “a”)

1.3.1. Forma 1. Empleando el método write.

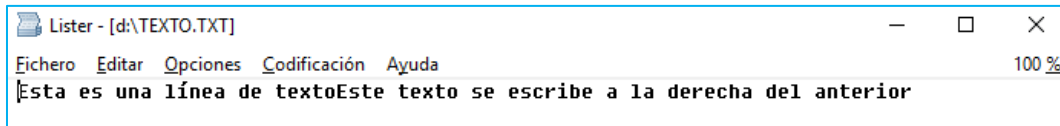
El método **write** escribe una cadena de caracteres en el archivo.

Sintaxis: VariableArchivo.write(CadenaDeCaracteres)

Algoritmo 6: Escritura sin salto de línea.

```
01 # -*- coding: utf-8 -*-
02
03 Archivo = open("TEXT0.TXT", "w")
04 Archivo.write("Esta es una línea de texto")
05 Archivo.write("Este texto se escribe a la derecha del anterior")
06 Archivo.close()
07 input("Presione Entre para Finalizar")
```

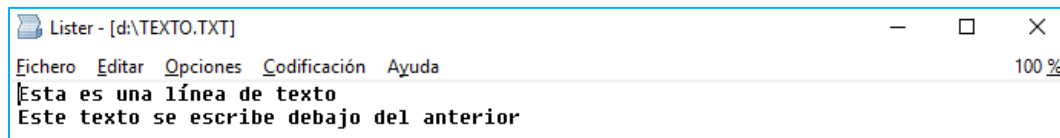
El resultado sería el siguiente:



Algoritmo 7: Escritura con salto de línea.

```
01 # -*- coding: utf-8 -*-
02
03 archivo = open("TEXT0.TXT", "w")
04 archivo.write("Esta es una línea de texto\n ")
05 archivo.write("Este texto se escribe debajo del anterior")
06 archivo.close()
07 input("Presione Entre para Finalizar"))
```

El resultado sería el siguiente:



1.3.2. Forma 2. Empleando el método writelines.

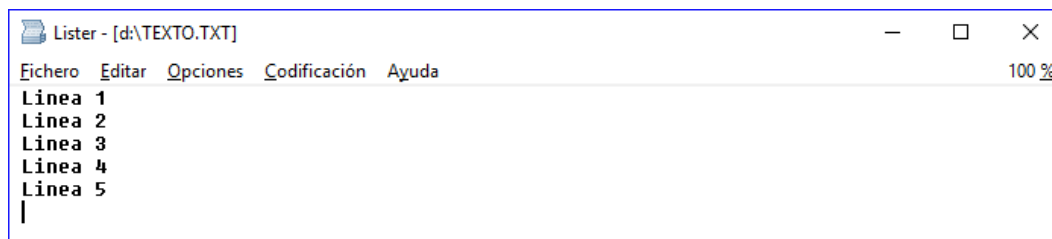
El método **writelines** escribe una lista (vector) en un archivo.

Sintaxis: VariableArchivo.**writelines**(CadenaDeCaracteres)

Algoritmo 8: El siguiente ejemplo almacena en el archivo TEXT0.TXT la lista almacenada en la variable Linea.

```
01 # -*- coding: utf-8 -*-
02
03 Linea = ['Linea 1\n', 'Linea 2\n', 'Linea 3\n', 'Linea 4\n', 'Linea 5\n']
04 Archivo = open("d:\TEXT0.TXT", "w")
05 Archivo.writelines(Linea)
06 Archivo.close()
07 input("Presione Entre para Finalizar")
```

El resultado es el siguiente:



```
Lister - [d:\TEXTO.TXT]
Fichero  Editar  Opciones  Codificación  Ayuda
Linea 1
Linea 2
Linea 3
Linea 4
Linea 5
|
```

En el ejemplo anterior, cada elemento de la lista iba acompañado de los caracteres de control de salto de línea y retorno de carro (`\n`)

1.4. Cierre.

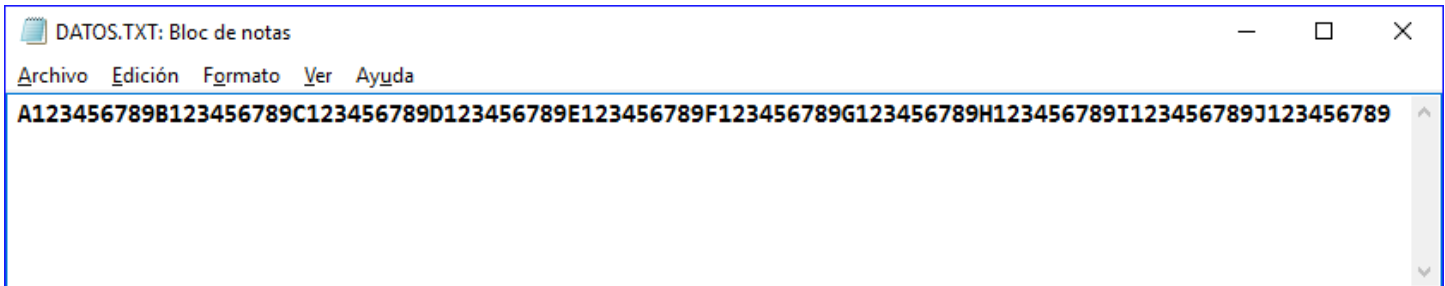
Todo archivo abierto debe ser cerrado, para ello se emplea la función `close`

Sintaxis: `VariableArchivo.close()`

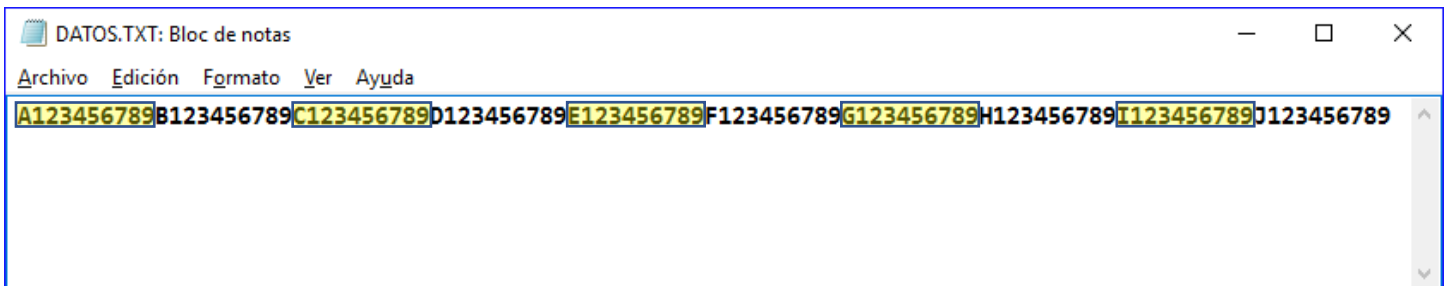
En rigor, todos los algoritmos anteriores debieran finalizar con la sentencia: `Archivo.close()`

2. Manejo de archivos con acceso directo.

Para los siguientes ejemplos, emplearemos el archivo DATOS.TXT, el que tendrá los siguientes datos almacenados:



Como se puede apreciar, un requisito fundamental para poder realizar un acceso directo a un registro es que cada uno tenga una longitud fija y que no se incorporen saltos de línea (aunque sí los podría tener, pero solamente si la longitud de cada campo sea fija, como se presenta en el # Ejemplo 2). Por lo tanto, los registros quedan destacados como se presentan a continuación:



A123456789 sería un registro, B123456789 sería otro, y así sucesivamente.

Internamente, se maneja un puntero que apunta a un byte en particular (un carácter en particular), por lo que es posible ubicar dicho puntero haciendo un simple cálculo matemático y ocupando el método [seek](#). A este puntero se le llama **puntero del archivo**.

En la siguiente tabla se muestran los métodos posibles a usar para el acceso directo a los registros del archivo.

MÉTODO	ACCIÓN
archivo.tell()	Entrega la posición actual del puntero del archivo.
archivo.seek	Permite desplazarse una cantidad inicio de bytes en el archivo, contando desde el comienzo del archivo, desde la posición actual o desde el final.
archivo.read(n)	Lee n bytes (caracteres)
archivo.write(contenido)	Escribe en la posición actual

2.1. Ejemplo 1. Lectura de los 5 primeros registros.

Algoritmo 9

```
01 Archivo = open("DATOS.TXT", "r")
02 Largo_Registro = 10
03
04 Linea = Archivo.read(Largo_Registro)
05 print(Linea)
06
07 Linea = Archivo.read(Largo_Registro)
08 print(Linea)
09
10 Linea = Archivo.read(Largo_Registro)
11 print(Linea)
12
13 Linea = Archivo.read(Largo_Registro)
14 print(Linea)
15
16 Linea = Archivo.read(Largo_Registro)
17 print(Linea)
18
19 Archivo.close()
20 input("Presione Enter para finalizar")
```

```
A123456789
B123456789
C123456789
D123456789
E123456789
Presione Enter para finalizar
```

2.2. Ejemplo 2. Lectura de los 3 primeros registros con posición sea impar.

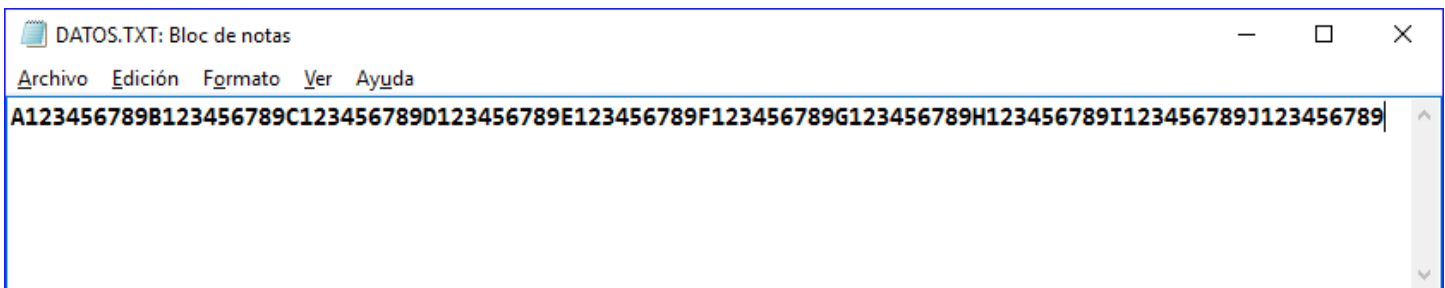
Algoritmo 10.

```
01 Archivo = open("DATOS.TXT", "r+")
02 Largo_Registro = 10
03
04 Archivo.seek(1 * Largo_Registro)
05 Linea = Archivo.read(Largo_Registro)
06 print(Linea)
07
08 Archivo.seek(3 * Largo_Registro)
09 Linea = Archivo.read(Largo_Registro)
10 print(Linea)
11
12 Archivo.seek(5 * Largo_Registro)
13 Linea = Archivo.read(Largo_Registro)
14 print(Linea)
15
16 Archivo.close()
17 input("Presione Enter para finalizar")
```

```
B123456789
D123456789
F123456789
Presione Enter para finalizar
```

2.3. Cómo saber cuántos registros hay almacenados.

Para saber cuántos registros tienen un archivo de datos, es necesario saber cuántos bytes tiene éste y dicha cantidad dividirlo por la longitud del registro. El siguiente archivo (DATOS.TXT) está compuesto por 10 registros de longitud 10 (caracteres), por lo que el tamaño del archivo debiera ser 100 Bytes y la cantidad de registros 10:



El siguiente código emplea la biblioteca `os` para hacer uso del método `getsize` para obtener el tamaño, en Bytes, del archivo.

Algoritmo 11.

```
01 import os
02
03 Largo_Registro = 10
04 Tamaño_Archivo = os.path.getsize("DATOS.TXT")
05 Cantidad_Registros = int(Tamaño_Archivo / Largo_Registro)
06
07 print("Tamaño del Archivo : " + str(Tamaño_Archivo) + " Bytes.")
08 print("Cantidad de Registros: " + str(Cantidad_Registros) + " Registros.")
09
10 input("Presione Enter para Finalizar")
```

El resultado en pantalla es el siguiente:

```
Tamaño del Archivo : 100 Bytes.
Cantidad de Registros: 10 Registros.
Presione Enter para Finalizar
```

2.4. Cómo saber en qué registro se encuentra el puntero del archivo.

Esto resulta de la división de la posición del puntero del archivo, entregado por el método `tell`, por el largo del registro.

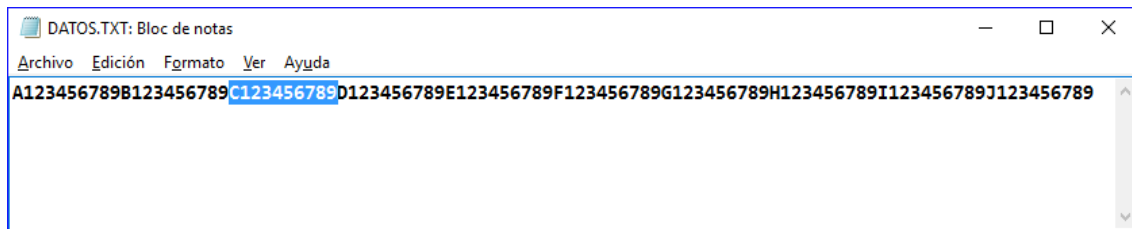
$$\text{Posicion} = \text{Archivo.tell()} / \text{Largo_Registro}$$

Debemos recordar que el método `tell` entrega la posición del byte en donde se encuentra el puntero del archivo.

2.5. Cómo sobrescribir un registro.

El siguiente ejemplo, ubica el puntero del archivo al comienzo del tercer registro (recordemos que el primero está en la posición 0 y el tercero en la posición 2). Ese registro corresponde al valor `C123456789` el que será reemplazado por el valor `Z123456789`. Observe, además, que el archivo `DATOS.TXT` fue abierto para su lectura y escritura (`r+`). Por otra parte, se ha destacado en azul el registro que será modificado

Archivo original (antes de ejecutar el siguiente programa)

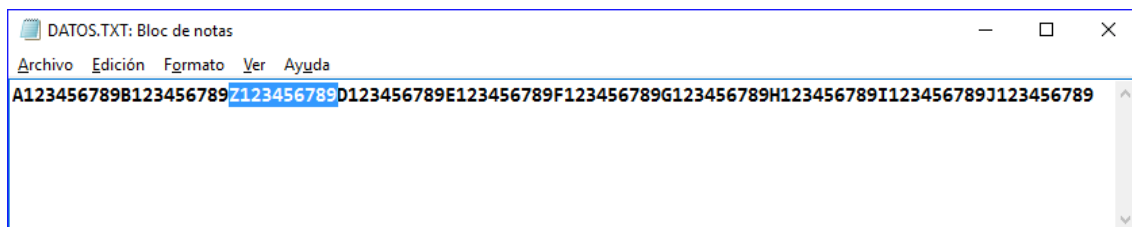


Programa que modificará el archivo:

Algoritmo 12.

```
01 Archivo = open("DATOS.TXT", "r+")
02 Largo_Registro = 10
03
04 Archivo.seek(2 * Largo_Registro)
05 Archivo.write("Z123456789")
06 Archivo.close()
07 input("Presione Enter para finalizar")
```

Archivo modificado (después de ejecutar el anterior programa)



3. ¿Cómo crear directorios (carpetas) en Python?

El manejo de directorios en Python es bastante simple, el objetivo es hacer más con menos código, así que comenzamos con una versión primitiva de cómo crear un directorio, usando siempre el paquete “os” que maneja este y muchos otros asuntos:

Algoritmo 13.

```
01 import os
02
03 def crear_directorio(ruta):
04     try:
05         os.makedirs(ruta)
06     except OSError:
07         pass
08     # si no podemos crear la ruta dejamos que pase
09     # si la operación resulto con éxito nos cambiamos al directorio
10     os.chdir(ruta)
```

Si lo queremos en menos líneas:

Algoritmo 14.

```
01 nuev Ruta = r'ruta/al/directorio/directorio'
02 if not os.path.exists(nuevaruta): os.makedirs(nuevaruta)
```

Este último se asegura que la ruta no exista, y si no existe pues la crea. Ahora veamos uno más compacto aun usando un shell del sistema operativo:

Algoritmo 15.

```
01 import os
02 os.system("mkdir /ruta/al/directorio")
```

si dentro de la ruta existe una “carpeta” que no existe esta será creada.

3.1. Como renombrar (mover) un archivo en python

Renombrar un archivo en Python puede resultar de usar la misma funcionalidad que se usa para mover un archivo, el truco consiste en “mover” un archivo en el directorio x al mismo directorio x pero con diferente nombre. Todo lo que toma es importar el módulo “shutil” y usar su función “move” que mueve directorios de un lado a otro.

Algoritmo 16.

```
01 import shutil
02
03 def move(src, dest):
04     shutil.move(src, dest)
```

Pero... entonces para que sirve el “os.rename”? Bueno aunque su nombre es mas descriptivo, resulta que el “shutil.move” utiliza el “os.rename” si el directorio fuente es igual al directorio de destino, y utiliza el “shutil.copy2” para copiar los archivos si el directorio es distinto, entonces en “shutil.move” tenemos una forma compacta de ocuparnos de ambos casos.

3.2. Como recorrer un directorio en Python

Algoritmo 17. Recorrido Básico.

```
01 import os
02
03 # Setteamos el directorio raiz a la variable rootDir
04 # En realidad la variable se puede llamar como sea :)
05
06 rootDir = '.'
07 for dirName, subDirList, fileList in os.walk(rootDir):
08     print('Directorio encontrado: %s' % dirName)
09     for fname in fileList:
10         print('\t%s' % fname)
```

la función “os.walk” se encarga de la recursividad y en cada iteración nos devuelve los siguientes valores:

- dirName: El siguiente directorio que encontré
- subDirList: Una lista de todos los sub-directorios en el directorio actual.
- fileList: Una lista de archivos en el directorio actual

Otra forma:

Algoritmo 18.

```
01 import os
02
03 rootDir = '.'
04 for dirName, subDirList, fileList in os.walk(rootDir, topdown=False):
05     print('Found directory: %s' % dirName)
06     for fname in fileList:
07         print('\t%s' % fname)
```

La diferencia de este snippet de código es que le pasamos el parámetro topdown el cual nos permite recorrer el árbol desde sus “hojas” en lugar de recorrerlo desde la raíz, así pues, el output es los archivos primero y luego los directorios. Mas eficaz si lo que buscamos puede estar en el último folder del directorio.

3.3. Revisar si un directorio o archivo existe en Python.

Existen varias formas de hacer esta operación, veamos algunas:

1. os.path.isfile(ruta)

Algoritmo 19.

```
01 os.path.isfile("foo.txt")
```

2. os.access(path, mode)

Algoritmo 20.

```
01 os.access("foo.txt", os.R_OK) # revisa si existe y se puede leer
02 os.access("foo.txt", os.W_OK) # revisa si existe y se puede escribir
```

3. open dentro de un try catch

Algoritmo 21.

```
01 try:
02     f = open("foo.txt")
03 except IOError as e:
04     print("Uh oh! Esto no existe")
```

En el caso 1 y 2 ambos regresan un valor booleano según el archivo sea accesible o no, en la mayoría de los casos el ejemplo 1 debería bastar.