Strutture dati complesse e cicli: esercizi

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

Università di Milano Bicocca

Merge di una lista e una tupla

- Data una lista *li* ed una tupla *tu*, entrambi contenenti numeri, fondi il contenuto delle due strutture dati in una lista *ris*.
- In ris, ogni numero deve ripetersi al massimo una volta

```
1  li=[1,4,5,6,9,10]
2  tu=(1,2,3,4,7,8,9)
3  ris=[]
4  i=0
5  while i<len(li):
6      if li[i] not in ris:
7          ris.append(li[i])
8      i=i+1
9
10 i=0
11 while i<len(tu):
12     if tu[i] not in ris:
13         ris.append(tu[i])
14     i=i+1
```

```
15 print('li')
16 print(li)
17 print('tu')
18 print(tu)
19 print('ris')
20 print(ris)
```

```
li
[1,4,5,6,9,10]
tu
(1,2,3,4,7,8,9)
ris
[1,4,5,6,9,10,2,3,7,8]
```

Ricerca del massimo in una lista

```
1  import random # libreria per generare numeri casuali
2  i=0
3  li=[]
4  while i<10:
5      # genero un num. casuale compreso tra 1 e 10
6      rn = random.randint(1,10)
7      if rn not in li: # se il valore non e' gia' presente
8          li.append(rn)
9      i=i+1
10 maxPos=0
11 i=1
12 while i<len(li):  # li potrebbe avere meno di 10 elementi
13     if li[i]>li[maxPos]:
14         maxPos=i
15     i=i+1
16 print(li)
17 print("Max in pos. %d con valore %d" % (maxPos, li[maxPos]))
```

```
[6, 2, 7, 5, 10, 9, 8]
"Max in pos. 4 con valore 10
```

Liste, Tuple, Dizionari

- Le tuple permettono di accoppiare informazione eterogenee tra loro. Vediamo un uso congiunto di tuple e dizionari
- Es:

```
1 a1=('Rossi', 'Mario')
2 a2=('Bianchi', 'Andrea')
3 diz1={'cf1':a1, 'cf2':a2}
4 print(diz1)
```

```
{'cf1': ('Rossi', 'Mario'),
 'cf2': ('Bianchi', 'Andrea')}
```

```
1 lista1=[a1,a2]
2 print(lista1)
```

```
[('Rossi', 'Mario'), ('Bianchi', 'Andrea')]
```

Liste, Tuple, Dizionari 2

```
1 a1=('Rossi', 'Mario')
2 a2=('Bianchi', 'Andrea')
3 lista1=[a1, a2]
4 print(lista1)
```

```
[('Rossi', 'Mario'), ('Bianchi', 'Andrea')]
```

```
1 print( lista1[0][1] ) #accedo all'elemento 0 della
2                       #lista lista1(si tratta di a1
3                       #ed e' una tupla),
4                       # all'interno della
5                       #quale accedo all'elemento 1
```

```
Mario
```

Calcolo calorie

```
1  trms=[ #(ingrediente, grammi, calorie_ogni_100_grammi)
2       ('savoiardi',300,391),
3       ('zucchero',100,392),
4       ('uova',220,130),
5       ('caffe',300,6),
6       ('mascarpone',500,460),
7       ('cacao',3,320)  ]
8
9  i=0
10 totCal=0
11 while i<len(trms):
12      grammi=trms[i][1]
13      cal100=trms[i][2]
14      totCal+=grammi*cal100 / 100.0
15      i+=1
16
17 print('Calorie totali tiramisu',totCal)
```

```
('Calorie totali tiramisu', 4178.6)
```

Tuple, immutabilità, creazione

- Le tuple non possono essere modificate una volta create (sono immutabili)
- Ma una tupla può essere sempre ricreata

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

```
(1, 2, 3)
```

```
1 a=(4,5)
2 print(a)
```

```
(4,5)
```

- Nella memoria del calcolatore, la nuova tupla *a* occupa una diversa area (è di fatto una nuova struttura dati)

Conteggio lettere di una stringa

```
1  testo=input("Inserisci una stringa di testo")
2  diz_lettere={}
3  i=0
4  while i<len(testo):
5      lettera=testo[i]
6      #in usato per testare la presenza di una chiave
7      if lettera in diz_lettere:
8          #lettera gia presente nel diz
9          diz_lettere[lettera]=diz_lettere[lettera]+1
10     else:
11         #lettera inserita per la prima volta
12         diz_lettere[lettera]=1
13     i=i+1
14
15 #ora stampo tutto il dizionario
16 print(diz_lettere)
```

Selection sort

- Supponiamo di avere una lista di numeri interi (senza ripetizioni) ma non ordinati
- Vogliamo applicare l'algoritmo di *selection sort* per l'odinamento (ordiniamo in ordine crescente)
  - Si seleziona l'elemento più piccolo
  - Lo si scambia con l'elemento al primo posto
  - Tra gli elementi rimanenti (dal secondo in avanti) si seleziona il valore più piccolo
  - Lo si scambia con l'elemento al secondo posto
  - . . . così via con l'elemento al terzo, al quarto posto, ... fino all'elemento n-1

```
1  li=[9,2,1,5,7,8,6,3,4,0]
2  i=0
3  while i<len(li)-1:
4      minPos=i
5      a=i+1
6      # Cerco la posizione del minimo a partire da a=i+1
7      while a<len(li):
8          if li[a]<li[minPos]:
9              minPos=a # se in a c'e' un minimo, a diventa la
10                      # nuova posizione del minimo
11         a=a+1
12     # scambio gli elementi in posizione i e minPos
13     temp = li[i]
14     li[i]=li[minPos]
15     li[minPos]=temp
16     i=i+1
17 print(li)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```