INDICE

Input ed output su file

dove sono?

Ogni volta che usiamo Python lo facciamo operando in una `cartella di lavoro`
Per sapere dove stiamo lavorando uso il modulo (package) os ed in particolare il modulo os.path:

In [1]:
```
import os
os.getcwd()
```

Out[1]:
```
'C:\\python\\corso\\lezioni'
```

Posso cambiare cartella specificando un percorso 'relativo' alla cartella di lavoro corrente

In [2]:
```
os.chdir('..')
print(os.path.abspath(os.curdir))
```

```
C:\python\corso
```

Oppure utilizzando un percorso assoluto:

In [3]:
```
os.chdir('C:/python/corso/lezioni')
print(os.path.abspath(os.curdir))
```

```
C:\python\corso\lezioni
```

La funzione open

documentazione di open in py3k-doc

La funzione open accetta molti parametri di cui due sono fondamentali:

- il riferimento al file da aprire
- uno o più caratteri che indicano la modalita' con cui il file e' aperto (vedi la tabella seguente)

La funzione restituisce il file aperto o segnala un errore nel caso non sia stato possibile aprirlo. Vedremo alcuni esempi nei paragrafi che seguono.

Character	Meaning
`'r'`	open for reading (default)
`'w'`	open for writing, truncating the file first
`'x'`	open for exclusive creation, failing if the file already exists
`'a'`	open for writing, appending to the end of the file if it exists
`'b'`	binary mode
`'t'`	text mode (default)
`'+'`	open a disk file for updating (reading and writing)
`'U'`	universal newlines mode (for backwards compatibility; should not be used in new code)

Scrittura di file di testo

apertura file in scrittura

Per poter scrivere un file di testo e' necessario prima aprirlo (o crearlo come in questo caso) in modalita' testuale e in scrittura:

In [13]:
```
#apro un file in scrittura specificandone nome 'mytxt.txt' e
#che voglio aprirlo in modalita' testuale ed in scrittura!
myfl = open('mytxt.txt','wt')
```

In [10]:
```
type(myfl),dir(myfl)
```

Out[10]:
```
(_io.TextIOWrapper,
 ['_CHUNK_SIZE',
  '__class__',
  '__delattr__',
  '__dict__',
  '__doc__',
  '__enter__',
  '__eq__',
  '__exit__',
  '__format__',
  '__ge__',
  '__getattribute__',
  '__getstate__',
  '__gt__',
  '__hash__',
  '__init__',
  '__iter__',
  '__le__',
  '__lt__',
  '__ne__',
  '__new__',
  '__next__',
  '__reduce__',
  '__reduce_ex__',
  '__repr__',
```

```
 '__setattr__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 '_checkClosed',
 '_checkReadable',
 '_checkSeekable',
 '_checkWritable',
 'buffer',
 'close',
 'closed',
 'detach',
 'encoding',
 'errors',
 'fileno',
 'flush',
 'isatty',
 'line_buffering',
 'mode',
 'name',
 'newlines',
 'read',
 'readable',
 'readline',
 'readlines',
 'seek',
 'seekable',
 'tell',
 'truncate',
 'writable',
 'write',
 'writelines'])
```

write

Per scrivere si usa principalmente il metodo write dell'oggetto file restituito da open.
Il metodo write accetta come argomenti delle stringhe e se voglio scrivere sul file un numero o altro devo convertirlo prima in stringa (siamo in modalita' testuale).

In [14]:
```
myfl.write('questa è una stringa\n')
myfl.write('sulla riga seguente c\'è un numero\n')
myfl.write(str(10)+'\n')
myfl.close()
```

writelines

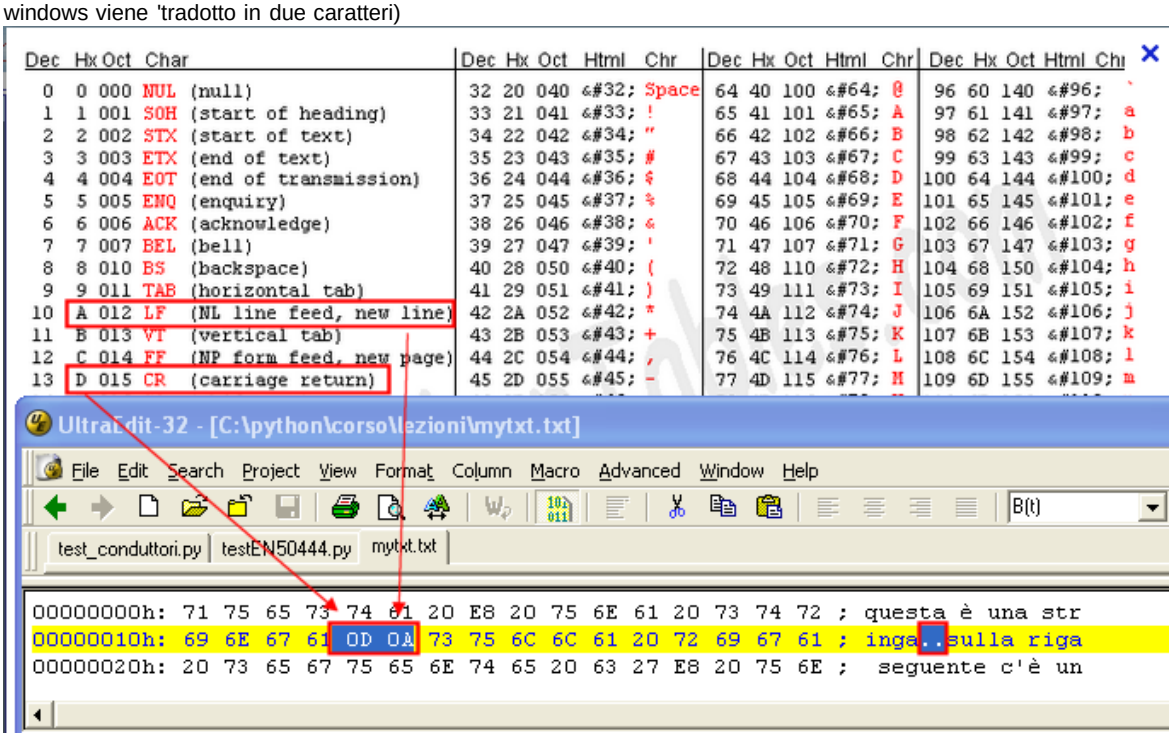
Per chi vuole scrivere un elenco di rige c'e' writelines:

In [15]:
```
rows=[]
for i in range(1,11):
    rows.append('stringa {0}\n'.format(i))#provare con e senza \n
#sovrascrivo il file
myfl = open('mytxt.txt','wt')

myfl.writelines(rows)

#myfl.writelines(rows)
#è equivalente a
#for row in rows:
#    myfl.write(row)

myfl.close()
```

Gestione del carattere 'fine riga':

Il carattere \n e' il fine riga e, in modalita' testuale, viene scritto sul file in modo diverso a seconda della piattaforma utilizzata (ad esempio in

windows viene 'tradotto in due caratteri)



sovrascrittura o append

- Attenzione, se riapro lo stesso file in scrittura indicando il modo 'w', il file viene sovrascritto.
- Per aggiungere nuovo contenuto in coda a quello esistente, riaprire il file indicando il modo 'a' (append) invece che 'w' (t non e' necessario perche' i file sono testuali di default)
- write restituisce il numero di caratteri scritti

In [16]:
```
myfl = open('mytxt.txt','a')# t non necessario
nc = myfl.write('\nun\'altra stringa con carattere con un carattere non ascii: è\n')
print('ho scritto altri {0} caratteri nell file'.format(nc))
myfl.close()
```

```
ho scritto altri 62 caratteri nell file
```

Lettura di file di testo

in un'unica stringa con read()

- la funzione open ha come modalità di default 'rt' ...
- per leggere tutto il file in un'unica stringa uso read()

In [17]:
```
#apro il file in lettura
myfl = open('mytxt.txt')#  (default 'rt')

#leggo tutto in una riga
mystr=myfl.read()
#chiudo il file
myfl.close()
#stampo la stringa
print(mystr)
mystr
```

```
stringa 1
stringa 2
stringa 3
stringa 4
```

```
stringa 5
stringa 6
stringa 7
stringa 8
stringa 9
stringa 10

un'altra stringa con carattere con un carattere non ascii: è
```

Out[17]:
```
"stringa 1\nstringa 2\nstringa 3\nstringa 4\nstringa 5\nstringa 6\nstringa 7\nstringa 8\nstringa
9\nstringa 10\n\nun'altra stringa con carattere con un carattere non ascii: è\n"
```

in una lista di stringhe con readlines()

In [20]:
```
#apro il file in lettura
myfl = open('mytxt.txt')
#leggo in una lista di stringhe
rows=myfl.readlines()
#chiudo il file
myfl.close()

#stampo la lista
print(rows)

#stampo le stringhe della lista
for row in rows:
    print(row, end="")#due \n uno nella riga, uno aggiunto da
              #print, usare row.strip() per eliminare il primo
              #o il parametro end di print per eliminare il secondo
```

```
['stringa 1\n', 'stringa 2\n', 'stringa 3\n', 'stringa 4\n', 'stringa 5\n', 'stringa 6\n', 'stringa
7\n', 'stringa 8\n', 'stringa 9\n', 'stringa 10\n', '\n', "un'altra stringa con carattere con un
carattere non ascii: è\n"]
stringa 1
stringa 2
stringa 3
stringa 4
stringa 5
stringa 6
stringa 7
stringa 8
stringa 9
stringa 10

un'altra stringa con carattere con un carattere non ascii: è
```

una riga alla volta con readline()

- readline restituisce una stringa vuota solo quando raggiunge la fine del file
- restituisce una stringa col carattere '\n' se incontra una riga vuota

In [21]:
```
#apro il file in lettura
myfl = open('mytxt.txt')
row = "segnaposto"
while row:
    row=myfl.readline()
    print(row,end='')
myfl.close()
```

```
stringa 1
stringa 2
stringa 3
stringa 4
stringa 5
stringa 6
```

```
stringa 7
stringa 8
stringa 9
stringa 10

un'altra stringa con carattere con un carattere non ascii: è
```

una riga alla volta iterando direttamente sul file

In [ ]:
```
#apro il file in lettura
myfl = open('mytxt.txt','r')
for row in myfl:
    print(row,end='')
myfl.close()
```

Gestione di file in context manager: with open(...) as ...

- Dopo aver aperto e usato un file è bene sempre chiuderlo (ad esempio per evitare di sprecare risorse di memoria)
- Fino ad ora negli esempi abbiamo sempre chiuso il file esplicitamente con close()
- E' possibile usare un costrutto particolare che permette di omettere la chiusura esplicita

with open(pathfile) as f: istrizioni in cui si usa il file al quale si è assegnato il nome f #il file f è chuiso automaticamente quando si esce dal blocco #di istruzioni indentato dopo la riga con with

Vediamo un esempio:

In [24]:
```
with open('mytxt.txt') as f:
    rows = f.readlines()
#il file viene chiuso

#uso le stringhe memorizzate nella lista rows
for i,row in enumerate(rows):
    print(row.rstrip()*i)
```

```
stringa 2
stringa 3stringa 3
stringa 4stringa 4stringa 4
stringa 5stringa 5stringa 5stringa 5
stringa 6stringa 6stringa 6stringa 6stringa 6
stringa 7stringa 7stringa 7stringa 7stringa 7stringa 7
stringa 8stringa 8stringa 8stringa 8stringa 8stringa 8stringa 8
stringa 9stringa 9stringa 9stringa 9stringa 9stringa 9stringa 9stringa 9
stringa 10stringa 10stringa 10stringa 10stringa 10stringa 10stringa 10stringa 10stringa 10

un'altra stringa con carattere con un carattere non ascii: èun'altra stringa con carattere con un
carattere non ascii: èun'altra stringa con carattere con un carattere non ascii: èun'altra stringa con
carattere con un carattere non ascii: èun'altra stringa con carattere con un carattere non ascii:
èun'altra stringa con carattere con un carattere non ascii: èun'altra stringa con carattere con un
carattere non ascii: èun'altra stringa con carattere con un carattere non ascii: èun'altra stringa con
carattere con un carattere non ascii: èun'altra stringa con carattere con un carattere non ascii:
èun'altra stringa con carattere con un carattere non ascii: è
```

Esempio in aula: scrittura e lettura di un file csv (comma separated values)

Stabiliamo le specifiche del file:

```
<li>riga delle intestazioni delle ncols colonne 'intestazione-01,....intestazione-ncol'</li>
<li>seguono un numero nrows indefinito di righe ciascuna con ncol colonne (nrows, ncols
indefiniti in lettura)</li>
<li>nella iesima riga e jesima colonna dati c'è l'intero j*10**(i) dove i e j partono da
zero</li>
```

Il file da scrivere e leggere avra cioé la seguente struttura

intestaz-01,intestaz-02,intestaz-03,intestaz-04,intestaz-05,.... 0, 1, 2, 3, 4, 5, ... 0, 10, 20, 30, 40, 50, ... 0, 100, 200, 300, 400, 500, ...

Scrittura del csv (v1)

- prima creo i dati da scrivere e li immagazzino in delle liste
- poi li scrivo sul file

In particolare:

- intestazioni è una lista di stringhe ciascuna delle quali rappresenta l'intestazione di una colonna
- righe è una lista di liste; ogni lista interna rappresenta una riga,che contiene le stringhe relative ai campi dati della riga stessa.

In [1]:
```
ncols = 10
nrows = 10
#....
intestazioni = ['intestaz-{0:02d}'.format(i) for i in range (1,ncols+1)]
righe =[  [str(icol*10**(irow)) for icol in range(ncols)] for irow in range(nrows)]
with open('provacsv.csv','wt') as fl:
    fl.write(','.join(intestazioni)+'\n')
    for colonne in righe:
        fl.write(','.join(colonne)+'\n')
```

Scrittura del csv (v2)

In [2]:
```
#versione -2
#scrivo una riga per volta generando contestualmente i dati
with open('provacsv2.csv','wt') as fl:
    for j in range(ncols):
        fl.write('Intestazione-{0:02d},'.format(j+1))
    fl.write('\n')
    for i in range(nrows):
        for j in range(ncols):
            fl.write(str(j*(10**i))+',')
        fl.write('\n')
```

Lettura del csv (v1)

In [3]:
```
#carico i dati scritti nel file tutti in una botta
with open('provacsv.csv','rt') as fl:
    rows=fl.readlines()

intestazioni = [i.rstrip() for i in rows[0].split(',')]
dati = []
for row in rows[1:]:
    datiriga = [ int(i) for i in row.split(',')]
    dati.append(datiriga)
```

In [4]:
```
#verifica lettura v1
print(intestazioni)
for r in dati:
    print(r)
```

```
['intestaz-01', 'intestaz-02', 'intestaz-03', 'intestaz-04', 'intestaz-05', 'intestaz-06', 'intestaz-
07', 'intestaz-08', 'intestaz-09', 'intestaz-10']
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[0, 10, 20, 30, 40, 50, 60, 70, 80, 90]
[0, 100, 200, 300, 400, 500, 600, 700, 800, 900]
[0, 1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000]
[0, 10000, 20000, 30000, 40000, 50000, 60000, 70000, 80000, 90000]
[0, 100000, 200000, 300000, 400000, 500000, 600000, 700000, 800000, 900000]
[0, 1000000, 2000000, 3000000, 4000000, 5000000, 6000000, 7000000, 8000000, 9000000]
```

```
[0, 10000000, 20000000, 30000000, 40000000, 50000000, 60000000, 70000000, 80000000, 90000000]
[0, 100000000, 200000000, 300000000, 400000000, 500000000, 600000000, 700000000, 800000000, 900000000]
[0, 1000000000, 2000000000, 3000000000, 4000000000, 5000000000, 6000000000, 7000000000, 8000000000,
9000000000]
```

Lettura del csv (v2)

In [5]:
```
dati2 = []
with open('provacsv2.csv','rt') as fl:
    for i,row in enumerate(fl):
        if i==0:
            intestazioni2 = [i.rstrip() for i in rows[0].split(',')]
        else:
            dati2.append([int(i) for i in row.split(',') if len(i.rstrip())>0])
```

In [6]:
```
#verifica lettura v2

print(intestazioni2)
for r in dati2:
    print(r)
```

```
['intestaz-01', 'intestaz-02', 'intestaz-03', 'intestaz-04', 'intestaz-05', 'intestaz-06', 'intestaz-
07', 'intestaz-08', 'intestaz-09', 'intestaz-10']
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[0, 10, 20, 30, 40, 50, 60, 70, 80, 90]
[0, 100, 200, 300, 400, 500, 600, 700, 800, 900]
[0, 1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000]
[0, 10000, 20000, 30000, 40000, 50000, 60000, 70000, 80000, 90000]
[0, 100000, 200000, 300000, 400000, 500000, 600000, 700000, 800000, 900000]
[0, 1000000, 2000000, 3000000, 4000000, 5000000, 6000000, 7000000, 8000000, 9000000]
[0, 10000000, 20000000, 30000000, 40000000, 50000000, 60000000, 70000000, 80000000, 90000000]
[0, 100000000, 200000000, 300000000, 400000000, 500000000, 600000000, 700000000, 800000000, 900000000]
[0, 1000000000, 2000000000, 3000000000, 4000000000, 5000000000, 6000000000, 7000000000, 8000000000,
9000000000]
```

Lettura e scrittura contemporanea di file di testo

Per leggere e contemporaneamente scrivere su un file si usano le modalita' (analoghe alla funzione fopen in C):

- r+: Apre il file per la lettura e la scrittura. Il puntatore viene posizionato all'inizio del file.
- w+: Apre il file per la lettura e la scrittura. Il file viene creato se non esiste, altrimenti viene troncato. Il puntatore viene posizionato all'inizio del file.
- a+: Apre il file per la lettura e la scrittura. Il file viene creato se non esiste. Il puntatore e' posizionato alla fine del file. Scritture successive con write finiscono sempre alla fine corrente del file, a prescindere da qualsiasi chiamata a seek (riposizionamento) o simili.

Vediamo un esempio.

Scriviamo un file di testo:

In [7]:
```
# ATTENZIONE se uso un file assegnandogli
# lo stesso nome (myfl)
# che utilizzo in un successivo costrutto with (anche se riferito ad un diverso file su disco)
# dopo il blocco with vengono chiusi entrambi i  files

#prove effettuate in aula
#myfl = open('mytxt2.txt','wt')
#myfl.write('ciao')

with open('mytxt.txt','wt') as myfl:
    for i in range(1,10):
        s='stringa {0}\n'.format(i)
        myfl.write(s)
        print(s, end="")

#LE SEGUENTI ISTRUZIONI GENERANO UN ERRORE
```

```
#myfl.write('ciao ciao')
#myfl.close()
```

```
stringa 1
stringa 2
stringa 3
stringa 4
stringa 5
stringa 6
stringa 7
stringa 8
stringa 9
```

Adesso leggiamo il file appena creato e contestualmente lo modifichiamo.
Per farlo, oltre alle funzioni già viste usiamo:

- seek
- tell

In [35]:
```
#AGGIUNGO UNA RIGA ALL'INIZIO
#apro il file in lettura + scrittura, scrivo all'inizio
myfl = open('mytxt.txt','r+')
myfl.write('stringa x\n')

#AGGIUNGO UNA RIGA ALLA FINE
#torno all'inizio del file
myfl.seek(0)
rows1 = myfl.readlines()
myfl.write('riga aggiunta alla fine!!!')

#MODIFICO LA TERZA RIGA
#torno all'inizio del file
myfl.seek(0)
#leggo due righe e le stampo
print('ho letto la riga: ',myfl.readline(),end="")
print('ho letto la riga: ',myfl.readline(),end="")
#devo spostare il puntatore esplicitamente
#altrimenti il successivo write scrive alla fine
myfl.seek(myfl.tell(),0)
#modifico la terza riga
myfl.write('stringa y\n')

#RILEGGO PER INTERO IL FILE MODIFICATO
#torno all'inizio del file altrimenti non leggo le righe aggiunte
myfl.seek(0)
rows2 = myfl.readlines()
myfl.close()


print('-----file modificato-----')
for row in rows2:
    print(row,end="")
print('\n-----fine file modificato-----')
```

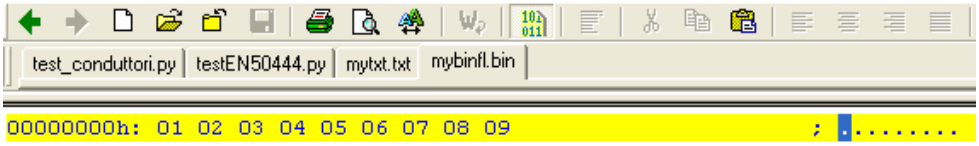
```
ho letto la riga:  stringa x
ho letto la riga:  stringa 2
-----file modificato-----
stringa x
stringa 2
stringa y
stringa 4
stringa 5
stringa 6
stringa 7
stringa 8
stringa 9
riga aggiunta alla fine!!!
-----fine file modificato-----
```

Lettura e scrittura di file binari

Scrittura di un file binario

- per aprire un file binario in scrittura uso open e mode 'bw'
- la funzione write si aspetta dei bytes e non delle stringhe

In [36]:
```
with open('mybinfl.bin','wb') as myflbin:
    myflbin.write(bytes(range(1,10)))
```



Lettura di un file binario

Tutto il file con read()

nota bene i flag rb passati a open

In [38]:
```
with open('mybinfl.bin','rb') as myflbin:
    #leggo tutto il contenuto in un'unica stringa di bytes
    data=myflbin.read()
    print(data)
```

```
b'\x01\x02\x03\x04\x05\x06\x07\x08\t'
```

Per blocchi di bytes con read(n)

In [39]:
```
with open('mybinfl.bin','rb') as myflbin:
    #leggo il primo byte
    b0=myflbin.read(1)
    #mi sposto al quarto byte
    myflbin.seek(4,0)
    #leggo i due bytes successivi al quarto
    b45=myflbin.read(2)
    print(b0)
    print(b45)
```

```
b'\x01'
b'\x05\x06'
```

Lettura e scrittura in file binario + conversione da/a bytes a/da altri tipi di dati (approccio 1: conversione di tipo)

esempio di scrittura (conversione da tipi di Python a Bytes)

In [40]:
```
#scrivo un intero, una stringa, su un file binario,
#dopo averli convertiti in bytes
#con un float e' + laborioso ed e' meglio usare l'approccio 2
i = 2
s = 'ciao'
print('Scrivo su un file (binario) :\n',i,s)
with open('mybinfl.bin','wb') as myflbin:
    myflbin.write(bytes([i]))
    myflbin.write(bytes(s,encoding='ascii'))
```

```
Scrivo su un file (binario) :
 2 ciao
```

esempio di lettura (conversione da Bytes a tipi di Python)

In [43]:
```
#leggo il file
with open('mybinfl.bin','rb') as myflbin:
    data=myflbin.read()

print('Il contenuto del file (in bytes) è:\n',data)

#decodifico il contenuto
ii=int(data[0])
ss=str(data[1:],encoding='ascii')
print('Il contenuto del file decodificato è:\n',ii,ss)
```

```
Il contenuto del file (in bytes) è:
 b'\x02ciao'
Il contenuto del file decodificato è:
 2 ciao
```

Lettura e scrittura in file binario + conversione da/a bytes a/da altri tipi di dati (approccio 2: uso il modulo struct)

Uso il modulo standard struct
Questo modulo effettua conversioni tra vari tipi di dati in Python e i bytes di Python. Cio' puo' essere utile nel trattamento dei dati binari memorizzati in file. Utilizza stringhe di formato come descrizioni compatte del layout delle strutture dati e la conversione a / da valori Python.

Le principali funzioni del modulo struct sono:

- pack: trasformazione tipi vari -> Bytes
- unpack:trasformazione Bytes -> tipi vari

Segue la tabella dei caratteri che compongono la stringa di formato usata in pack ed unpack:

Format	C Type	Python type	Standard size	Notes
x	pad byte	no value		
c	char	bytes of length 1	1	
b	signed char	integer	1	(1),(3)
B	unsigned char	integer	1	(3)
?	_Bool	bool	1	(1)
h	short	integer	2	(3)
H	unsigned short	integer	2	(3)
i	int	integer	4	(3)
I	unsigned int	integer	4	(3)
l	long	integer	4	(3)
L	unsigned long	integer	4	(3)
q	long long	integer	8	(2), (3)
Q	unsigned long long	integer	8	(2), (3)
n	ssize_t	integer		(4)
N	size_t	integer		(4)
f	float	float	4	(5)
d	double	float	8	(5)
s	char[]	bytes		
p	char[]	bytes		
P	void *	integer		(6)

esempio di scrittura (uso struct.pack)

In [52]:
```
fl=open('mybinfl.bin','wb')
fl.write(bytes([]))
fl.close()
```

In [54]:
```
import struct
#scrivo un intero, una stringa, su un file binario,
#dopo averli convertiti in bytes
#con un float e' molto + laborioso ed e' meglio usare l'approccio 2
i = 2
s = 'ciao'
f = 12.456


print('Scrivo su un file (binario) :\n',i,s,f)
```

```
with open('mybinfl.bin','wb') as myflbin:
    #trasformo i s e f in Bytes
    dataw = struct.pack('@i4sd',i,s.encode('ascii'),f)
    print('in Bytes:')
    print(dataw)
    myflbin.write(dataw)
```

```
Scrivo su un file (binario) :
 2 ciao 12.456
in Bytes:
b'\x02\x00\x00\x00ciao\xb6\xf3\xfd\xd4x\xe9(@'
```

esempio di lettura (uso struct.unpack)

In [56]:
```
#leggo il file
with open('mybinfl.bin','rb') as myflbin:
    datar=myflbin.read()
print('Il contenuto del file (in bytes) è:\n',datar)

#decodifico il contenuto
ii,ss,ff=struct.unpack('@i4sd',datar)
print('Il contenuto del file decodificato è:\n',ii,ss.decode('ascii'),ff)
```

```
Il contenuto del file (in bytes) è:
 b'\x02\x00\x00\x00ciao\xb6\xf3\xfd\xd4x\xe9(@'
Il contenuto del file decodificato è:
 2 ciao 12.456
```

memory IO

Tutte le operazioni di scrittura e lettura di file viste fino ad ora possono essere effettuate anche in memoria volatile usando le classi StringIO (modalita' testuale) e BytesIO (modalita' binaria).
In pratica le due classi Vedi anche in-memory-streams

scrivo in memoria

In [8]:
```
import io
#creo un oggetto 'file-like' testuale in memoria
output = io.StringIO()
#scrivo con write come su un file di testo
output.write('Prima riga\n')
#scrivo con print specificando il file
print('Seconda riga.\n', file=output)
```

leggo dalla memoria

In [9]:
```
#Recupero l'intero contenuto del file
#dopo aver riportato il puntatore all'inizio
output.seek(0)
contents = output.read()
# chiudo l'oggetto e cancello i dati dalla memoria
# dopo la chiusura una chiamata a read o a write provocherebbe un errore

output.close()

print(contents)
```

```
Prima riga
Seconda riga.
```

link sparsi

- reading-and-writing-files in py3k doc
- funzione open

- modulo io