Programming Lab

Parte 3
Liste e List comprehension
I dati: interagire con i file ed il formato CSV

Laura Nenzi

Many slides by Stefano Russo

Liste

- Una lista è una sequenza di elementi di qualsiasi tipo
- Gli indici delle liste funzionano nello stesso modo di quelli delle stringhe:
  - L'indice può essere qualsiasi espressione di tipo intero.
  - Se tentate di leggere o modificare un elemento che non esiste, ottenete un messaggio d'errore IndexError.
  - Con un indice di valore negativo, si conta a ritroso dalla fine della lista.

```
>>> t = ['a', 'b', 'c', 'd', 'e', 'f']
>>> t[-1]
['f']
```

Liste

- Operazioni sulle liste: concatenazione (+), ripetizione (*)
- Slicing delle liste

```
>>> t = ['a', 'b', 'c', 'd', 'e', 'f']
>>> t[1:3]
['b', 'c']
```

- Metodi delle liste: append, sort

```
>>> t = ['a', 'b', 'c']
>>> t.append('d')
>>> t
['a', 'b', 'c', 'd']
```

Liste

- Una lista è mutabile

```
>>> numeri = [42, 123]
>>> numeri[1] = 5
>>> numeri
[42, 5]
```

- Gli operatori di appartenenza funzionano con le liste

Operator	Description	Example
in	Returns True if a sequence with the specified value is present in the object	x in y
not in	Returns True if a sequence with the specified value is not present in the object	x not in y

Liste

- Anche l'operatore **in** funziona con le liste:

```
>>> formaggi = ['Cheddar', 'Edam', 'Gouda']
>>> 'Edam' in formaggi
True
>>> 'Brie' in formaggi
False
```

- Il modo più frequente di attraversare gli elementi di una lista è un ciclo for

```
for formaggio in formaggi:
    print(formaggio)
```

```
for i in range(len(formaggi)):
    print(formaggi[i])
```

```
for i, formaggio in enumerate(formaggi):
    print(i, formaggio)
```

Liste

- Attenzione: i metodi delle liste modificano l'oggetto di partenza

```
>>> list1 = ['carlo','magno']
>>> list2=list1 #list2 ed list1 puntano alla stesso oggetto in memoria.
>>> list2[0]='alessandro'
>>> list1
['carlo','alessandro']
```

- Per puntare ad un oggetto diverso usare .copy() oppure list4=list3[:]

```
>>> list3 = ['11',7,23]
>>> list4 = list3.copy()
```

Liste

- È importante distinguere tra operazioni che modificano le liste e operazioni che creano nuove liste.
- Ad esempio il metodo append non crea una nuova lista

```
>>> t1 = [1, 2]
>>> t2 = t1.append(3)
>>> t1
[1, 2, 3]
>>> t2   #Il valore di ritorno di append è None.
None
```

Liste

- Questa differenza è importante quando scrivete delle funzioni che devono modificare delle liste.
- Per esempio, questa funzione non cancella il primo elemento della lista:

```
def non_decapita(t):
    t = t[1:]                # SBAGLIATO!
t4 = [1, 2, 3]
non_decapita(t4)
print(4)
```

- Un’alternativa valida è scrivere una funzione che crea e restituisce una nuova lista.

```
def ritaglia(t):
    return t[1:]
resto = ritaglia(t4)
print(resto)
```

Ridurre: compattare una sequenza di elementi in un singolo valore

- Esempio di come sommare tutti i numeri in una lista:

```
def somma_tutti(lista):
    totale = 0
    for elemento in lista:
        totale += elemento
    return totale
```

Forma abbreviata di:
`totale = totale + elemento`

- In realtà è una cosa così comune che esite la funzione sum

```
def somma_tutti(lista):
        return sum(lista)
```

Mappare: applicare una funzione su ciascun elemento di una sequenza.

- Esempio: funzione che prende in input una lista di numeri interi in [0, 9] e ritorna una lista di stringhe, corrispondenti ai numeri scritti in Italiano

```
def converti_numeri(lista_numeri):
    lista_stringhe = []
    dizionario_numeri_stringhe = { 0:"0", 1: "1", 2: "2", 3:"3", 4:"4",
                                5:"5", 6:"6", 7:"7", 8:"8", 9:"9"}
    for numero in lista_numeri:
        lista_stringhe.append(dizionario_numeri_stringhe[numero])
    return lista_stringhe
```

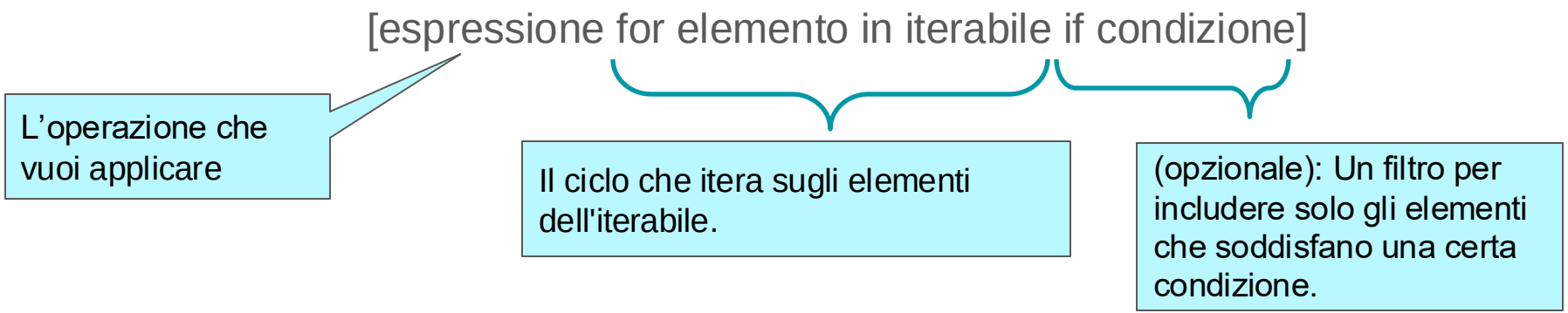
Filtrare: selezionare alcuni elementi di una lista per formare una sottolista

- Esempio: funzione che prende una lista di numeri e restituisce solo quelli pari

```
def solo_pari(lista):
    pari = []
    for n in lista:
        if n%2==0:
            pari.append(n)
    return pari
```

List comprehension

È una sintassi che permette di creare liste nuove applicando una trasformazione o un filtro a un iterabile esistente (come una lista, una tupla o un range):



List comprehension

Usando il ciclo for:

```
pari = []
for n in range(10):
    if n%2==0:
        pari.append(n)
print(pari)
```

Usando la list comprehension:

```
pari2 = [n for n in range(10) if n%2==0]
print(pari2)
```

I file

Per i programmi, uno dei modi più semplici di mantenere i loro dati è di leggerli e scriverli su file di testo.

I files sono molto comodi per salvare dati ancora prima dei database.

Un file di testo è un una sequenza di caratteri salvata su un dispositivo permanente come un disco fisso, una memoria flash.

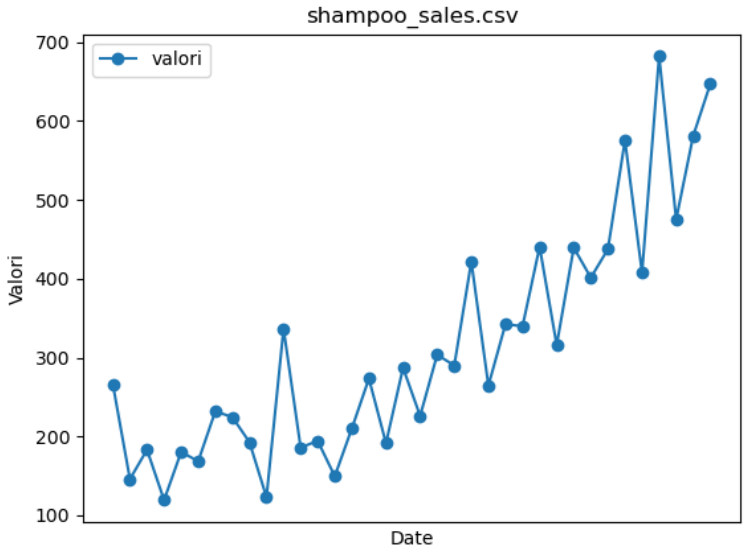
Uno dei formati più standard è il CSV, ovvero “Comma-Separated Values”.

I file

In genere, in un file CSV ogni riga è una “entry”, e ci può essere una intestazione (opzionale) di una o più righe. Estensione .csv o alle volte .txt

Esempio “shampoo_sales.csv”:

```
Date,Sales
01-01-2012,266.0
01-02-2012,145.9
01-03-2012,183.1
01-04-2012,119.3
```



Interagire con i files in Python

Si usa l'oggetto "file". Cosa è un oggetto in dettaglio lo vedremo nella prossima lezione, per ora prendiamolo così com'è.

```
my_file = open('shampoo_sales.csv', 'r')
print(my_file.read())
my_file.close()
```

Modalità di apertura del file, in questo caso "r" sta per "read". Se vorrò anche scriverci, userò "w"

dopo aver lavorato con un file bisogna chiuderlo

```
> python read_file.py
Date,Sales
01-01-2012,266.0
01-02-2012,145.9
01-03-2012,183.1
...
```

Interagire con i files in Python

L’ogetto file ha I seguenti metodi:

Modalità di accesso	
'r'	lettura (reading) (default)
'w'	scrittura (writing)
'x'	crea un nuovo file e lo apre in scrittura
'a'	scrittura e se il file esiste aggiunge (append) il contenuto in coda al file
'b'	modalità binaria
't'	modalità testuale (default)
'+'	open a disk file for updating (reading and writing)

Usare lo “slicing” delle stringhe

Si può usare lo slicing delle stringhe per stampare solo un pezzetto del file:

```
my_file = open('shampoo_sales.txt', 'r')
print(my_file.read()[0:50])
my_file.close()
```

```
> python read_file.py
Date,Sales
01-01-2012,266.0
01-02-2012,145.9
01-03
```

Usare lo “slicing” delle stringhe

Si può usare lo slicing delle stringhe per stampare solo un pezzetto del file (versione più sofisticata)

```
# Apro il file
my_file = open('shampoo_sales.txt', 'r')

# Leggo il contenuto
my_file_contents = my_file.read()

# Stampo a schermo i primi 50 caratteri
if len(my_file_contents) > 50:
    print(my_file_contents[0:50] + '...')
else:
    print(my_file_contents)

# Chiudo il file
my_file.close()
```

Leggere i file

Il file si può anche leggere riga per riga, una alla volta:

```
my_file = open('shampoo_sales.csv', 'r')
print(my_file.readline())
print(my_file.readline())
print(my_file.readline())
my_file.close()
```

```
> python read_file.py
Date,Sales

01-01-2012,266.0

01-02-2012,145.9
```

Leggere i file

Il file si può anche leggere riga per riga, una alla volta:

```
my_file = open('shampoo_sales.csv', 'r')
print(my_file.readline())
print(my_file.readline())
print(my_file.readline())
my_file.close()
```

```
> python read_file.py
Date,Sales

01-01-2012,266.0

01-02-2012,145.9
```

Leggere i file

Il file si può anche leggere riga per riga tutto in un colpo in modo “pythonico”:

```
my_file = open('shampoo_sales.csv', 'r')

for line in my_file:
    print(line)

my_file.close()
```

```
> python read_file.py
Date,Sales

01-01-2012,266.0
```

...

Leggere i file

Per non dover ogni volta chiudere il file posso usare with:

```
with open('shampoo_sales.csv') as file:
        print(file.read())
```

```
with open('shampoo_sales.csv') as file:
        print(file.readline() )
```

Nomi di file e percorsi

- Il file sono organizzati in directory (chiamate anche “cartelle”).
- Ogni programma in esecuzione ha una “directory corrente”, che è la directory predefinita per la maggior parte delle operazioni che compie.
- Quando aprite un file in lettura, Python lo cerca nella sua directory corrente.
- Il modulo os fornisce delle funzioni per lavorare con file e directory (“os” sta per “sistema operativo”). os.getcwd restituisce il nome della directory corrente:

```
>>> import os
>>> cwd = os.getcwd()  #current working directory
>>> cwd
'/Users/laura/git/MyProgrammingLab'
```

- Una stringa come `'/Users/laura/git/MyProgrammingLab'`, che individua la collocazione di un file o una directory, è chiamata percorso.

Nomi di File e percorsi

- Un semplice nome di file, come shampoo_sales.csv è pure considerato un percorso, ma è un percorso relativo perché si riferisce alla directory corrente. Se la directory corrente è '/Users/laura/git/MyProgrammingLab', il nome di file shampoo_sales.csv starebbe per:
  '/Users/laura/git/MyProgrammingLab/shampoo_sales.csv’

- Un percorso che comincia per / non dipende dalla directory corrente; viene chiamato per- corso assoluto.

- I percorsi visti finora sono semplici nomi di file, quindi sono percorsi relativi alla directory corrente.

Nomi di file e percorsi

- Per avere invece il percorso assoluto, potete usare os.path.abspath:

```
>>> os.path.abspath('shampoo_sales.csv')
'/Users/laura/git/MyProgrammingLab/shampoo_sales.csv'
```

- Vedere i vari metodi di `os.path` come exists, isfile

- Se voglio tenere i file dentro una cartella specifica ad esempio ”dati” posso inserire il percorso dalla cartella sono:

```
my_file = open('dati/shampoo_sales.csv', 'r')

for line in my_file:
    print(line)

my_file.close()
```

Scrivere i file

Scrivere su un file (nota il “w” nella funzione open):

```
my_file = open('saluti.txt', 'w')
my_file.write('Ciao mondo!')
my_file.close()
```

.

..ma non lo useremo granchè in questo corso.

Scrivere i file

Scrivere su un file (nota il “w” nella funzione open):

```
my_file = open('saluti.txt', 'w')
my_file.write('Ciao mondo!')
my_file.close()
```

```
with open('saluti.txt', 'w’) as file:
        file.write('Ciao mondo!')
```

..ma non lo useremo granchè in questo corso.

Aggiungere testo su un file

Modificare un file (nota il “a” nella funzione open):

```
my_file = open('saluti.txt', 'a')
my_file.write('Addio!')
my_file.close()
```

```
with open('saluti.txt', 'a' ) as file:
        file.write('Addio!')
```

..ma non lo useremo granchè in questo corso.

Scrivere moduli

- Qualunque file che contenga codice Python può essere importato come modulo.
- Supponiamo di avere un file di nome wc.py che contiene il codice che segue:

```
def contarighe(nomefile):
    conta = 0
    for riga in open(nomefile):
        conta += 1
return conta
print(contarighe('wc.py'))
```

- legge se stesso e stampa il numero delle righe nel file

Scrivere moduli

- Potete importare il file così:

```
>>> import wc
7
>>> wc.contarighe('wc.py')
7
```

- I programmi che verranno importati come moduli usano spesso questo costrutto:

```
if __name__ == '__main__':
    print(contarighe('wc.py'))
```

- Il codice così è eseguito solo se avvio lo script non se lo importo

Leggiamo i valori di un file CSV

Per leggere i dati da un file CSV bisogna fare un po’ di cose nuove:

1) Il metodo “.split” per separare le stringhe su uno specifico carattere;

2) La conversione di una stringa a valore numerico (floating point);

3) Sapere come aggiungere un elemento ad una lista.

Leggere il valori di un file CSV

1) Il metodo “.split” per separare le stringhe su uno specifico carattere:

```
mia_stringa = 'Date,Sales\n'
lista_elementi = mia_stringa.split(',')
print(lista_elementi)
```

```
> python read_file.py
['Date', 'Sales\n']
```

Leggere il valori di un file CSV

2) La conversione di una stringa a valore numerico (floating point)

```
mia_stringa = '5.5'
mio_numero = float(mia_stringa)
```

Leggere il valori di un file CSV

3) Sapere come aggiungere un elemento ad una lista

```
mia_lista = [1,2,3]
mia_lista.append(4)
```

Leggere il valori di un file CSV

```
# Inizializzo una lista vuota per salvare i valori
values = []

# Apro e leggo il file, linea per linea
my_file = open('shampoo_sales.csv', 'r')
for line in my_file:

   # Faccio lo split di ogni riga sulla virgola
   elements = line.split(',')

   # Se NON sto processando l'intestazione...
   if elements[0] != 'Date':

        # Setto la data e il valore
        date  = elements[0]
        value = elements[1]

        # Aggiungo alla lista dei valori questo valore
        values.append(value)
```

Esercizio

Scrivete una funzione `sum_csv(file_name)`

- che sommi tutti i valori delle vendite degli shampoo del file passato come argomento

Provatelo sul file “shampoo_sales.csv”.

Poi, scaricate il vostro script e testatelo su Autograding

p.s. il massimo punteggio è 8/10 con quello che abbiamo vosto fino ad oggi a lezione. Potete però provare a raggiungere il 10 da soli.

Altri esercizi

1. Definire una funzione che prende in input un file ed una parola e conta quante volte quella parola è presente sul file
2. Definire una funzione che prende come input un file e conta quante volte ogni parola è presente
3. Definire una funzione che prende in input un file e costruisce un dizionario con chiavi la lettere iniziali e con valore le parole di lunghezza maggiore contenute nel file che iniziano con quelle lettere.
4. Definire una funzione conteggio che prende come input un file e ritorna un dizionario con chiave la prima parola di ogni frase e valore il numero di volte che una frase inizia con quella parola. Considerare come inizio di frase qualsiasi parola che segue un punto, un punto esclamativo, un punto interrogativo o si trova all'inizio del testo.
5. Definire una funzione che prende come input un file, rimuove tutte le righe duplicate, scrive il risultato in un nuovo file chiamato unique.txt.