Manejo de archivos de datos con `python`

Curso de Física Computacional

curso.fisica.comp@gmail.com

M. en C. Gustavo Contreras Mayén.

1. Los archivos de datos con `python`.

1.1. Consideraciones

Una tarea muy común y frecuente en el ámbito de la Física Computacional, será el manejo de archivos de datos para su proceso y análisis.

Como una primera tarea que hay que atender en el desarrollo de prácticas de programación bajo cualquier lenguaje, es la generación de un archivo de datos con una serie de características necesarias:

1. Una estructura de construcción, que permita la posterior lectura del archivo.
2. La codificación de los caracteres que conforman los datos, este punto es crítico si además de números, se incluyen o manejan cadenas de texto (el manejo de acentos y ciertos caracteres es crítico si no se considera este elemento)

1.2. Estructura del archivo.

Debemos de elegir anticipadamente el tipo de archivo que vamos a generar, ya que de esta forma, al momento de definir el código, se implementará la estructura del

archivo.

Quien reciba el archivo para su procesamiento, análisis, etc. debe de conocer el tipo de estructura que lleva. En el posible caso de que no avisemos esta información, el usuario del archivo, deberá de hacer una revisión preliminar del archivo para identificar la estrucutura. En el caso de que nosotros seamos los usuarios de un archivo de datos y no tengamos la informción, será nuestra tarea entonces, hacer la revisión de la estructura interna del archivo.

Existen diversos formatos para los archivos de texto, a continuación se describen cuatro de los más conocidos:

Tipo de archivo	**Descripción**
txt	Texto plano que representa sólo caracteres (o cadenas), excluye cualquier tipo de metadatos.
csv	Valores separados por coma (Comma-separated values), utiliza comas (u otro delimitador) para estructurar los datos almacenados, permitiendo guardarlos en un formato tipo tabla.
HMTL	Lenguaje de marcado de hipertexto (HyperText Markup Language) almacena los datos de manera estrucutrada, se utiliza comúnmente en páginas web.
JSON	Notación de objeto de JavaScript (JavaScript Object Notation) es un formato sencillo y eficiente, siendo un formato para almacenar y transferir datos.

Al momento en que compartan el archivo, se recomienda elaborar un archivo **README**, que contiene información acerca de otros archivos ya sea en un directorio o un archivo comprimido.

Es una forma de documentación de software, consiste en un archivo de texto plano llamado **READ.ME**, **README.TXT**, **README.md** (para un archivo markdown), **README.1ST** o simplemente **README**.

El nombre del archivo es generalmente escrito en mayúsculas. En los sistemas Unix-like, generalmente los nombres se escriben en minúscula, y esto hace que en un listado

de archivos salga primero el archivo **README**.

2. Creando un archivo de datos.

Con `python` será una tarea muy frecuente el generar un archivo de datos y compartirlo con otros usuarios.

Las instrucciones para crear el archivo son sencillas de implemantar en el código, toma en cuenta lo siguiente:

1. El archivo de datos se va a crear en la misma carpeta en donde se está ejecutando el archivo de `python`, es decir, el archivo con extensión `*.py`
2. Un archivo de datos requiere de la "apertura" de un espacio en memoria, para que con una instrucción, se "escriban" los datos en el archivo.
3. Una vez que se completa la tarea de escribir los datos, hay que cerrar el espacio de memoria.

2.1. El código con **python**.

Del ejercicio que se realizó para resolver el cálculo de valores a través de la interpolación de Newton:

```
1  from newtonPoli import coeficientes, evaluaPoli
2  import numpy as np
3
4  resultados = open('DatosNewton.dat','w')
5
6  xDatos = np.array([0.15,2.30,3.15,4.85,6.25,7.95])
7  yDatos = np.array([4.79867,4.49013,4.2243,3.47313,2.66674,1
       .51909])
8
9  a = coeficientes(xDatos, yDatos)
10
11 print ('{:^3} \t {:^7} \t {:^7} \t {:<11}'.format('x', '
       yInterpol', 'yExacta', 'Err relativo'))
```

```
12 print ('-'*55)
13
14
15 for x in np.arange(0.0, 8.1, 0.5):
16     y = evaluaPoli(a, xDatos, x)
17     yExacta = 4.8* np.cos(np.pi*x/20.0)
18     print ('{:1.1f} \t {:1.5f} \t {:1.5f} \t {:1.5E}'.format(
           x, y, yExacta, abs(yExacta - y)/yExacta*100))
19     resultados.write ('{:1.1f} \t {:1.5f} \t {:1.5f} \n'.
           format(x, y, yExacta))
20
21 resultados.close()
```

En la línea 4 del código, se indica la función **open**, que se requiere para "abrir" el espacio para el archivo, recibe como primer argumento el nombre del archivo y su extensión: *DatosNewton.dat*, nótese que la extensión que hemos dejado, es **.dat*, se puede utilizar libremente el tipo de extensión **.txt*, **.inf*, **.text*, **.data*, etc.

Lo relevante de la extensión es que el sistema operativo con el que trabajen, pueda distinguir que se trata de un archivo de datos en texto plano.

Como segundo argumento, se indica la *'w'* en la misma línea 4 del código, indica que se va a realizar la instrucción *write*, lo que permitirá escribir en el archivo. Como punto importante hay que señalar, que en caso de que el archivo *DatosNewton.dat* no exista, `python` creará un archivo nuevo, en caso de que ya exista, lo va a sobre-escribir.

En la línea 19 del código, se señala la instrucción que va a escribir los datos en el archivo: `resultados.write(argumento)`, en donde en el caso del ejercicio, el argumento consiste en una cadena de formato para tres columnas, cada una de ellas, separada mediante una tabulación (\t) y al final de la cadena de formato, se incluye un salto de línea (\n), que permitirá que en el siguiente ciclo del bucle, se escriba en otra línea del archivo, los valores que se calculan en esa iteración.

Una vez concluido el proceso de iteración, o el proceso donde se calcularon los valores del problema, se debe de cerrar el espacio del archivo, o de lo contrario tendremos un error. Para cerrar el objeto tipo *file*, se ocupa la función **nombreobjeto.close()**, como lo vemos en la línea de código 21.

Con estas instrucciones de manejo de un archivo, ya contamos en nuestro directorio de un archivo de datos que podremos utilizar posteriormente. Toma en cuenta que no se han incluido cabeceras de las columnas, en el ejercicio, las líneas 11 y 12, muestran en la terminal una cabecera con el nombre de la columna y una línea que separa a modo de tabla.

La pregunta natural es: ¿podremos incluir cabeceras o información adicional en nuestro archivo y que no interfiera con la recuperación de los datos?

3. Recuperando los datos.

Siguiendo con el ejercicio de la interpolación de Newton y contando ya con un archivo de datos: *DatosNewton.dat*, ahora nos aplicaremos para recuperar los datos y con ellos, generar una gráfica con la librería **matplotlib**.

3.1. Extraer los datos del archivo.

Revisemos el código que se requiere para primero obtener los datos del archivo y posteriomente usar la rutina ya conocida para graficar.

```
1  import matplotlib.pyplot as plt
2  import csv
3
4  x = []
5  yi = []
6  ye = []
7
8
9  with open('DatosNewton.dat', 'r') as csvfile:
10    plots = csv.reader(csvfile, delimiter = '\t')
11    for row in plots:
12       x.append(float(row[0]))
13       yi.append(float(row[1]))
14       ye.append(float(row[2]))
15
16
17 plt.plot(x, yi, label='Polinomio Newton', ls = 'dashed')
```

```
18 plt.plot(x, ye, label='Valor exacto', color = 'red')
19 plt.title('Grafica de los datos de un archivo')
20 plt.legend(loc = 1)
21 plt.show()
```

En la línea 2 se importa la librería *csv* que permitirá el manejo de un archivo de este tipo; como se mencionó en el apartado anterior, se requiere conocer el tipo de delimitador que se utilizó para generar el archivo.

Se requieren tres listas en donde vamos a almacenar los valores del archivo:

1. Un arreglo para representar la variable independiente, en este caso x.
2. Un arreglo para los valores de interpolación yi.
3. Un arreglo para los valores evaluados con la función exacta ye.

En la línea 9 del código mediante la instrucción **`with open`**, se declara un objeto de tipo `file`, al que se le da un alias: `csvfile`. Necesita al menos de dos argumentos: el nombre del archivo y el tipo de operación que se va a ejecutar, en nuestro ejemplo, vamos a usar `r`, que corresponde a la acción de leer los datos del archivo. Toma en cuenta que con la misma función **`open`** se pueden realizar las tareas de escribir un archivo y de leer los datos contenidos en el mismo.

Una vez definido el objeto `file`, se necesita recuperar los elementos contenidos, para ello usamos la función **`csv.reader`**, con dos argumentos: el nombre del objeto file y el delimitador con el que se guardó el archivo. El delimitador es un argumento necesario, como se mencionó previamente para los archivos de datos, puede usar un delimitador:

- Espacio en blanco.
- Tabulación.
- Coma.
- Guión.

De acuerdo al delimitador utilizado, se debe de indicar en el respectivo argumento, para nuestro ejemplo `delimiter = '\t'`.

3.2. Almacenando temporalmente los datos.

Posteriormente tenemos un ciclo **`for`**, mediante el cual se van agregando los elementos del archivo en cada lista, toma en cuenta que los datos que están en el archivo, a pesar de que la representación que vemos de ellos, correspondan a números, `python` los guardó como **`strings`**, siendo necesario la conversión a un tipo de dato **float** para su uso en la gráfica.

A partir de la línea 17 del código, se presenta la rutina de graficación ya conocida y manejada en el curso de Física Computacional.

El resultado obtenido es la siguiente gráfica:

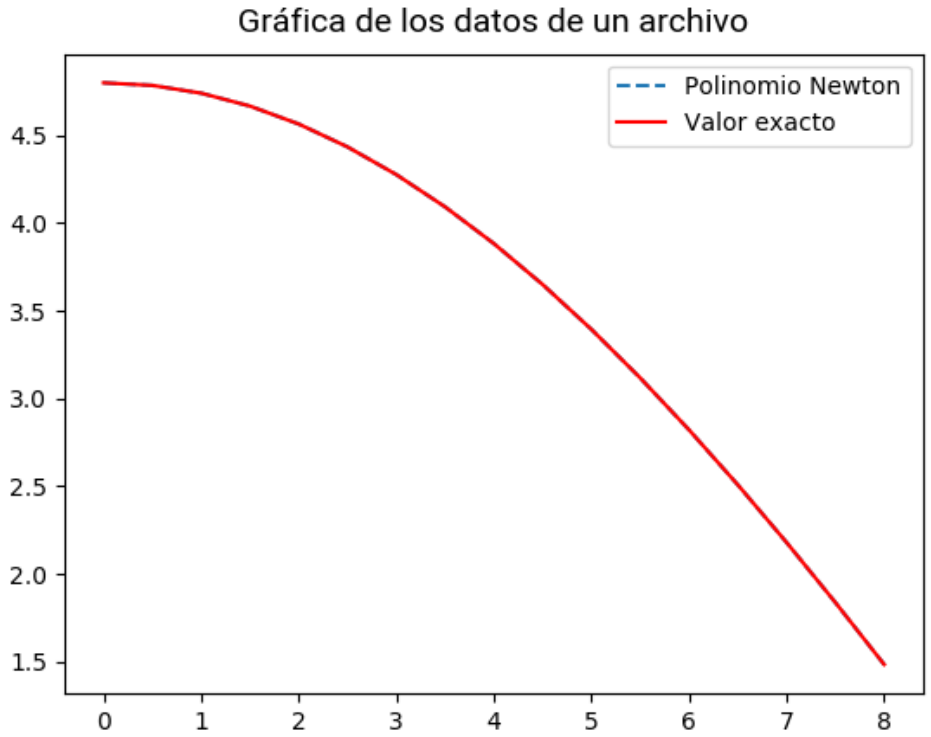

Figura 1: Gráfica generada a partir de los datos en un archivo.

Las opciones que hemos visto para el manejo de archivos son las básicas, podrás extender su uso e implementación cuando revises la respectiva documentación.

El potencial que se tiene con `python` para el manejo de arhivos es muy grande, ya que se pueden manejar archivos de datos y analizarlos sin necesidad de cambiar de formatos, por ejemplo, un archivo de datos en Excel (que en algunos casos suele ser un tipo de archivo aceptado para el ámbito científico) se puede operar con librerías especializadas como **`pandas`** que contiene funciones que operan directamente en los archivos.