Archivos

Unidad 5

Apunte de cátedra

Pensamiento Computacional (90)
Cátedra: Camejo

.UBAXXI

Archivos

Las Estructuras de Datos **persistentes** (perduran más allá de la vida de quien las genera) se conocen genéricamente como **archivos**. Claramente, un archivo debe guardarse en algún dispositivo que administre datos **no volátiles** (es decir, no desaparecen cuando se apaga el equipo). Por ello los archivos (en todos sus formatos y organizaciones) son las Estructuras de Datos que se emplean para compartir información entre usuarios, procesos, equipos.

Si generamos un grupo de datos corriendo un programa, ya sea que se ingresen por teclado o se generen como resultado de transformaciones, y deseamos acceder a ellos en otro momento, o desde otro programa, necesitamos guardarlos en una Estructura capaz de sobrevivir a la ejecución corriente (proceso activo). De esta manera, se pueden rescatar ulteriormente y seguir trabajando con ellos. Como lo hacemos habitualmente con un texto que estamos editando, una planilla que actualizamos mensualmente, o un programa Python que estamos generando. La manera de conservarlos para más tarde es grabándolos, como archivos, con un nombre, una extensión, y claramente una organización interna conveniente.

Todos los lenguajes de Programación admiten la interacción con archivos. Cada lenguaje provee una colección de sentencias a tales efectos.

La diversidad de tipos de archivos es enorme. Cada uno de ellos tiene sus particularidades y requiere ciertas formas específicas en su manejo. Además, dependiendo de su morfología, el tipo de datos que maneja, y sus dimensiones puede requerir algoritmos especiales para su empleo.

En este apunte simplemente daremos algunas directivas básicas para que puedas trabajar con tipos simples (y disponibles en cualquier lado) de archivos y las operaciones esenciales que te requerirá su uso.

Lo primero que debemos recordar es que estamos haciendo programas que correrán en una **Arquitectura Von Neumann**. Por lo tanto, para que los datos puedan ser procesados, primero deben ser alojados en Memoria Interna (**MI**). Y para que se puedan sacar datos y resultados (de la computadora) hacia dispositivos externos (discos, redes, etc.), donde estará alojado nuestro archivo, deben pasar por la **MI**.

Así que no hay posibilidad de trabajar con los datos que están en un archivo si no los copiamos a MI, si los modificamos y deseamos guardarlos modificados, deberemos bajar la versión que tengamos en MI al archivo.

Pero normalmente los archivos contienen grandes volúmenes de datos, por ello, copiar un archivo completamente en MI puede ser una tarea muy costosa, sino imposible. Así que una solución salomónica es

copiar los datos por partes. A medida que necesitamos información la subimos a MI. Y a medida que modificamos información, resguardamos una copia en el archivo. Como ya se sabe que vamos a estar trayendo y enviando datos al archivo de manera continua, en lugar de generar un montón de variables, el Sistema Operativo (**SO**) reserva una zona de memoria (que está dentro de la región del programa que emplea el archivo) y la establece como el canal de comunicación. Así el programa sabe que al traer algo desde el archivo lo encontrará allí y si quiere respaldar algún cambio, lo hará desde allí. Genéricamente estos miniespacios de intercambio entre un programa y un archivo se denominan **buffers**.

Por supuesto que el costo operativo de ir a buscar al archivo un dato, ponerlo en el buffer y recién usarlo, es alto (sobre todo porque los archivos normalmente se encuentran alojados en medios más lentos que la MI). Por ello los SO en realidad implementan un sistema de tráfico de intercambio, **lectura y escritura en bloque**. Es decir, si el programa necesita un dato en particular, no se trae sólo ese dato, sino todo un bloque de información. No entraremos en detalle con respecto a los criterios y tamaños de ablocamiento, pero basta con saber que suele ser un esquema de trabajo estadísticamente muy eficiente.

Sin embargo, esta avivada del SO no modifica nuestro trabajo dentro de un programa, por lo tanto, la ignoraremos…

El proceso de preparar la sala de reuniones con el archivo deseado (pedirle al SO que defina un buffer) se llama genéricamente **abrir archivo**. Al abrir un archivo le estamos pidiendo al SO que nos asigne un buffer, dándole datos relevantes, tales como el nombre, dónde se ubica en la estructura de directorios, qué haremos con él, entre otras cosas. Además, le asignamos un **nombre interno (alias)** para referenciarlo directamente con ese nombre dentro del programa. Ya que **un mismo programa puede abrir simultáneamente todos los archivos que necesite.**

Primitivas Generales

Para abrir un archivo en Python disponemos de la función **open**(). Esta función le solicita al SO que establezca el vínculo de trabajo y nos devuelva la dirección de inicio del buffer del archivo. Es menester que guardemos esa dirección en una variable; que será el nombre interno del archivo desde ese momento. Es decir que, el nombre interno de nuestro archivo es una etiqueta que apunta al inicio del buffer.

Nota:

Una vez abierto, sólo referenciamos al archivo por su nombre interno o alias.

Sintaxis de open:

open(nombre_completo_archivo[,modo])

nombre_completo_archivo

Debe indicarse el nombre y extensión con el que fue guardado; así como localización completa, o relativa en los directorios (el inefable path)

modo

Los modos (para qué será empleado el archivo dentro del programa) pueden ser los siguientes (debemos emplear el símbolo para especificar la acción):

'r'	Lectura
'r+'	Lectura/Escritura
'w'	Sobreescritura. Si no existe archivo se creará
'a'	Añadir. Escribe al final del archivo

Por defecto la conexión se abre en modo lectura, con lo que no es posible escribir en el archivo. Para poder escribir es necesario utilizar la opción 'w' con la que se eliminará cualquier archivo existente y creará uno nuevo. Otra opción que se puede utilizar es 'a', con la que se añadirá nuevo contenido al archivo existente, o 'r+' para leer y escribir.

Una vez que **terminamos de usar** el archivo es importante realizar la **operación de cierre**. Ya que esta operación libera el espacio de buffer y completa el proceso de grabado, si hay algo pendiente. Así como el SO trae más información de la que le pedimos (por el ablocamiento) sin preguntarnos, tampoco se molesta en salir corriendo a grabar un dato cuando se lo solicitamos, en realidad junta varios encargos de grabado antes de realizar la tarea.

El cierre del archivo lo obliga a grabar todo lo que haya quedado pendiente. Para cerrar un archivo disponemos del método **close**().

Archivos Planos – Archivos de Texto (txt)

¿Qué podemos hacer con un archivo de texto?

Seguramente traer líneas o bytes desde él (lectura), para trabajarlos en el programa, y eventualmente guardar cambios y agregados (escritura).

Para realizar lecturas disponemos de: **read**(),**readline**(),**readlines**()

Para escritura: **write**(),**writelines**()

read() permite leer una determinada cantidad de bytes, si no se pone nada, lee hasta el final.

readline() lee la siguiente línea (caracteres hasta el próximo \n).

readlines() permite leer varias líneas, si no se indica nada, lee todas las que falten.

write() permite escribir en el archivo lo que se envía como argumento.

writelines() permite escribir varias líneas en un archivo.

Nota:

\n es un carácter de control, por eso se escribe con dos caracteres, el primero \ que indica que lo que sigue debe ser interpretado de manera especial. \n es el carácter de bajada de línea. Cuando agregamos \n al final de cualquier texto, si lo mostramos por pantalla al finalizar el texto el cursor baja una línea.

Veamos este ejemplo:

- Agrega oraciones en arch1.txt

```
vacaTxt=open('arch1.txt','r+')
t=vacaTxt.readline()
print(t)

t=input('Ingresá un texto con vaca: ')
while (t.lower()).count('vaca')==0:

    t=input('Ingresá un texto con vaca: ')
vacaTxt.write(t+'\n')
vacaTxt.close()
vacaTxt=open('arch1.txt')

todas=vacaTxt.readlines()
for linea in todas:

    print(linea.strip('\n'))
vacaTxt.close()

exit()
```

Atención!

*Existen diferentes formatos de codificación para archivos de texto. Al usar **open**() como en el ejemplo, estamos dejando que Python abra el archivo con la codificación que tiene por defecto el **SO**. Para modificar eso, o simplemente para asegurarnos de que el archivo sea abierto correctamente en diferentes plataformas se puede emplear el parámetro **encoding**.*

Ejemplo:

```
vacaTxt=open('arch2.txt','r+',encoding='utf_8')
```

*Acá se está pidiendo que **arch2.txt** sea interpretado como grabado con codificación **utf_8**, una de las clásicas. Puede ser **ansi**, u otras más.*

Nota:

*En este ejemplo tenemos en la misma carpeta del programa un archivo: **arch1.txt** que contiene al momento de correr el programa lo siguiente:*

```
La vaca LOCA
VACAYENDO gente al baileee
siga la vaca
vacaciones en faMIlia...
es lila la vaca de MILKA
```

Por lo que, si ejecutamos el programa, tendremos una salida como esta:

```
La vaca LOCA

Ingresá un texto con vaca:    tanto va el cántaro a la fuente, que al
final se rompe
Ingresá un texto con vaca:    Señora VACaaa, señora VACAAA!!!

La vaca LOCA
VACAYENDO gente al baileee
siga la vaca
vacaciones en faMIlia...
es lila la vaca de MILKA
Señora VACaaa, señora VACAAA!!!
>>>
```

Y el archivo **arch1.txt** contendrá ahora lo siguiente:

```
La vaca LOCA
VACAYENDO gente al baileee
siga la vaca
vacaciones en faMIlia...
es lila la vaca de MILKA
Señora VACaaa, señora VACAAA!!!
```

¿Qué hicimos en el programa?

Código	Efecto
# pruebaarch1.py	

vacaTxt=open('arch1.txt','r+')	Abre arch1.txt para lectura y escritura, con la codificación por defecto del **SO** y define nombre interno o alias vacaTxt
t=vacaTxt.readline()	Lee desde el archivo la primer línea (todos los caracteres hasta encontrar un \n y se guarda en t
print(t)	Muestra la línea
t=input('Ingresá un texto con vaca: ')	Pide al usuario un nuevo texto con la palabra vaca
while (t.lower()).count('vaca')==0:	Valida que el texto incluya al menos vez
t=input('Ingresá un texto con vaca: ')	vaca
vacaTxt.write(t+'\n')	Graba en el archivo la línea leída
vacaTxt.close()	Cierra el archivo
vacaTxt=open('arch1.txt')	Reabre el archivo, pero ahora sólo lectura
todas=vacaTxt.readlines()	Carga la lista todas con las líneas del archivo. Cada línea es un elemento
for linea in todas:	Muestra el contenido de la lista quitando
print(linea.strip('\n'))	\n para evitar doble interlineado, uno sale naturalmente por el print()
vacaTxt.close()	Cierra el archivo

Archivos Planos – Comma Separated Values (csv)

Mirá un ejemplo de cómo podés integrar datos de tres **Hojas de Cálculo** en un único archivo. Desde Excel[M] podés guardar una hoja de cálculo con formato **csv** (valores separados por coma). Esta clase de archivo tiene un formato interno estándar, se puede tratar como un archivo plano, cada dato en la línea está separado por**,** o **;**

Nota:
*El caracter de separación de campos en un archivo **csv** puede variar (**‘,’** o **’;’**) dependiendo de la Planilla que lo abre. Por ejemplo, en Excel[M] se usa **‘;’**, mientras que en algunas Planillas Open se emplea **‘,’**. Averiguar cuál usar es simple. Generas un archivo **csv** en tu Planilla y luego lo abres que un editor simple de texto y miras con cuál símbolo se separan los campos. Así sabrás qué separador indicar en tu programa.*

datos1.csv

Luciana	juárez	santiago del estero	95
Julián	manzur	santiago del estero	98
Mariano	álvarez	san luis	68
rogelio	linares	córdoba	88
Anabel	llanes	mendoza	83

datos2.csv

Jazmín	gerez	formosa	95
ana laura	jiménez	chaco	98

Roberto	luna	corrientes	68
Joaquín	ortiz alba	entre ríos	88
Antonio	peñate	río negro	83
Marcelo	villate	santa cruz	99
Mariano	villafáñez	santa cruz	98
Nicolás	luzuriaga	santa cruz	97
Agustín	méndez	formosa	77

datos3.csv

Andrés	gil gómez	caba	85
Andrea	paz	caba	99
Pablo	paz iraola	buenos aires	77
Paulina	estarda	la pampa	88
Paula	garcía	río negro	83
Lucía	gonzález	neuquén	99
Luciano	pinto	la rioja	81
Rossana	del olmo	chubut	84
Victoria	antón	salta	62

Nota:

Supondremos que cada una de ellas tiene datos relacionados con puntajes obtenidos por personas en un concurso
Nuestro programa va a guardar la información de todos en un archivo de tipo **.csv** (se podrá abrir desde block de notas, Excel[M], etc.); y la presentará ordenada de mayor a menor puntaje obtenido:

Nota:

Emplearemos separación de campos con ***';'***

Código	Efecto
#Genera Planilla Integrada de Datos	
completo=open('datosCompletos.csv','w')	Crea el archivo datosCompletos
b=[]	Crea la lista b vacía
dat=open('datos1.csv')	Abre datos1 sólo para lectura como dat
a=dat.readlines()	Carga en la lista a todas las líneas
dat.close()	Cierra archivo
for i in range(len(a)):	
a[i]=a[i].strip('\n')	Quita bajada de línea a cada línea
a[i]=a[i].split(';')	Arma tabla quitando ;
a[i][3]=int(a[i][3])	A la columna 3 la convierte a entero

b=b+a	Agrega la lista a a b
dat=open('datos2.csv')	Lo mismo que antes pero para datos2
a=dat.readlines()	
dat.close()	
for i in range(len(a)):	
a[i]=a[i].strip('\n')	
a[i]=a[i].split(';')	
a[i][3]=int(a[i][3])	
b=b+a	
dat=open('datos3.csv')	Lo mismo que antes pero para datos3
a=dat.readlines()	
dat.close()	
for i in range(len(a)):	
a[i]=a[i].strip('\n')	
a[i]=a[i].split(';')	
a[i][3]=int(a[i][3])	
b=b+a	
b.sort(reverse=True,key=lambda b:(b[3],b[2]))	Ordena la tabla b que tiene todas las filas de datos1, datos2 y datos3
for elemento in b:	
print(elemento)	Muestra las filas de b
for elemento in b:	
elemento[3]=str(elemento[3])	Para cada fila convierte columna 3 en texto
ele=';'.join(elemento)+'\n'	Arma string con separaciones de ; y agrega bajada de línea al final
completo.write(ele)	Graba la línea en el archivo completo
completo.close()	Cierra el archivo

Que producirá la siguiente salida en pantalla:

```
['marcelo','villate', 'santa cruz', 99]
['lucía', 'gonzález', 'neuquén', 99]
['andrea', 'paz', 'caba', 99]
['julián', 'manzur','santiago del estero', 98]
['mariano', 'villafáñez', 'santa cruz', 98]
['ana laura', 'jiménez', 'chaco', 98]
['nicolás', 'luzuriaga', 'santa cruz', 97]
['luciana', 'juárez', 'santiago del estero', 95]
['jazmín', 'gerez', 'formosa', 95]
['paulina', 'estarda', 'la pampa', 88] ['joaquín',
'ortiz alba', 'entre ríos', 88] ['rogelio ',
'linares', 'córdoba', 88]
['andrés', 'gil gómez', 'caba', 85]
['rossana', 'del olmo', 'chubut', 84]
['antonio', 'peñate', 'río negro', 83]
```

```
['paula', 'garcía', 'río negro', 83] ['anabel',
'llanes', 'mendoza', 83]
['luciano', 'pinto', 'la rioja', 81]
['agustín', 'méndez', 'formosa', 77]
['pablo', 'paz iraola', 'buenos aires', 77]
['mariano', 'álvarez', 'san luis', 68] ['roberto',
'luna', 'corrientes', 68] ['victoria', 'antón',
'salta', 62]
```

Y generará el siguiente archivo. Para verlo lo abriremos desde Excel:

datosCompletos.csv

Marcelo	villate	santa cruz	99
Lucía	gonzález	Neuquén	99
Andrea	paz	Caba	99
Julián	manzur	santiago del estero	98
Mariano	villafáñez	santa cruz	98
ana laura	jiménez	Chaco	98
Nicolás	luzuriaga	santa cruz	97
Luciana	juárez	santiago del estero	95
Jazmín	gerez	Formosa	95
Paulina	estarda	la pampa	88
Joaquín	ortiz alba	entre ríos	88
rogelio	linares	Córdoba	88
Andrés	gil gómez	Caba	85
Rossana	del olmo	Chubut	84
Antonio	peñate	río negro	83
Paula	garcía	río negro	83
Anabel	llanes	Mendoza	83
Luciano	pinto	la rioja	81
Agustín	méndez	Formosa	77
Pablo	paz iraola	buenos aires	77
Mariano	álvarez	san luis	68
Roberto	Luna	Corrientes	68
Victoria	Antón	Salta	62

Una mejora:

Como lo que se emplea para abrir un **archivo** en la función **open**() es una string que contiene el nombre, podemos editar esa string y automatizar, si cabe, la construcción del nombre del **archivo** a abrir.

- Integración con bucle de Carga

```
completo=open('datosCompletos.csv','w')
```

```
b=[]

j in range(1,4):
    dat=open('datos'+str(j)+'.csv')
    a=dat.readlines()
    dat.close()
for i in range(len(a)):
     a[i]=a[i].strip('\n')
   a[i]=a[i].split(';')
   a[i][3]=int(a[i][3])

   b=b+a
b.sort(reverse=True,key=lambda b:(b[3],b[2]))
for elemento in b:
   print(elemento)
for elemento in b:
   elemento[3]=str(elemento[3])
   ele=';'.join(elemento)+'\n'
   completo.write(ele)
completo.close()
```