3-data_io

October 22, 2020

0.1 Introducción

- **I/O** significa *input/output*.
- Escribir y leer datos desde archivos u otras fuentes, es fundamental en la programación, y más aún en programación científica.

0.2 Leer archivos

[1]:
```
%%file Salidas-data/inout.dat
Hola, desde el archivo
Este es un archivo de texto
Escrito en ASCII
```

```
Overwriting Salidas-data/inout.dat
```

Lee el archivo de una sola pasada

[2]:
```
archivo = open('Salidas-data/inout.dat')
print (archivo.read())
archivo.close()
```

```
Hola, desde el archivo
Este es un archivo de texto
Escrito en ASCII
```

Línea por línea

[3]:
```
archivo = open('Salidas-data/inout.dat')
print (archivo.readlines())
archivo.close()
```

```
['Hola, desde el archivo\n', 'Este es un archivo de texto\n', 'Escrito en
ASCII\n']
```

Otra manera

[4]:
```
for line in open('Salidas-data/inout.dat'):
    print (line.split())
```

```
['Hola,', 'desde', 'el', 'archivo']
['Este', 'es', 'un', 'archivo', 'de', 'texto']
['Escrito', 'en', 'ASCII']
```

0.3 Escribir archivos

`write()` es lo contrario a `read()` (¡Vaya sorpresa!)

[5]:
```
contents = open('Salidas-data/inout.dat').read() #Lo abrio, leyo y cerro solo
out = open('Salidas-data/my_output.dat', 'w')
out.write(contents.replace(' ', '_'))
out.close()
```

[7]:
```
!cat Salidas-data/my_output.dat
```

```
Hola,_desde_el_archivo
Este_es_un_archivo_de_texto
Escrito_en_ASCII
```

Ejercicio Cambia la segunda línea del archivo a *¿Cómo has estado?* ¿Qué sucede?

Ejercicio Escribe un archivo `CSV`. Calcula $y(x) = x^2 \cos x$ para los valores del $x \in 1..100$. En la primera columna guarda x y en la segunda $y(x)$.

[8]:
```
import math
archivo = open("Salidas-data/archivo.csv", "w")
for x in range(1,101):
    #archivo.write("{},{}\n".format(x,x**2*math.cos(x)))
    archivo.write(str(x)+","+str(x**2*math.cos(x))+"\n")
archivo.close()
!cat Salidas-data/archivo.csv
```

```
1,0.5403023058681398
2,-1.6645873461885696
3,-8.909932469404009
4,-10.458297933817791
5,7.091554636580656
6,34.566130319413176
7,36.94121046282193
8,-9.312002163751266
9,-73.80155121265884
10,-83.90715290764524
11,0.5355094565541451
12,121.51497005747886
13,153.35850606508316
14,26.800494768735387
15,-170.9297803932348
16,-245.16082696278647
17,-79.52220469691152
```

18,213.94261347108198
19,356.9223671653876
20,163.23282472535678
21,-241.54860375890235
22,-483.98103997500436
23,-281.8686677563673
24,244.32710822611025
25,619.501757414671
26,437.3174618941609
27,-212.96919156696657
28,-754.6829991898362
29,-629.1163824684493
30,138.82630489882564
31,879.0674058501546
32,854.2447211586665
33,-14.458377725911772
34,-980.9472376510037
35,-1107.0229512370956
36,-165.84094175711647
37,1047.8518371131752
38,1379.1263420042937
39,405.56390011946456
40,-1067.100898643619
41,-1659.7173255175524
42,-705.5740956394517
43,1026.4044945116368
44,1935.6966455419301
45,1063.7770273559026
46,-914.4885313761908
47,-2192.0690513544014
48,-1474.892558137036
49,721.7226975284726
50,2412.4150712302835
51,1930.3430659126484
52,-440.7270712715876
53,-2579.456346469842
54,-2418.267472628946
55,66.9334376924161
56,2675.698257818024
57,2923.6673208229104
58,400.9219756498281
59,-2684.130256178917
60,-3428.6867294945628
61,-960.3961873262932
62,2588.9615319718655
63,3913.0235323011398
64,1605.0472158394368
65,-2376.367521481277

```
66,-4354.464318189421
67,-2324.2686312550886
68,2035.2213360216922
69,4729.5315978577355
70,3103.264095122869
71,-1557.7835726851622
72,-5014.227049611806
73,-3923.1709954333664
74,940.3241638653379
75,5184.8508922017145
76,4761.337768477253
77,-183.65096313438238
78,-5218.874019302506
79,-5591.754678922401
80,-706.4783605699045
81,5095.836728043922
82,6385.632840562221
83,1719.0818727183264
84,-4798.2457848642625
85,-7112.121248521952
86,-2837.833698848291
87,4312.440280054147
88,7739.146708935045
89,4041.1123729829596
90,-3629.396290646278
91,-8234.356943946437
92,-5302.22580711311
93,2745.4408394438997
94,8566.14296389601
95,6589.816387978248
96,-1662.8470206666298
97,-8704.713171835658
98,-7868.444307779175
99,390.28444873315436
100,8623.18872287684
```

0.4 Numpy I/O

[9]:
```
%pylab inline
import numpy as np
import matplotlib.pyplot as plt
```

```
Populating the interactive namespace from numpy and matplotlib
```

- `NumPy` permite escribir y leer los arreglos a archivo de varias maneras, como **texto** o en **binario**.
- Si escribes a un archivo usando el modo de **texto**, el número π, se escribirá como 3.141592653589793. Algo que un humano puede leer (bajo ciertas condiciones, obvio), es

decir, una cadena de texto. El modo de texto, ocupa más espacio, la precisión se puede perder (no todos los dígitos se escribirán al disco), pero puede ser editada a mano. Si guardas un arreglo, sólo se pueden guardar arreglos bidimensionales.

- En cambio, si usas el modo **binario** para escribir a archivo, se escribirá como una cadena de 8 bytes que será idéntica a como se guarda en la memoria de la computadora. Sus únicas desventaja es que no puede ser editado a mano y que es dependiente de `NumPy` (no puede ser leído por otro programa, sin un convertidor).

0.5 Modo Texto

[11]:
```
arr = np.arange(10).reshape(2, 5)
np.savetxt('Salidas-data/test.out', arr, fmt='%.2e', header="My dataset")
!cat Salidas-data/test.out
```

```
# My dataset
0.00e+00 1.00e+00 2.00e+00 3.00e+00 4.00e+00
5.00e+00 6.00e+00 7.00e+00 8.00e+00 9.00e+00
```

[12]:
```
DataIn = np.loadtxt('Salidas-data/test.out')
print (DataIn.shape)
print (DataIn)
```

```
(2, 5)
[[0. 1. 2. 3. 4.]
 [5. 6. 7. 8. 9.]]
```

[13]:
```
print (DataIn[1,:])
```

```
[5. 6. 7. 8. 9.]
```

Leyendo archivos CSV

[14]:
```
%%file Salidas-data/input.csv
# Mis datos de ejemplo
    0.0,  1.1,  0.1
    2.0,  1.9,  0.2
    4.0,  3.2,  0.1
    6.0,  4.0,  0.3
    8.0,  5.9,  0.3
```

```
Overwriting Salidas-data/input.csv
```

[15]:
```
!cat Salidas-data/input.csv
```

```
# Mis datos de ejemplo
    0.0,  1.1,  0.1
    2.0,  1.9,  0.2
    4.0,  3.2,  0.1
```

```
   6.0,  4.0,  0.3
   8.0,  5.9,  0.3
```

[16]:
```
x, y, z = np.loadtxt('Salidas-data/input.csv',unpack=True, delimiter=',',␣
↪usecols=[0,1,2])
print (x,y,z)
```

```
[0. 2. 4. 6. 8.] [1.1 1.9 3.2 4.  5.9] [0.1 0.2 0.1 0.3 0.3]
```

0.6 Modo Binario

Para guardar datos binarios, `NumPy` provee los métodos `np.save` y `np.savez`. El primero sólo guarda un arreglo y el archivo tendrá la extensión `.npy`, mientras que el segundo se puede utilizar para guardar varios arreglos a la vez con una extensión `.npz`.

[17]:
```
arr2 = DataIn
#Lo guarda en binario
np.save('Salidas-data/test.npy', arr2)
# Lo leemos de nuevo
arr2n = np.load('Salidas-data/test.npy')
# Veamos si hay una diferencia...
print( 'Any differences?', np.any(arr2-arr2n))
```

```
Any differences? False
```

[19]:
```
np.savez('Salidas-data/test.npz', arr, arr2)
arrays = np.load('Salidas-data/test.npz')
arrays.files
```

[19]:
```
['arr_0', 'arr_1']
```

[20]:
```
np.savez('Salidas-data/test.npz', array1=arr, array2=arr2)
arrays = np.load('Salidas-data/test.npz')
arrays.files
```

[20]:
```
['array1', 'array2']
```

[21]:
```
print( 'First row of first array:', arrays['array1'][0])
# Este es una manera equivalente de obtener el primer elemento
print ('First row of first array:', arrays.f.array1[0])
```

```
First row of first array: [0 1 2 3 4]
First row of first array: [0 1 2 3 4]
```

Ejercicio: - Crea un arreglo bidimensional con 100 elementos flotantes al azar y guárdalos en formato de texto.

- Guárdalos también en formato binario. ¿Hay alguna diferencia entre ellos?

- Ahora crea un arreglo tridimensional con los elementos del 1 al 50 y guárdalos en formato binario ¿Qué pasa si los quieres guardar en formato de texto?