Laboratorio d'Informatica

Corso di Laurea Magistrale in Ingegneria per l'Ambiente e il Territorio

A.A. 2021/2022

Docente: Lorenzo Putzu

Lezione 08

Il linguaggio Python

Accesso ai *file*

- L'unico meccanismo di acquisizione dei dati d'ingresso (*input*) da parte di un programma Python visto finora è basato sulla funzione input, che consente di acquisire sequenze di caratteri scritte attraverso la tastiera (mentre il programma è in esecuzione) nella finestra della *shell*.

- Similmente, per comunicare all'utente i risultati delle elaborazioni svolte da un programma (*output*) si è usata la funzione print, che consente di stampare nella finestra della *shell* il valore di qualsiasi espressione.

- Oltre a queste modalità di *input*/*output* ne esistono altre, per esempio l'uso di interfacce grafiche e sistemi di puntamento (*mouse*, ecc.), e l'uso dei *file* della memoria secondaria.

Accesso ai *file*

- L'uso dei *file* della memoria secondaria è una modalità fondamentale sia per l'*input* (per acquisire dati precedentemente memorizzati in un *file*, per esempio da un utente o da un altro programma) che per l'*output* (per memorizzare in un *file* i risultati prodotti da un programma).

- Dal punto di vista di un programma Python un *file* consiste in una **sequenza ordinata** di valori. Più precisamente, un *file* può essere:
  - una sequenza di ***byte***, che possono essere visti come numeri interi con valori nell'insieme {0*,* 1*, . . . ,* 255}
  - una sequenza di **caratteri**, che sono i classici *file* di testo che prenderemo in considerazione in questo corso

Accesso ai *file*

- Nel linguaggio Python l'accesso ai *file* è reso possibile attraverso diverse funzioni di libreria, le quali consentono di eseguire su un *file* di testo due tipi di operazioni:
  - lettura: acquisizione dati dal *file*, sotto forma di una **stringa**
  - scrittura: memorizzazione nel *file* sotto forma di una **stringa**
- Quindi per entrmabe la conoscenza delle funzioni per l'elaborazione di stringhe è fondamentale e entrambe si compongono di tre fasi:
  - apertura del *file*, per mezzo della funzione predefinita open
  - esecuzione di una o più operazioni di lettura o scrittura
  - chiusura del *file*, per mezzo della funzione predefinita close

Accesso ai *file*

- La funzione open consente di indicare il nome del *file* al quale si vuole accedere e la modalità di accesso desiderata (lettura o scrittura): tale operazione viene detta "apertura" del *file*.

- Da un *file* aperto in modalità di lettura è possibile solo acquisire dati; su un *file* aperto in modalità di scrittura è invece possibile solo scrivere dati. Il tentativo di scrivere dati su un *file* aperto in modalità di lettura, o di acquisire dati da un *file* aperto in modalità di scrittura, produce un errore.

- La funzione close "chiude" il *file*, cioè impedisce di eseguire ulteriori operazioni di lettura o scrittura su di esso (fino a che non venga eventualmente riaperto con un'altra chiamata di open).

Accesso ai *file: apertura*

- La funzione open restituisce un valore di un tipo predefinito contenente alcune informazioni sul *file*. Tali informazioni dovranno essere usate in ogni successiva operazione sullo stesso *file*, inclusa la chiusura, perciò è fondamentale assegnarle ad una variabile. La sintassi è la seguente:
- **Sintassi**:

```
variabile = open (nome-file, modalità)
```

- variabile: il nome della variabile che verrà associata al *file*
- nome-*file*: una **stringa** contenente il nome del *file*
- modalità: una **stringa** che indica la modalità di apertura (lettura o scrittura)

Accesso ai *file: apertura*

- Il nome del *file* che si desidera aprire deve essere passato come argomento della funzione open sotto forma di **stringa** e può essere:
  - assoluto, cioè preceduto dalla sequenza (detta anche *pathname*) dei nomi delle *directory* che lo contengono a partire dalla *directory* radice del *file system*, scritta secondo la sintassi prevista dal sistema operativo del proprio calcolatore
  - relativo, cioè composto dal solo nome del *file*: questo è possibile solo se la funzione open è chiamata da un programma o da una funzione che si trovi nella **stessa** *directory* che contiene il *file* da aprire

Accesso ai *file: apertura*

- Per esempio, nel caso di un *file* di nome dati.txt che si trovi nella *directory* C:\Users\Lorenzo\ di un sistema operativo Windows:
  - il nome **assoluto** è C:\Users\Lorenzo\dati.txt
  - il nome **relativo** è dati.txt

- Se un *file* con lo stesso nome (dati.txt) è memorizzato nella *directory* /Users/Lorenzo/ di un sistema operativo Linux oppure Mac OS:
  - il nome **assoluto** è /Users/Lorenzo/dati.txt
  - il nome **relativo** è ancora dati.txt

Accesso ai *file: apertura*

- Per i sistemi operativi Windows, nella stringa contenente il nome assoluto di un *file* il linguaggio Python consente di usare come separatore il carattere / (separatore nei sistemi Unix like) invece del carattere \ (separatore nei sistemi Windows).

- Questo consente di evitare ambiguità nel caso di nomi di *file* o *directory* che iniziano con il carattere n, dato che nella stringa corrispondente comparirebbe la sequenza "\n" che Python interpreta come *newline*, altrimenti si dovrebbe scrivere come nome del file assoluto:

```
"C:\\Users\\Lorenzo\\dati.txt"
```

- In alternativa (solo nel caso di funzioni non da *Shell*) si può usare il nome del file relativo, che nel caso in cui la funzione che lo richiama sia scritta in un file memorizzato nella **stessa** *directory* nella quale si trova il *file* da aprire, sarà

```
"dati.txt"
```

Accesso ai *file: apertura*

- È possibile aprire in modalità di lettura solo un *file* **esistente**. Se il *file* non esiste si otterrà un errore.

- La modalità di scrittura consente invece anche la **creazione** di un nuovo *file*. Più precisamente, attraverso la modalità di scrittura è possibile:
  - **creare** un **nuovo** *file*
  - **aggiungere** dati **al termine** di un *file* già **esistente**
  - **sovrascrivere** (cancellare e sostituire) il contenuto di un *file* già **esistente**

Accesso ai *file: apertura*

- Nella chiamata di open la modalità di accesso è indicata (come secondo argomento) da una **stringa** composta da un singolo carattere:
  - "r" (*read*): **lettura** (se il *file* non esiste si ottiene un **errore**)
  - "w" (*write*): (sovra)**scrittura** se il *file* non esiste viene creato ma se il *file* esiste viene sovrascritto, cancellando i dati che contiene
  - "a" (*append*): **scrittura** (aggiunta) anche in questo caso se il *file* non esiste viene creato mentre se il *file* esiste i nuovi dati saranno aggiunti **dopo** quelli già presenti

Accesso ai *file: apertura*

- Consideriamo ancora il file dati.txt presente nella *directory* C:\Users\Lorenzo\. Per aprire tale *file* in modalità di **lettura**, memorizzando il valore restituito da open in una variabile di nome f, si dovrà scrivere l'istruzione:

```
f = open("C:/Users/Lorenzo/dati.txt", "r")
```

- oppure se la chiamata di open **non** è scritta nella *shell*, **e** il programma che la contiene si trovi nella **stessa** *directory* del *file.*

```
f = open("dati.txt", "r")
```

- Consideriamo ancora la *directory* C:\Users\Lorenzo\. Per creare un **nuovo** *file* di nome dati2.txt al suo interno e associarlo a una variabile di nome nf si potrà usare una qualunque delle due modalità di scrittura:

```
nf = open("C:/Users/Lorenzo/dati2.txt", "w")   oppure
nf = open("C:/Users/Lorenzo/dati2.txt", "a")
```

- Anche in questo caso è possibile usare il nome relativo del *file* (sempre sotto le stesse ipotesi):

```
nf = open("dati2.txt", "w")          oppure
nf = open("dati2.txt", "a")
```

Accesso ai *file: apertura*

- Se invece si volesse aprire in scrittura il *file* già esistente dati.txt nella *directory* C:\Users\Lorenzo\ **cancellando** automaticamente gli eventuali dati in esso presenti, si dovrà usare la modalità "w"

```
fw = open("C:/Users/Lorenzo/dati.txt", "w")
```

- Se invece si volesse aprire in scrittura un *file* già esistente per **aggiungere** dati dopo quelli eventualmente già presenti (senza cancellare questi ultimi) si dovrà usare la modalità "a".

```
fw = open("C:/Users/Lorenzo/dati.txt", "a")
```

- Anche in questo caso è possibile usare il nome relativo del *file*, nelle condizioni indicate in precedenza.

Accesso ai *file: chiusura*

- Quando le operazioni di lettura o scrittura su un *file* sono terminate, il *file* deve essere chiuso attraverso la funzione predefinita close. Questo impedirà l'esecuzione di ulteriori operazioni su tale *file*, fino a che esso non venga eventualmente riaperto.
- **Sintassi**:

```
variabile.close()
```

- dove variabile deve essere la variabile usata nell'apertura dello **stesso** *file* attraverso la funzione open

- Perciò assumendo che un *file* di nome dati.txt sia stato aperto in lettura con l'istruzione

```
f = open("dati.txt", "r")
```

- esso dovrà essere chiuso con la chiamata:

```
f.close()
```

Accesso ai *file*

- Se lo si desidera è anche possibile aprire più file contemporaneamente. A questo scopo nel momento dell'apertura è necessario associare ogni *file* a una **diversa** variabile.
- Per esempio, se si volesse aprire in lettura un *file* esistente di nome dati.txt, e creare un nuovo *file* di nome risultati.txt da aprire in scrittura, associandoli rispettivamente alle variabili f_dati e f_risultati, le operazioni di apertura e chiusura sarebbero le seguenti:

```
f_dati = open("dati.txt", "r")
f_risultati = open("risultati.txt", "w")
...
f_dati.close()
f_risultati.close()
```

Accesso ai *file: scrittura*

- In un *file* di testo che sia stato aperto in scrittura (in modalità "w" oppure "a") è possibile scrivere dati sotto forma di **stringhe** attraverso la funzione predefinita write.
- **Sintassi**:

```
variabile.write(stringa)
```

- variabile è la variabile associata al *file*
- stringa è una **stringa** contenente la sequenza di caratteri da scrivere nel *file*

- Mentre un *file* è aperto in scrittura la funzione write può essere chiamata più volte per scrivere nel *file* diverse sequenze di caratteri. Qualsiasi chiamata di write scrive la corrispondente sequenza di caratteri **al termine** del *file*, cioè dopo gli eventuali caratteri già presenti al suo interno.

Accesso ai *file: scrittura*

- L'esecuzione di un programma contenente la seguente sequenza di istruzioni:

```
f = open("testo.txt", "w")
f.write("Prima riga.\nSeconda riga.")
f.close()
```

- crea un *file* di nome testo.txt nella stessa *directory* del programma (se il *file* esiste ne cancella il contenuto) e scrive al suo interno la stringa indicata.
- Aprendo successivamente il *file* con un qualsiasi *editor* di testi si osserverà il seguente contenuto:

```
Prima Riga
Seconda Riga
```

Accesso ai *file: aggiunta*

- Se successivamente si riapre il *file* in scrittura in modalità "a" sarà possibile **aggiungere** del testo **al termine** dello stesso *file*, senza cancellare il testo esistente.
- Per esempio, dopo l'esecuzione del seguente programma, anch'esso memorizzato nella stessa *directory* del *file* testo.txt:

```
f = open("testo.txt", "a")
f.write("\nTerza riga.")
f.close()
```

- il contenuto del *file* sarà il seguente:

```
Prima Riga
Seconda Riga
Terza Riga
```

Accesso ai *file*

- Se successivamente si riaprisse il *file* in scrittura in modalità "w" il suo contenuto verrebbe cancellato.
- Per esempio, si memorizzi ancora nella stessa *directory* il seguente programma, e lo si esegua:

```
f = open("testo.txt", "w")
f.write("Nuova riga.")
f.close()
```

- Il contenuto del *file* sarà ora il seguente:

```
Nuova Riga
```

Accesso ai *file: lettura*

- Le operazioni di lettura da un *file* possono essere eseguite attraverso tre diverse funzioni predefinite:
  - read
  - readline
  - readlines
- Tutte e tre le funzioni restituiscono una parte del contenuto del *file*, o l'intero contenuto, sotto forma di **una stringa** oppure di **una lista di stringhe**.
- Il valore restituito da tali funzioni deve di norma essere memorizzato in una variabile per poter essere successivamente elaborato.

Accesso ai *file: lettura*

- La funzione read acquisisce l'intero contenuto di un *file*, che viene restituito sotto forma di **una stringa**. Le eventuali interruzioni di riga presenti nel *file* vengono codificate con il carattere *newline* "\n".
- La **sintassi** della chiamata di questa funzione (così come quella di readline e readlines) è analoga a quella della funzione close:

```
variabile.read()
```

- dove variabile è la variabile associata al *file*.
- Come si è detto in precedenza, di norma la stringa restituita da read deve essere memorizzata in una variabile per poter essere successivamente elaborata.

Accesso ai *file: lettura*

- Si consideri un *file* di nome testo.txt scritto manulamente con un editor di testo classico e contenente le seguenti righe di testo:

```
Prima riga.
Seconda riga.
Terza riga.
```

- Si memorizzi ora il seguente programma nella **stessa** *directory* di tale *file*, e lo si esegua:

```
f = open("testo.txt", "r")
s = f.read()
f.close()
```

- Il risultato sarà la memorizzazione della seguente stringa nella variabile s, come si potrà osservare valutando l'espressione s nella *shell*:

```
"Prima riga.\nSeconda riga.\nTerza riga."
```

- Si noti la codifica delle interruzioni di riga con il carattere *newline*.

Accesso ai *file: lettura*

- La **prima** chiamata di read dopo l'apertura (in modalità di lettura) di un *file* ne acquisisce sempre l'intero contenuto. Questo significa che dopo aver chiuso e riaperto (in lettura) uno stesso *file* sarà possibile riacquisirne il contenuto con una chiamata di read.
- Se invece si eseguono due o più chiamate di read mentre un *file* è aperto (senza chiuderlo e riaprirlo prima di ognuna di tali chiamate), ogni chiamata successiva alla prima restituirà **una stringa vuota**, poiché l'intero contenuto del *file* è già stato acquisito.
- Modificando quindi il programma precedente ed eseguendolo nuovamente:

```
f = open("testo.txt", "r")
s = f.read()
s2 = f.read()
f.close()
```

- La variabile s2 conterrà una stringa vuota

Accesso ai *file: lettura*

- La funzione readline acquisisce **una singola riga** di un *file*, restituendola sotto forma di **una stringa**.
- Per "riga" di un *file* s'intende una sequenza di caratteri fino alla prima interruzione di riga, oppure, se il *file* non contiene interruzioni di riga, l'intera sequenza di caratteri contenuta in esso.
- Nel primo caso anche l'interruzione di riga, codificata con il carattere *newline*, farà parte della stringa restituita da readline.
- La **sintassi** è la seguente:

```
variabile.readline()
```

- dove variabile indica come al solito la variabile associata al *file*.

Accesso ai *file: lettura*

- La **prima** chiamata di readline dopo l'apertura (in lettura) di un *file* acquisisce sempre la **prima** riga ed ogni eventuale altra chiamata di readline (prima che il *file* venga chiuso) acquisisce la riga **successiva** a quella acquisita dalla chiamata precedente.
- Se tutte le righe di un *file* sono state acquisite da una sequenza di chiamate di readline, ogni chiamata successiva restituirà **una stringa vuota**.
- Considerando sempre il *file* testo.txt degli esempi precedenti e si esegua il seguente programma dopo averlo memorizzato nella stessa *directory* di tale *file*:

```
f = open("testo.txt", "r")
a = f.readline()
b = f.readline()
c = f.readline()
d = f.readline()
f.close()
```

- Le variabili corrispondenti ad a, b, c e d conterranno rispettivamente le tre righe del file e una stringa vuota:

Accesso ai *file: lettura*

- La funzione readline è utile quando si desidera acquisire ed elaborare una singola riga alla volta di un dato *file*.
- Questo si può ottenere attraverso una sequenza di chiamate di readline all'interno di un'istruzione iterativa, che dovrà terminare non appena readline restituirà una stringa vuota (che indica che l'intero contenuto del *file* è già stato acquisito) come nel programma seguente

```
f = open("testo.txt", "r")
print("Il file contiene le seguenti righe:")
riga = f.readline()
while riga != "" :
print(riga)
riga = f.readline()
f.close()
```

Accesso ai *file: lettura*

- Questo programma funziona anche su file che presentano righe “vuote” seguite da altre righe, infatti le prime righe “vuote” contengono in realtà un’interruzione di riga, cioè il carattere *newline*, e quindi non sono realmente vuote.
- Per questo motivo l’iterazione del programma precedente termina **non** quando s’incontra la prima di tali righe, ma quando si è raggiunta l’effettiva fine del *file*.
- E’ anche possibile chiamare la funzione read dopo una o più chiamate di readline, ma in questo caso read acquisirà il contenuto del *file* a partire dalla riga **successiva** all’ultima acquisita da readline, e fino al termine del *file*.
- In generale per ogni operazione di lettura valgono le seguenti regole:
  - l’acquisizione inizia dal punto in cui si era conclusa la precedente operazione di lettura
  - se l’intero contenuto del file è già stato acquisito dalle operazioni precedenti, viene restituita una stringa vuota

Accesso ai *file: lettura*

- La funzione readlines acquisisce l'**intera** sequenza di caratteri contenuta in un *file*, e la restituisce sotto forma una **lista di stringhe**, ciascuna delle quali contiene **una riga** del *file* (incluso il carattere *newline*).
- La **sintassi** è la seguente:

```
variabile.readlines()
```

- dove variabile è ovviamente la variabile associata al *file*.
- Anche la funzione readlines è utile quando si desidera elaborare separatamente ogni riga di un dato *file*, ma a differenza di readline consente l'acquisizione dell'intero *file* con una sola chiamata.

Accesso ai *file: lettura*

- Considerando ancora il *file* testo.txt e il seguente programma

```
f = open("testo.txt", "r")
righe = f.readlines()
f.close()
```

- Dopo l'esecuzione la variabile righe conterrà la lista:

```
["Prima riga.\n", "Seconda riga.\n", "Terza riga."]
```

- Per poter eleborare ogni singola riga sarà quindi necessario un'istruzione iterativa in modo simile a quanto visto per l'elaborazione delle liste

Accesso ai *file: lettura*

- Il comportamento della funzione readlines e' simile a quello di read:
  - la **prima** chiamata di readlines dopo l'apertura (in modalità di lettura) di un *file* ne acquisisce sempre l'intero contenuto
  - se si eseguono due o più chiamate di readlines mentre un *file* è aperto (senza chiuderlo e riaprirlo prima di ognuna di tali chiamate), ogni chiamata successiva alla prima restituirà **una lista vuota**, poiché l'intero contenuto del *file* è già stato acquisito

Accesso ai *file: lettura*

- Le tre funzioni viste finora sono equivalenti dal punto di vista dell'acquisizione di dati da un *file* di testo. In altre parole, qualsiasi operazione di lettura si possa svolgere con una di esse si potrà svolgere anche con le altre due.
- Tuttavia una data operazione potrebbe richiedere programmi di diversa **complessità** a seconda della funzione che si intende usare. È quindi importante individuare la funzione più appropriata da usare in un dato contesto.
- Per esempio, se s'intende elaborare separatamente ciascuna riga di un *file* è consigliabile usare readline o readlines, dato che entrambe suddividono automaticamente le righe del *file*; usando read sarà invece il programmatore a dover scrivere ulteriori istruzioni per individuare il termine di ciascuna riga, in corrispondenza dei caratteri *newline*.

Esercizi

1. Scrivere un programma che utilizzi la funzione già scritta nel esercizio Lez06_es05, che riceva una lista di dizionari contenenti ciascuno nome, cognome ed età di una persona
   - definire una funzione che scriva tali dati in un nuovo *file* di nome `dati_anagrafici.txt` (una riga per persona). Per esempio, i dati del dizionario:
     `{"nome":"Ugo", "cognome":"Gui", "eta'":25}`
     dovranno corrispondere alla riga:
     `Ugo Gui 25`
   - definire una funzione senza argomenti che acquisisca i dati dal *file e* li memorizzi nella lista di dizionari e restituisca tale lista

Esercizi di riepilogo: campionato di basket

2. Un *file* di testo di nome `partite.txt` contiene i risultati di tutte le partite giocate nel corso di un campionato di pallacanestro. Ogni riga corrisponde a una partita, per esempio:

```
Milano Venezia 80 75
Trento Sassari 82 95
```

- scrivere una funzione senza argomenti che restituisca un dizionario contenente la classifica del campionato (2 punti per una vittoria, 0 per una sconfitta), e avente per chiavi i nomi delle squadre; per esempio:
  `{"Milano":20, "Trento":24, ...}`
- Scrivere una funzione che, dato un dizionario come quello precedente, stampi sulla *shell* la classifica in ordine decrescente di punteggio; esempio:

```
Trento 24
Milano 20
...
```

Esercizi di riepilogo: campionato di basket

Suggerimento:

- memorizzazione della classifica in un dizionario:
  - acquisire il contenuto del *file* con `readlines`
  - scandire le partite (le righe del *file*) e costruire un dizionario avente per chiavi i nomi delle squadre e zero (punti) come valore associato a ciascuna chiave; esempio:`{"Milano":0, "Trento":0, ...}`
  - scandire una seconda volta le partite e aggiungere nel dizionario due punti alla squadra vincente di ogni partita
- stampa della classifica:
  - costruire due liste contenenti una le chiavi (nomi delle squade), l'altra i valori (i punti delle stesse squadre) del dizionario, usando le funzioni (*metodi*) `keys` e `values`, che restituiscono chiavi e valori di un dizionario **nello stesso ordine**
  - ordinare le due liste, per esempio con l'algoritmo di ordinamento per selezione