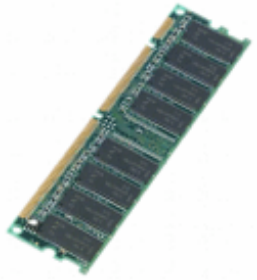
Gestione File

Mirko Cesarini - Dario Pescini
nome.cognome@unimib.it

Università di Milano Bicocca

Hardware di un computer

- **Memoria centrale** (o RAM)



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

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

- Caratteristiche
  - **Memoria secondaria di lavoro**
  - Capienza molto maggiore (rispetto alla memoria centrale)
  - Accesso più lento ai dati (rispetto alla memoria centrale)
  - Non volatile (spegnendo il computer, NON si perdono le informazioni)

Paragone con le biblioteche

- 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 computer e biblioteca

- Memoria centrale. Equivalente alla scrivania della biblioteca
  - 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. I dati sono organizzati sotto forma di file e directory.
  - File: contenitore di dati (equivalente al libro)
  - Directory: contenitore di file (equivalente ad uno scaffale della biblioteca)

File: motivazioni

- Se devo spegnere il computer, come faccio a non perdere i risultati ottenuti?
  - Salvo i dati nella memoria di massa
- Se devo processare una grossa 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 alla volta
- Obiettivo della lezione: vedere la gestione dei file in Python

Altro paragone

- "Lavorare con i file è simile a leggere un libro: per usarli li devi prima **aprire** e quando hai finito li **chiudi**. Mentre il libro è aperto puoi **leggerlo**, puoi **scrivere** una nota sulle sue pagine. La maggior parte delle volte leggerai il libro in ordine, una pagina dopo l'altra, ma nulla ti vieta di **saltare** a determinate pagine facendo uso dell'indice."
- **Apertura**, **chiusura**, **lettura**, **scrittura** e **posizionamento** sono le attività tipiche che un linguaggio di programmazione mette a disposizione per gestire il contenuto di un file . . .
- . . . i linguaggi di programmazione gestiscono ulteriori attività: la creazione di un nuovo file e la cancellazione del contenuto di un file esistente

Due operazioni frequenti: apertura e chiusura

- **Apertura** di un file: si informa il sistema operativo ...
  - ... che si vuole utilizzare un certo file. Il sistema operativo effettua dei controlli prima di consentire l'accesso (Il file esiste? L'utente dispone dei permessi di accesso? Qualche altro utente lo sta utilizzando?).
  - ... indicando 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
  - Quando un file viene chiuso, il sistema operativo cerca di completare velocemente le operazioni in sospeso
  - Domanda: perché è necessario informare il sistema operativo prima di staccare una chiavetta usb?
  - Quando si informa il sistema operativo che si vuole staccare un dispositivo di memorizzazione esterno, il sistema operativo completa le operazioni in sospeso prima di dare l'ok alla rimozione

Python: apertura e chiusura di un file

- Esempio di apertura di due file

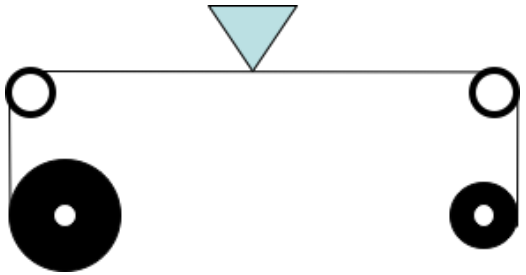
```
1 f1 = open("source.dat","r")
2 f2 = open("destination.dat","w")
```

- La funzione *open* accetta due argomenti:
  - il nome del file
  - la modalità di apertura (maggiori dettagli nelle slide successive)
    - (Sola) lettura: "r"
    - (Lettura e) scrittura: "w"
    - Append: "a"
    - ...
- *f1* ed *f2* sono i descrittori dei file
  - Descrittore: una variabile che rappresenta il file nello script
  - I comandi che implementano le operazioni sul file saranno richiamati con la *notazione punto* sul descrittore es:,
    - `f1.close()  #sara' descritto in seguito`
    - `f2.read()  #sara' descritto in seguito`
- A breve torneremo sul descrittore del file e sulla *notazione punto*

Python: modalità di apertura di un file

- "r". Chiamata anche *read*, o *(sola) lettura*: vogliamo aprire un file solo per leggerne il contenuto (senza scriverci sopra). Alcuni sistemi operativi permettono a più utenti contemporaneamente di aprire un file in lettura. Se si tenta di scrivere su un file aperto in sola lettura, si genera un errore e il programma termina.
- "w". Chiamata anche *write*, o *scrittura*, si desidera aprire un file per poterci scrivere sopra
  - `f2 = open("destination.dat","w")`
    Se non esiste già un file chiamato *destination.dat* l'apertura in scrittura creerà un nuovo file (vuoto)
  - Se il file esiste già, il contenuto preesistente sarà eliminato
- "a". *append*: simile alla modalità *write*, ma il contenuto preesistente del file non viene cancellato, i nuovi dati saranno accodati ai dati già esistenti. Se il file è vuoto o non esiste già, non c'è differenza tra modalità "w" e "a"
- . . . esistono altre modalità di apertura che non vedremo

Modello del *nastro magnetico*



- Una testina di lettura / scrittura
- La testina scorre su un "nastro" logicamente suddiviso in caselle (una di seguito all'altra)
  - ogni casella rappresenta un byte
  - le caselle sono numerate . . .
    - a partire dalla posizione 0
    - fino alla fine del file
- i file sono memorizzati uno dopo l'altro (come più film su una videocassetta)
- Operazioni di lettura/scrittura: il nastro scorre, la testina legge/scrive caselle contigue
- Dopo l'ultima operazione di lettura/scrittura, la testina è **posizionata** all'inizio della casella successiva (pronta per una nuova operazione)
- Il nastro può essere fatto scorrere velocemente, senza dover leggere quello che c'è in mezzo

File: note sul posizionamento

- Quando si apre un file in modalità sola lettura ("r")
  - la testina viene posizionata all'inizio del file
  - ogni volta che si leggono dei dati il nastro scorre
  - lettura dopo lettura, la testina arriva fino alla fine del file
  - è possibile effettuare dei salti, sia avanti, sia indietro
- Quando si apre un file in modalità scrittura ("w")
  - viene cancellato tutto il contenuto del file esistente
  - la testina viene posizionata all'inizio del file pronta per scrivere
  - dopo ogni scrittura, la testina si posiziona alla fine dell'ultima cella letta/scritta (pronta per eseguire l'operazione successiva)
  - è possibile effettuare dei salti, sia avanti sia indietro
- Quando si apre un file in modalità "append" ("a")
  - la testina viene posizionata alla fine del file
  - i nuovi contenuti vengono aggiunti (scritti) in coda
  - dopo ogni scrittura, la testina si posiziona alla fine dell'ultima cella letta/scritta (pronta per eseguire l'operazione successiva)

Python: esempi di lettura e scrittura

- Apertura di un file

```
1 f = open("test.txt","w")
```

- Per scrivere dati nel file occorre utilizzare il metodo write:

```
2 f.write("Adesso")
3 f.write("Scriviamo qualcosa")
```

- Modello nastro magnetico: i due comandi write scrivono le 2 frasi una di seguito all'altra
- Notazione punto dei comandi es., `f.write()` (sarà ripresa in seguito)
- La chiusura del file avvisa il sistema operativo che la scrittura è conclusa ed il file è disponibile per altri scopi:

```
4 f.close()
```

- Solo dopo aver chiuso il file possiamo riaprirlo in lettura e leggerne il contenuto
- Se cerchiamo di aprire in lettura un file che non esiste otteniamo un errore:

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

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

- Nel comando open possiamo
  - fornire solo il nome del file
    in tal caso il file sarà ricercato nella directory corrente
  - fornire il percorso completo, es.

```
2 k = open("c:\\notes\\lecture1.txt","r")
```

- Nelle stringhe python, quando si vuole inserire il \ semplice, questo va scritto raddoppiato
- Perchè secondo voi?
- Per evitare che il \ e i caratteri successivi vengano interpretati come comandi di formattazione. In questo modo il \n presente nella stringa non viene interpretato come *a capo*

Lettura

- Reminder

```
1 f = open("test.txt","w")
2 f.write("Adesso")
3 f.write("Scriviamo qualcosa")
4 f.close()
```

- Immaginiamo di riaprire il file dopo un po' di tempo
- Questa volta la modalità di apertura è "r":

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

- Il metodo read legge dati da un file. Senza argomenti legge l'intero contenuto del file:

```
6 testo = g.read()
7 print( testo )
```

```
AdessoScriviamo qualcosa
```

- Notate qualcosa di strano?
- Non c'è spazio tra *Adesso* e *Scriviamo* perché la seconda stringa è stata scritta subito dopo la prima
- read accetta anche un argomento che specifica quanti caratteri leggere

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

```
Adess
```

```
1  f = open("caratteri.txt","w")
2  f.write("abcdefgh")
3  f.close()
```

```
4  g = open("caratteri.txt","r")
5  txt = g.read(5)
6  print(txt) # abcde
```

- In un'operazione di lettura, read restituisce solamente i caratteri effettivamente disponibili,
- Es: si richiede di leggere 5 caratteri, ma il file termina dopo averne letti 3. In qs caso read restituisce solamente i caratteri effettivamente letti

```
7  txt2 = g.read(5)
8  print(txt2) # fgh
```

- Se si cerca di leggere qualcosa dopo aver raggiunto la fine del file, read restituisce una stringa vuota

```
9  txt3 = g.read()
10 print(len(txt3)) # 0
```

Copiare il contenuto di un file su un altro file

- Esercizio su lettura e scrittura di un file

```
1 f1 = open("source.dat","r")
2 f2 = open("destination.dat","w")
3 print('Copying content')
4 content = f1.read()
5 f2.write(content)
6 f1.close()
7 f2.close()
8 print('Done')
```

- Questo script presuppone che l'elaboratore abbia abbastanza memoria per poter caricare in memoria centrale l'intero contenuto del file

Variabili, Oggetti, Metodi

- Abbiamo visto che sulle variabili che ospitano i descrittori di file è possibile eseguire delle operazioni con la *notazione punto*
- Le operazioni richiamate con la *notazione punto* sono chiamate *metodi*
- *read*, *write* e *close* sono metodi messi a disposizione da una variabile che ospita un descrittore di file
- Abbiamo già visto altri esempi di metodi

```
1 a=[]
2 a.append(5)
3 print(a) # [5]
```

- Una variabile su cui possono essere richiamati dei metodi è chiamata *oggetto*
- In python tutte le variabili sono oggetti. Questo significa che ogni variabile mette a disposizione dei metodi

- I metodi richiamabili dipendono dal tipo di valore ospitato dalla variabile

```
1 a=[]
2 a.append(5)
3 print(a) # [5]
4 f=open('abc.txt', 'r')
5 cont=f.read()
6 print(cont) # ...
7 f.close()
```

ok

```
1 f=open('abc.txt', 'r')
2 f.append('5')
```

AttributeError: 'file' object
has no attribute 'append'

```
1 a=[]
2 a.close()
```

AttributeError: 'list' object
has no attribute 'close'