Programmazione in Python

File

Dario Pescini - Mirko Cesarini

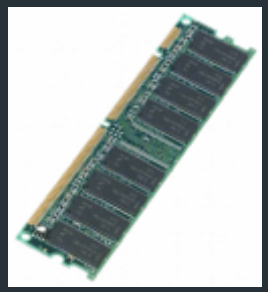
Università degli Studi di Milano-Bicocca

Dipartimento di Statistica e Metodi Quantitativi

nome.cognome@unimib.it

Hardware: immagazzinamento dati

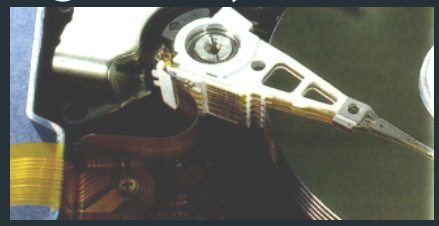
- Memoria centrale (o RAM)



Caratteristiche:

  - Accesso ai dati veloce
  - Capienza limitata
  - **Memoria principale di lavoro**
  - Volatile (spegnendo il computer, si perdono tutte le informazioni)

- Memoria di massa (disco rigido, ...)



Caratteristiche:

  - Accesso ai dati lento
  - Capienza ampia
  - **Memoria secondaria di lavoro**
  - Non volatile (spegnendo il computer, NON si perdono le informazioni)

Memoria e biblioteche: paragone

Scenario: una persona cerca dei libri negli scaffali, poi li porta con se su una scrivania e inizia a consultarli

- Sulla scrivania
  - i libri presenti possono essere consultati velocemente
  - è possibile appoggiare solo un numero limitato di libri
  - se è necessario un libro non presente nella scrivania, occorre andarlo a cercare negli scaffali
  - se, dopo aver cercato nuovi libri, nella scrivania non c'è più spazio, occorre riporre qualcosa negli scaffali
- Negli scaffali
  - capienza molto maggiore della scrivania
  - tuttavia recuperare un libro richiede tempo

Analogia tra memorie e biblioteca

- Memoria centrale (scrivania)
  - Variabili python: ospitate nella memoria centrale
  - Le variabili ereditano vantaggi e svantaggi della memoria centrale
    - Velocità
    - Capienza limitata
    - Volatilità (al mancare della corrente si perde tutto)
- Memoria di massa (scaffali)
  - I dati sono organizzati sotto forma di file e directory
  - File: contenitore di dati (equivalente al libro)
  - Directory: contenitore di file (equivalente allo scaffale della biblioteca)

File

- Persistenza: se devo spegnere il computer, come faccio a non perdere i risultati ottenuti?
  - Salvo i dati nella memoria di massa
- Efficienza: se devo processare una quantità di informazioni le cui dimensioni eccedono la capienza della memoria centrale di un computer?
  - memorizzo i dati nella memoria di massa
  - carico in memoria centrale ed elaboro un sottoinsieme di dati per volta

Servizio prestiti di una biblioteca o l'interazione con il sistema operativo

- Prestito di un libro:
  - si richiede il prestito di un libro indicandone gli estremi
  - Il servizio effettua dei controlli: Il libro esiste? L'utente dispone dei permessi per il prestito? Il libro è disponibile o qualcun altro lo sta utilizzando?
  - Il servizio consente o nega l'accesso al libro
  - Il libro viene eventualmente consegnato
- Restituzione di un libro:
  - si riconsegna il libro alla biblioteca
  - il servizio chiude il prestito.

Interazione con il Sistema Operativo: apertura e chiusura

- Apertura di un file:
  - si richiede al SO l'accesso ad un file.
  - Il SO effettua dei controlli: Il file esiste? L'utente dispone dei permessi di accesso? L'accesso è libero o qualcun altro lo sta utilizzando?
  - Il SO consente o nega l'accesso al file.
  - É necessario indicare quali operazioni saranno svolte (solo lettura, scrittura, ...)
- Chiusura di un file:
  - si informa il sistema operativo che il programma non ha più bisogno di leggere e/o scrivere sul file
  - (per ottimizzare le prestazioni) le operazioni di lettura e scrittura dei dati non vengono eseguite appena richieste: il SO completa le operazioni in sospeso.
  - il SO chiude il l'accesso al file.

Per leggere un file, è necessario aprire il flusso verso di esso.

```
nomeOggettoFile = open(nomeFile, modo)
```

Apertura di un file `open()`

- `nomeOggettoFile` è l'oggetto restituito dal metodo e che permette di interagire con il file
- `nomeFile` è la stringa contenente il nome del file che, se presente nella directory corrente, viene aperto
- `modo` è una delle seguenti stringhe e descrive come si interagisce con il file
  - `'r'` viene aperto in sola lettura
  - `'w'` viene aperto in sola (sovra)scrittura
  - `'a'` viene aperto in sola scrittura e i nuovi dati vengo aggiunti alla fine

Modalità di apertura di un file

- 'r' (read) - sola lettura:
  significa che vogliamo aprire un file solo per leggerne il contenuto (scrittura non possibile).
  - Alcuni sistemi operativi permettono di aprire in lettura un file a più utenti contemporaneamente.
  - Se si tenta di scrivere su un file aperto in sola lettura, viene sollevato un errore.
- 'w' (write) - scrittura:
  si desidera aprire un nuovo file per potervi scrivere dati
  - se il file non esiste l'apertura in scrittura creerà un nuovo file vuoto
  - Se il file esiste, il contenuto preesistente sarà eliminato
- 'a' (append) - aggiunta:
  si desidera aprire un file per aggiungere in coda dati
  - se il file non esiste l'apertura in aggiunta creerà un nuovo file vuoto
  - Se il file esiste, i nuovi dati saranno accodati ai dati già esistenti.
- ...'r+', 'b'

- Se cerchiamo di aprire in lettura un file che non esiste otteniamo un errore:

```
h = open("test.cat","r")
```

```
IOError: [Errno 2] No such file or directory: 'test.cat'
```

Terminato il flusso verso il file è necessario chiuderlo

```
nomeOggettoFile.close()
```

chiusura di un file `close()`

- `nomeOggettoFile` è l'oggetto file che deve essere chiuso
- in questo modo le risorse occupate da `nomeOggettoFile` vengono liberate
- non è più possibile interagire con il file se non riaprendolo

Elaborazione di un file

Una volta ottenuto il flusso verso il file, è possibile elaborarlo.

Le operazioni coinvolte nell'elaborazione sono:

- il posizionamento
- la lettura
- la scrittura

La modalità di accesso al file ne determina anche l'elaborazione

Il file come un libro: accesso al contenuto sequenziale o diretto

É possibile leggere un libro in due modalità:

- una pagina per volta dall'inizio alla fine
- sfruttando l'indice per spostarsi alle pagine d'interesse

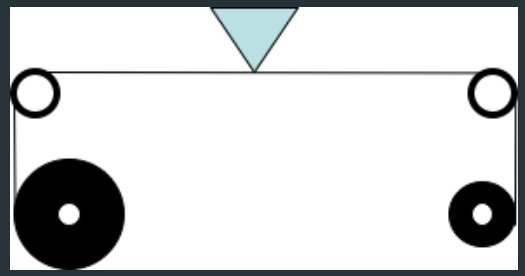
Poichè i file sono sequenze indicizzate di elementi, è possibile utilizzare queste due modalità di accesso anche per un file:

- sequenziale: i dati vengono caricati (letti) in successione
- diretto (o random): i dati vengono letti tramite un indice

Istruzioni per l'elaborazione dei file

- Scrittura
  - `write`
- Lettura
  - `read`
  - `readline`
  - `readlines`
- Posizionamento
  - `seek`
  - `tell`

Modello *nastro magnetico*



- Una testina di lettura/scrittura
- La testina scorre su un "nastro" formato da tante caselle contigue
  - ogni casella rappresenta un byte
  - le caselle sono numerate sequenzialmente
    - 0 per l'inizio del nastro
    - fino alla fine del file
  - Operazioni di lettura/scrittura: il nastro scorre, la testina legge/scrive caselle contigue
  - Al termine di ogni operazione di lettura/scrittura, la testina è **posizionata** all'inizio della casella successiva sul nastro (pronta per una nuova operazione)
  - Il nastro può essere fatto scorrere velocemente, senza dover leggere quello che c'è in mezzo (sfruttando l'indice)

File: note sul posizionamento

Quando si apre un file in:

- sola lettura (r), la testina viene posizionata all'inizio del file
- sola scrittura (w), la testina viene posizionata all'inizio del file
- modalità "append" (a), la testina viene posizionata alla fine del file

É sempre possibile effettuare dei salti, sia avanti sia indietro

Scrivere dati nel file

```
nomeOggettoFile.write(stringa)
```

Scrittura dati `write()`

- `stringa` è necessario convertire in stringa quanto deve essere scritto nel file

Esempio

```
f = open("c:\\temp\\test.txt","w")
f.write("Adesso")
f.write("Scriviamo qualcosa")
f.close()
```

- Apertura di un file
- Per scrivere dati nel file occorre utilizzare il metodo write
  `f.write()`
- Modello nastro magnetico: i due comandi write scrivono le 2 frasi una di seguito all'altra
- La chiusura del file avvisa il sistema operativo che la scrittura è conclusa ed il file è disponibile per altri scopi.

Leggere dati dal file

```
stringa = nomeOggettoFile.read()
```

lettura dei dati `read()`

- `stringa` è la variabile di tipo stringa che immagazzina l'intero contenuto del file
- `read(size)` tramite il parametro `size` si può limitare la dimensione dei dati in lettura

Esempio

```
g = open("test.txt","r")
Testo = g.read()
print ( Testo )
```

- Questa volta la modalità di apertura è "r":
- Il metodo `read` legge dati da un file. Senza argomenti legge l'intero contenuto del file:

```
AdessoScriviamo qualcosa
```

- Non c'è spazio tra *Adesso* e *Scriviamo* perché le due frasi sono state scritte una di seguito all'altra

```
g = open("test.txt","r")
print (g.read(5))
```

```
Adess
```

- `read` accetta anche un argomento che specifica quanti caratteri leggere

- In un'operazione di lettura, `read` restituisce solamente i caratteri effettivamente presenti,
  - Es: si richiede di leggere 5 caratteri, ma dopo averne letti 3, si incontra la fine del file, in qs caso read restituisce solamente i 3 caratteri letti
  - Se si cerca di leggere qualcosa dopo aver raggiunto la fine del file, `read` restituisce una stringa vuota

Accesso Sequenziale

I dati vengono letti in sequenza un frammento dopo l'altro:

```
stream = open('nomeFile', 'r')
data = f.read(100)
while data != '':
    data = stream.read(100)
    print data
stream.close()
```

Accesso Diretto

É possibile variare il punto in cui verranno letti/scritti i prossimi dati.

```
nomeOggettoFile.seek(posizione)
```

spostamento posizione `seek()`

- `seek()` sposta il punto in cui iniziano le oprazioni di lettura/scrittura del file
- `posizione` è la posizione (indice) del file in cui ci si vuole spostare

```
posizione = nomeOggettoFile.tell()
```

posizione attuale `tell()`

- `posizione` è la posizione (indice) corrente nel file

Esempio

```
f = open('test.txt', 'w+')
for i in range(4):
    stringa = 'il contatore vale ' + str(i) + '\n'
    f.write(stringa)
print('|' + f.read() + '|')
f.close()
```

Output:

Esempio

```
f = open('test.txt', 'w+')
for i in range(4):
    stringa = 'il contatore vale ' + str(i) + '\n'
    f.write(stringa)
print('|' + f.read() + '|')
f.close()
```

Output:

||

Esempio

```
f = open('test.txt', 'w+')
for i in range(4):
    stringa = 'il contatore vale ' + str(i) + '\n'
    f.write(stringa)
f.seek(0)
print('|' + f.read() + '|')
f.close()
```

Output:

Esempio

```
f = open('test.txt', 'w+')
for i in range(4):
    stringa = 'il contatore vale ' + str(i) + '\n'
    f.write(stringa)
f.seek(0)
print('|' + f.read() + '|')
f.close()
```

Output:

```
|il contatore vale 0
il contatore vale 1
il contatore vale 2
il contatore vale 3
|
```

Esempio

```
f = open('test.txt', 'w+')
for i in range(4):
    f.seek(0)
    stringa = 'il contatore vale ' + str(i) + '\n'
    f.write(stringa)
f.seek(0)
print('|' + f.read() + '|')
f.close()
```

Output:

Esempio

```
f = open('test.txt', 'w+')
for i in range(4):
    f.seek(0)
    stringa = 'il contatore vale ' + str(i) + '\n'
    f.write(stringa)
f.seek(0)
print('|' + f.read() + '|')
f.close()
```

Output:

```
|il contatore vale 3
|
```

File di testo

- Un file di testo è un file che contiene caratteri stampabili e spazi bianchi, organizzati in linee separate da caratteri di ritorno (`\n`) a capo(`\r`).
- Per esempio il testo seguente,

  ```
  Nel mezzo del cammin di nostra vita
  mi ritrovai per una selva oscura,
  ché la diritta via era smarrita.
  ```

- memorizzato in un file, apparirebbe in questo modo: Nel_mezzo_del_cammin_di_nostra_vita\nmi_ritrovai_per_una_selva_oscura,\nché_la_diritta_via_era_smarrita.\n
- Nota: `\n` significa *a capo*, `_` significa *spazio*

I/O di file di testo

- Ci occuperemo principalmente di file di testo
- Creazione di un file di testo composto da tre righe separate da caratteri di *a capo*:

```
f = open("test.txt","w")
f.write("linea uno\nlinea due\nlinea tre\n")
f.close()
```

- Il metodo readline legge tutti i caratteri, dalla posizione corrente fino al prossimo ritorno a capo:

```
f = open("test.txt","r")
print ( f.readline() )
```

```
linea uno
```

Lettura: una riga alla volta

```
stringa = nomeOggettoFile.readline()
```

lettura dei dati `readline()`

- `stringa` è la variabile che contiene il contenuto di una riga
- il puntatore nel file viene posizionato all'inizio della riga successiva

Lettura: tutte le righe in una sola volta

```
lista = nomeOggettoFile.readlines()
```

lettura dei dati `readlines()`

- `lista` ha per elementi una stringa per ogni riga del file

Fine file

- Quando il file è stato letto tutto, readline restituisce una stringa vuota:

```
print ( f.readline() )
```

← La stringa vuota

- Si può utilizzare quanto appena detto come test per verificare se ci sono ancora righe da leggere

```
c = open("test.txt","r")
linea = c.readline()
while linea!="":
    print (linea)
    linea = c.readline()
```

- Nota: ogni riga di un file di testo contiene almeno un carattere, una riga bianca contiene almeno *\n* (new line)
- Non è possibile confondere una riga bianca con la fine del file

Scrittura

- L'argomento della write dev'essere una stringa se vogliamo inserire altri tipi di valore in un file occorre prima convertire le variabili in stringhe.
- Il modo più semplice è quello di usare la funzione str():

```
x = 52
f.write (str(x))
```

- Un esempio più complesso di conversione in stringa

```
NumAuto=100
a="In luglio abbiamo venduto %d automobili." % NumAuto
print(a)

In luglio abbiamo venduto 100 automobili

f.write(a)
```

Esempi di lettura file

- Ciclo che legge tutte le righe:

```
f = open("prova.dat", "r")
line=f.readline()
while line!="":
    print (line)
    line=f.readline()
```

- Il ciclo for permette di riscrivere il ciclo while di cui sopra in forma sintetica

```
f = open("dati.dat", "r")
for line in f:
   print(line)
```

- Il ciclo while permette di variare l'ordine di lettura, il ciclo for no

`read()`, `readline()` o `readlines()`?

Dipende dallo scopo!

- Se il file da manipolare è “piccolo” allora `read()` e `readlines()` sono utilizzabili.
- Se la dimensione del file eccede la memoria disponibile allora è necessario usare un ciclo con:
  - `readline()` perché carica in memoria solo la riga corrente
  - `read(size)` perché carica in memoria solo `size` elementi

Metodo ‘sicuro’ per leggere sequenzialmente un file

```
with open('nomefile', 'r') as f:
  for line in f:
    print(line)
  f.close()
```

l’uso di `with ... as f:` garantisce la chiusura del file anche se dovessimo dimenticare `f.close()`

l’uso di `for line in f:` garantisce un uso efficiente della memoria

Dai dati alle informazioni: lettura da file

- Le informazioni vengono immagazzinate in maniera “permanente” tramite file.
- Il primo passo per l’analisi dei dati è la loro estrazione dai file.
- É fondamentale sapere come sono archiviate le informazioni nei dati: formattazione.

csv, tsv ...

Due formati per immagazzinare tabelle di dati

- CSV: comma separeted values, i campi sono separati da ',' o ';'

```
CET,Temperatura maxC,Temperatura mediaC,Temperatura minC,Punto di rugiadaC,MeanDew PointC,Min DewpointC,
Max Umidità, Mean Umidità, Min Umidità, Max Pressione a livello del marehPa, Mean Pressione a livello
del marehPa, Min Pressione a livello del marehPa, Max VisibilitàKm, Mean VisibilitàKm, Min VisibilitàkM,
Max Velocità del ventoKm/h, Mean Velocità del ventoKm/h, Max Velocità RafficaKm/h,Precipitazionimm,
CloudCover, Eventi,WindDirDegrees
2015-1-1,6,-1,-7,-2,-4,-8,100,83,48,1037,1034,1033,10,9,8,11,3,,0.00,,Nebbia,271
2015-1-2,8,1,-7,1,-3,-8,100,81,40,1036,1031,1027,19,10,8,11,5,,0.00,3,,310
2015-1-3,9,2,-4,2,-1,-5,100,85,48,1029,1024,1018,10,7,5,11,5,27,0.00,,Nebbia,37
2015-1-4,17,7,-3,5,-4,-14,100,51,7,1029,1022,1017,31,16,6,35,13,53,0.00,1,Nebbia,11
2015-1-5,12,3,-6,-3,-7,-10,80,52,16,1032,1028,1024,,,,19,8,,0.00,,,342
2015-1-6,8,1,-5,-1,-3,-7,100,72,36,1029,1024,1021,26,12,8,19,10,34,0.00,3,,22
2015-1-7,9,2,-4,2,1,-2,100,84,54,1033,1030,1028,19,8,5,13,5,,0.00,6,,360
2015-1-8,9,3,-3,2,1,-4,100,80,47,1035,1033,1031,19,8,6,11,6,,0.00,6,,315
2015-1-9,12,3,-6,2,-1,-5,100,80,35,1034,1030,1023,26,16,6,11,6,,0.00,,,344
2015-1-10,17,8,-1,6,2,-1,100,73,38,1024,1019,1017,26,14,5,11,6,,0.00,,Nebbia,303
2015-1-11,17,7,-3,7,-4,-12,100,58,7,1026,1017,1012,9,4,1,53,11,58,0.00,3,Nebbia,1
2015-1-12,14,4,-4,0,-6,-10,81,48,13,1030,1028,1026,31,31,31,13,6,,0.00,,,330
```

- TSV: tab separated values, i campi sono separati da tabulazioni

```
CET Temperatura maxC    Temperatura mediaC  Temperatura minC    Punto di rugiadaC   MeanDew PointC  Min
DewpointC   Max Umidità  Mean Umidità    Min Umidità     Max Pressione a livello del marehPa     Mean
Pressione a livello del marehPa     Min Pressione a livello del marehPa     Max VisibilitàKm    Mean
VisibilitàKm    Min VisibilitàkM    Max Velocità del ventoKm/h  Mean Velocità del ventoKm/h     Max
Velocità RafficaKm/h    Precipitazionimm    CloudCover  Eventi  WindDirDegrees
2015-1-1    6   -1  -7  -2  -4  -8  100 83  48  1037    1034    1033    10  9   8   11  3       0.00
    Nebbia  271
2015-1-2    8   1   -7  1   -3  -8  100 81  40  1036    1031    1027    19  10  8   11  5       0.00    3
       310
2015-1-3    9   2   -4  2   -1  -5  100 85  48  1029    1024    1018    10  7   5   11  5   27  0.00
```

Dalle stringhe ai dati

- `stringa.strip()` restituisce una copia della stringa senza spazi, tabulazioni o fine linea
- `stringa.strip('abcde')` restituisce una copia della stringa senza i caratteri `abcde`
- `tokens = split('delim')` restituisce una lista con i frammenti della stringa separati dalla stringa `'delim'`

Dalle stringhe ai dati

Quando non si identificano separatori nella stringa potrebbero esserci regolarità posizionali

una stringa è una lista di stringhe: si può sfruttare lo splicing

```
valore1 = float(stringa[:5])
```

Alcuni trucchi utili

- Il carattere \n (new line) a volte da fastidio

```
f = open("test.txt","w")
f.write("linea uno\nlinea due\nlinea tre\n")
f.close()
c = open("test.txt","r")
linea = c.readline()
if linea != "linea uno": #linea="linea uno\n"
   print ("E' diverso")
```

- Viene stampato unicamente "E' diverso"
- Il metodo strip permette di risolvere il problema

```
linea="linea uno\n"
v=linea.strip("\n") # elimina il \n finale
print (v)
```

```
linea uno
```

Trucchi 2: split

- Spesso si ha a che fare con file CSV (Comma separated value)
- Esempio del contenuto di un file CSV
  Paolo,Rossi,25
  Gaetano,Scirea,28
  ...
- Leggendo, una riga, il metodo **split** permette di ottenere una lista in cui gli elementi sono ...

```
c = open("provacsv.txt","r")
linea = c.readline()
valori = linea.split(",")
print (valori)
```

```
['Paolo', 'Rossi', '25\n']
```

File CSV – MS Excel

- I file CSV sono spesso usati per scambiare dati tra applicazioni
- MS Excel può sia leggere sia salvare i dati di un foglio di calcolo in formato .CSV
  - Fate una prova: aprite excel, inserite dei dati in alcune celle e salvate il file in formato CSV
  - Aprite il file con un editor di testi
- I file CSV quindi non sono altro che file di testo