Strutture dati complesse

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

Università di Milano Bicocca

Python: tipi di variabili

- Tipi semplici (già visti)
  - int
  - float
  - ...
- Tipi complessi
  - liste
  - dizionari
  - tuple
  - ...

Liste

Liste

- Una variabile di tipo semplice permette di memorizzare solamente un valore per volta, es.

```
1 voto=30
```

- Una lista permette di memorizzare più informazioni contemporaneamente, es.

```
1 elencoVoti=[28, 30, 21, 27]
```

- Paragone:
  - Variabile di *tipo int*: cassetto singolo
  - Variabile di tipo *lista*: cassettiera
- Vantaggi delle liste rispetto all'uso delle variabili singole:
  - gestione unitaria di informazioni tra loro collegate
  - implementazione di operazioni ripetitive (semplificazione attraverso l'uso dei cicli . . . lo approfondiremo a breve)

Terminologia: *elementi* di una lista

- *Elementi*: i valori che costituiscono la lista
- Una lista è creata racchiudendo gli *elementi* (separati da virgola) tra parentesi quadrate [ ]
- Esempio:

```
1 a=[10, 20, 30, 40]
2 b=["Pippo", "Pluto", "Paperino"]
3 c=[1, "Qui", 20, "Quo", 5, "Qua"]
```

- La lista è un tipo di dato che svolge quelle funzionalità che in altri linguaggi di programmazione sono svolte dagli *array* (chiamati anche *vettori*)
- In python, una lista può ospitare tipi di dati diversi

Indice di una lista

- In una lista ogni elemento è identificato da un *indice* numerico
- Cosa viene visualizzato?

```
1 a=[10, 20, 30, 40]
2 b= ["Pippo", "Pluto"]
3 print( a[0] )   # 10
4 print( a[1] )   # 20
5 print( a[1] + a[2] )   # 50
6 print( a[1 + 2] )      # 40
7 print( b[1] )   # Pluto
```

- Con la notazione nome_lista[indice_numerico] è possibile accedere ad un singolo elemento della lista, utilizzando il corrispondente indice_numerico
- Il primo elemento (a partire da sx di una lista) è contraddistinto dall'indice 0, il secondo elemento è contraddistinto dall'indice 1, ...
- L'indice numerico permette di definire un ordine tra gli elementi di una lista

Utilità delle liste

- Esecuzione di operazioni ripetitive facilitata
  - Raggruppo i dati da elaborare in una lista
  - Scrivo un algoritmo
    - Utilizzo una variabile intera come indice per accedere ad un elemento
    - Con un ciclo ripeto l'operazione su (tutti/una parte) degli elementi della lista
- Esempio

```
1 voti_libretto=[28, 30, 21, 27]
2 i=0
3 while i<4:
4     print ("voto: "+str(voti_libretto[i]))
5     i = i+1
```

```
28
30
21
27
```

- Lo stesso algoritmo può essere facilmente adattato ad una lista di centinaia di elementi

- Altro esempio

```
voti_libretto=[28, 30, 21, 27]
somma=0
i=0
while i<4:
    somma=somma+voti_libretto[i]
    i = i+1
media = somma / float(i)
print(media)
```

- Che cosa fa questo script?

Tipi ammessi per l'indice

- Data una lista generica (per esempio)

```
1 a = [1, 3, 5, 8] # 3 e' l'elemento di indice 1
```

- Come indice è possibile utilizzare . . .

. . .

Espressioni algebriche

```
1 print( a[3-2] )
```

```
3
```

Variabili

```
1 b=2
2 print( a[b] )
```

```
5
```

- Unico requisito: l'indice (o il risultato dell'eventuale espressione) deve essere di tipo intero

```
1 print( a[1.0] )
```

```
TypeError: sequence
index must be integer
```

```
1 print( a["a"] )
```

```
TypeError: sequence
index must be integer
```

Operazioni sulle liste

- Oltre ad accedere ad un elemento di una lista

```
1 a = [1, 3, 5, 8]
2 print( a[1] )
```

```
3
```

- E' possibile assegnare un nuovo valore ad uno degli elementi già esistenti della lista

```
1 a = [1, 3, 5, 8]
2 a[3] = 100
3 print(a)
```

```
[1, 3, 5, 100]
```

Inserimento di elementi in una lista vuota

- Il comando nome_lista[indice]=nuovo_valore si può utilizzare solamente per rimpiazzare elementi già esistenti. Es.

```
1 lista=[1,3,6,7]
2 lista[3]=50
3 print( lista )
```

```
[1, 3, 6, 50]
```

```
1 lista=[2,3]
2 lista[5]=50
```

```
IndexError: list assignment
index out of range
```

- Per aggiungere *posti vuoti* in coda si può utilizzare il comando append (vedi esempio qua sotto):

```
1 lista=[]          #creo una lista vuota
2 lista.append(4) #aggiungo un elemento con valore 4
3 lista.append(6) #aggiungo un elemento con valore 6
4 print( lista )
```

```
[4, 6]
```

Accesso dal fondo e cancellazione di elementi

- Domanda, cosa viene visualizzato?

```
a = [1, 3, 5, 8]
print( a[-1] ) # 8
print( a[-0] )  # 1 ... e' equivalente ad a[0]
print( a[-2] ) # 5
```

- Se un indice ha valore negativo il conteggio parte dalla fine della lista!
  - L'ultimo elemento ha indice -1
  - Il penultimo elemento ha indice -2, ...
- Per cancellare un elemento

```
a = [1, 3, 5, 8]
del(a[1])
print(a)  # [1, 5, 8]
del(a[2])
print(a)  # Che cosa viene visualizzato? [1, 5]
```

- La funzione len() restitutisce la lunghezza di una lista

```
1 a=["Marco", "Marta", "Martino", "Mario"]
2 print(len(a))
```

```
4
```

- esiste una lista speciale che non contiene alcun elemento: si chiamata lista vuota ed è indicata da []

```
1 b=[]
2 print(b)
3 print(len(b))
```

```
[]
0
```

Operatore *in*

- *in* è un operatore booleano
- Controlla se un valore è presente in una lista
- Esempio

```
1 elSquadre = ['Juventus', 'Inter', 'Milan', 'Roma']
2 sq1='Inter'
3 testResult = sq1 in elSquadre
4 print(testResult)
5 print('Giacomo' in elSquadre)
```

```
True
False
```

Esempio

```
1  voti_libretto=[28, 30, 28, 27]
2  target=28
3  i=0
4  count=0
5  if target in voti_libretto:
6      while i<len(voti_libretto):
7          if voti_libretto[i]==target:
8              count = count + 1
9          i=i+1
10     print("Voto %d conseguito %d volte" % (target, count))
11 else:
12     print("Non ci sono "+str(target))
```

```
Voto 28 conseguito 2 volte
```

Semantica degli operatori

- L'operatore + concatena le liste:

```
1 a = [1, 2, 3]
2 b = [4, 5, 6]
3 c = a + b
4 print(c)
```

```
[1, 2, 3, 4, 5, 6]
```

- L'operatore * ripete una lista un certo numero di volte:

```
1 d = [0] * 4
2 print(d)
```

```
[0, 0, 0, 0]
```

```
1 e=[1, 2, 3] * 3
2 print(e)
```

```
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

- Nel primo esempio abbiamo ripetuto [0] quattro volte. Nel secondo abbiamo ripetuto la lista [1, 2, 3] tre volte

Riepilogo: come creare una lista

- Inizializzo una lista con valori predefiniti

```
a=[1,5,10,'Pippo', 3, 'Pluto', 'Topolino']
```

- Voglio creare una lista vuota di 10 elementi, tutti uguali a 0

```
b=[0] * 10  # [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

- Voglio creare una lista di 10 elementi generati casualmente

```
import random
li=[]
i=0
while i<10:
    rnd=random.randint(0,10)
    li.append(rnd)
    i=i+1
print(li)
```

```
[8, 1, 9, 7, 4, 2, 8, 6, 10, 9]
```

- . . . ci sono tanti altri modi

Tuple

Tuple

- Una tupla è una struttura dati simile alla lista ma immutabile
  - Gode della maggior parte delle proprietà di una lista
  - Non può essere modificata, una volta creata
- Alcuni esempi:

```
1 a = (5, 3, 8)
2 b = ("Mario", "Rossi", 24)
3 c = 4, 10, 21
```

- Una tupla è dichiarata:
  - assegnando ad una variabile dei valori, separati da virgole
  - (opzionale) gli elementi possono essere racchiusi tra parentesi tonde ( )
- I concetti di *indice*, *elementi*, *indici negativi* e *ordine tra elementi* delle liste, si applicano anche alle tuple
  - . . . non sarano ripetuti qua tali concetti, fate riferimento a quanto detto sulle liste

Operazioni ammesse sulle tuple

- E' possibile applicare alle tuple le operazioni applicate alle liste che non alterano il contenuto della tupla stessa
  - L'operatore *in*

```
1 print(5 in (3, 2, 5))
```

```
True
```

  - La funzione *len*

```
1 a=(2, 3, 5)
2 print(len(a))
```

```
3
```

  - (e ora il più importante) accedere ad un singolo elemento della tupla

```
1 a=(2,3,5)
2 print( a[1] ) #notate l'uso delle [] e NON delle ( )
```

```
3
```

Accesso ad elementi: liste e tuple

```
1 a=(10,11,12,13,14) # creazione di una tupla
2 b=['d', 'e', 'f']  # creazione di una lista
3 print(a[2])
4 print(b[2])
```

```
12
f
```

- Cosa cambia (tra liste e tuple) nella sintassi per accedere ad un elemento?
  - Niente: in entrambi i casi si usano le [ ]
  - In generale, l'operatore [ ] serve per estrarre un elemento da una collezione. Si applica sia alle *tuple* sia alle *liste*
- Posso usare le ( ) per accedere ad un elemento di una lista/tupla? **NO!**

```
1 print(  a(2)  )
```

TypeError: 'tuple' object is not callable

```
1 print(  b(2)  )
```

TypeError: 'list' object is not callable

Le tuple non possono essere modificate

- Non si possono aggiungere elementi

```
1 a=(1,2,3)
2 a.append(4)
```

```
... ?AttributeError: 'tuple' object has no attribute 'append'
```

- Non si possono rimuovere elementi

```
1 del(a(0))
```

```
... SyntaxError: can't delete function call
```

- Provo a cancellare ma con le parentesi quadre

```
1 del(a[0])
```

```
... TypeError: object doesn't support item deletion
```

- Non si possono modificare elementi

```
1 a[0]=2
```

```
... TypeError: 'tuple' object does not support item assignment
```

A che cosa servono le tuple?

- Permettono di raggruppare informazioni eterogenee e gestirle unitariamente, es.

```
studente = (629435, 'Rossi', 'Mario')
auto = ('BA555AB', 'Fiat', 'Punto')
```

  - La non modificabilità è un vantaggio in questi casi

- Utili per definire degli insiemi di costanti

```
giorniSettimana=('lunedi', 'martedi', 'mercoledi',
        'giovedi', 'venerdi', 'sabato', 'domenica')
```

- Le tuple permettono di memorizzare i dati e accedervi più velocemente delle liste

Attenzione!

- Per creare una tupla basta inizializzarla

```
1 a=(2, 5, 'ciao')
2 print( a )
```

```
(2, 5, 'ciao')
```

- Se inizializzate una tupla con un singolo valore, ricordatevi di inserire una virgola alla fine, altrimenti il singolo valore verrebbe interpretato non come una tupla ma come un'espressione matematica
- è la virgola prima ancora delle () a definire una tupla

```
1 b=2, # b=(2,) #equivalente
2 print( b )
3 print( type(b) )
```

```
(2)
<type 'tuple'>
```

```
1 b=(2)
2 print( b )
3 print(type(b))
```

```
2
<type 'int'>
```

Avete notato . . .

- Qualche volta è stata utilizzata la print con una sintassi strana

```
variabile = 10
print("testo", 5, "altro testo", variabile)
```

```
('testo', 5, 'altro testo', 10)
```

- Come viene interpretato l'insieme degli argomenti della print()?
  - Come una tupla

```
tu="testo", 5, "altro testo"
print(tu)
print(type(tu))
```

```
('testo', 5, 'altro testo')
<type 'tuple'>
```

Stringhe: rivisitazione

- L'operatore [] permette di estrarre un elemento da una collezione.
- Proviamo ad applicarlo ad una stringa, che cosa viene stampato?

```
1 a = "io sono una stringa"
2 print( a[0] )    # i
3 print( a[1] )    # o
4 print( a[-1] )   # a
```

- Le stringhe potrebbero essere assimilate a *liste* di caratteri?
- Se una stringa fosse una lista di caratteri, dovrei poter assegnare valori

```
1 a[1]="f"
```

TypeError: TypeError: 'str' object does not support item assignment

- Quindi, una stringa **non è assimilabile** ad una lista di caratteri

Stringhe, assimilabili a ...

- Riepilogo delle proprietà delle stringhe
  - E' possibile accedere ad un singolo carattere utilizzando l'operatore [ ]
  - E' possibile usare sia indici positivi, sia indici negativi con l'operatore [ ]
  - Non è possibile assegnare un nuovo valore ad un elemento della stringa
- Domanda: a quale struttura dati (tra quelle da voi conosciute) è assimilabile una stringa?
- Ad una *tupla di caratteri*
- Facciamo una verifica applicando ad una stringa alcuni degli operatori delle tuple

```
1 st='ciao'
2 print(len(st))    # 4
3 print('a' in st) # True
```

```
1  st='ciao, analizzami 10 volte'
2  target=('vocali', 'consonanti', 'cifre', 'spazi')
3  domini=('aeiou', 'bcdfghjklmnpqrstvwxyz', '0123456789', ' ')
4  cont=[0] * len(domini)
5  i=0
6  while i<len(st):
7      j=0
8      while j<len(domini):
9          if st[i] in domini[j]:
10             cont[j]+=1
11         j+=1
12     i+=1
13 print('Statistiche su "%s"' % (st))
14 j=0
15 while j<len(domini):
16     print('- %s %d' % (target[j], cont[j]))
17     j+=1
```

```
Statistiche su "ciao, analizzami 10 volte"
- vocali 10
- consonanti 9
- cifre 2
- spazi 3
```

Slicing

Slicing

- Reminder: con la notazione nome_lista[indice_numerico] è possibile accedere ad un singolo elemento di una lista (o di una tupla)
- Date un'occhiata al codice seguente

```
1 a='Ecco un esempio'
2 b = a[5:7]
3 print(b)
```

```
un
```

- Che compito svolge l'istruzione a riga 2?
- Effettua uno *slicing*, cioè estrae un sottoinsieme (una sottostringa) dalla stringa di partenza
- Qual è la sintassi?
  sotto_stringa=stringa[pos_a:pos_b]
  - pos_a: indice del primo elemento della sottostringa da estrarre
  - pos_b: indice dell'elemento successivo all'ultimo elemento da estrarre
  - reminder: il primo elemento di una stringa ha indice 0

Esempi di Slicing

sub_st=st[pos_a:pos_b]

- pos_a: indice del primo elemento della sottostringa
- pos_b: indice dell'elemento successivo all'ultimo (della sottostringa)

Secondo voi cosa viene stampato?

```
1 st='ABCDEFGHIL'
2 print(st[0:5])   # ABCDE
3 print(st[:5])    # ABCDE
4 print(st[5:])    # FGHIL
5 print(st[:])       # ABCDEFGHIL
6 print(st[0:-1])  # ABCDEFGHI
7 print(st[-5:-3]) # FG
```

- Se nello slicing non viene inserito
  - il primo elemento (a sinistra dei :), si assume che la sottostringa parta da posizione 0
  - l'ultimo elemento (a destra dei :), si assume che sia len(st), in altre parole l'ultimo elemento della sottostringa coinciderà con l'ultimo elemento della stringa di partenza

Slicing, liste, tuple

Lo slicing può essere applicato anche a liste e tuple

```
1 tu=(0,1,2,3,4,5,6,7,8)
2 s1=tu[2:5]
3 print(s1)
```

```
(2, 3, 4)
```

```
1 i=0
2 li=[]
3 while i<9:
4     li.append(i)
5     i+=1
6 sl=li[2:5]
7 print(li)
8 print(sl)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8]
[2, 3, 4]
```

Slicing, liste, tuple 2

- Aggiungiamo dei comandi ad uno degli script precedenti
- Cosa viene stampato?

```
1  li=[0,1,2,3,4,5,6,7,8]
2  sl=li[2:5]
3  print(li) # [0, 1, 2, 3, 4, 5, 6, 7, 8]
4  print(sl) # [2, 3, 4]
5  li[2]=77
6  print(li) # [0, 1, 77, 3, 4, 5, 6, 7, 8]
7  print(sl) # [2, 3, 4]
8  sl[0]=99
9  print(sl) # [99, 3, 4]
10 print(li) # [0, 1, 77, 3, 4, 5, 6, 7, 8]
```

- Conclusione: le (sotto)liste ottenute con lo slicing:
  - sono copie delle liste di partenza (quindi sono strutture dati diverse)
  - godono di vita propria

Dizionari

I dizionari

- Problema: vogliamo memorizzare in una struttura dati il contenuto di un vocabolario inglese-italiano
  - semplificazione: lavoriamo con parole singole
  - vogliamo memorizzare una parola in inglese e la corrispondente parola in italiano
  - altra semplificazione: supponiamo che data una parola inglese, ci sia sempre una e una sola parola italiana associata
- Potrei usare 2 liste:
  - parole_italiano[‘automobile’, ‘motocicletta’, ‘bicicletta’]
  - english_words[‘car’, ‘motorbike’, ‘bicycle’]
  - gli elementi delle due liste sono logicamente collegati tra loro (parole con lo stesso significato occupano la stessa posizione)
  - ogni volta che aggiungo, cancello, modifico un elemento da una lista, devo fare lo stesso nell’altra
- Sarebbe comodo poter rappresentare i due insiemi di dati attraverso un’unica struttura dati
- I dizionari servono a questo

Esempi di creazione e uso di un dizionario

- Un dizionario è un insieme di (più) coppie
- Un esempio

```
1 d={"auto":"car", "moto":"motorbike"}
2 print(d)
```

```
{'auto': 'car', 'moto': 'motorbike'}
```

- Ogni elemento del dizionario è una coppia chiave:valore
- Nella coppia, la chiave è separata dal valore per mezzo dei :
- In fase di dichiarazione, l'insieme delle coppie è separato da virgole e racchiuso tra parentesi graffe
- La comodità maggiore dei dizionari: per mezzo della chiave è possibile risalire al valore

```
1 print( d["auto"] )
```

```
car
```

Chiave

- Immaginiamo di avere un insieme di dati su persone che possiamo strutturare come nella tabella qua sotto
  - Di ogni persona abbiamo una collezione di informazioni: Matricola, Cognome, Nome
  - I dati di una persona sono raccolti in una singola riga
- All'interno della tabella qua sotto la **chiave** è quella colonna, il cui valore permette di identificare univocamente una (e una sola) riga
- Ogni singolo valore di *matricola* permette di rintracciare i dati di una e una sola persona

Matricola	**Cognome**	**Nome**
62345	Rossi	Mario
45678	Rossi	Antonio
54255	Verdi	Matteo
89422	Brambilla	Marco

Dizionari: una generalizzazione delle liste

- Le liste usano gli interi come chiavi
- Qualsiasi tentativo di usare altri tipi di dati come indice produce un errore
- I dizionari sono analoghi alle liste, ma possono usare come chiavi anche valori non numerici
- Possono essere usate come chiave di un dizionario
  - Stringhe,
  - Tuple
  - Valori numerici,
  - Valori booleani
- Non possono essere usate come chiavi di un dizionario
  - Liste, dizionari ...
  - ... le strutture dati complesse che mutano nel tempo

Dizionari: precisazioni

- E' possibile da una chiave risalire al valore corrispondente,

```
1 a={"auto":"Fiat Punto", "scooter":"honda 125"}
2 print( a["auto"] )
```

```
Fiat Punto
```

- Non possono esserci duplicazioni di chiavi in un dizionario
- Assegnare un valore ad una chiave esistente sovrascrivere il vecchio valore

```
1 a={"auto":"Fiat Punto", "scooter":"honda 125"}
2 a["auto"]="Volkswagen Golf"
3 print( a )
```

```
{'auto': 'Volkswagen Golf', 'scooter': 'honda 125'}
```

Dizionari: precisazioni 2

- E' possibile aggiungere nuove coppie chiave-valore in ogni momento.
- Si può utilizzare la seguente sintassi

```
1 a={"auto":"W Golf", "scooter":"Honda 125"}
2 a["triciclo"]="ape"   # aggiunta di una coppia
3 print(a)
```

```
{'auto': 'W Golf', 'scooter': 'honda 125',
      'triciclo':'ape'}
```

- Attenzione, facendo il parallelo con una lista:
  - l'utilizzo di un tipo non intero come indice della lista avrebbe generato un errore
  - l'occupazione di un nuovo elemento alla lista avrebbe richiesto l'uso del comando .append()

Duplicazioni di valori in un dizionario

- Le chiavi devono essere univoche, i valori possono essere duplicati
- Es. a={'auto': 'BMW', 'moto': 'BMW', 'bicicletta' : 'Bianchi'}
- E' uno dei motivi per cui non è possibile risalire da un valore ad un'unica chiave

```
1 print( a["BMW"] )
```

```
... ?KeyError: 'BMW'.
```

- Dato un valore, è possibile ricercare la/le chiavi associate
- Come? Lo vedremo tra breve