# sequenze

June 5, 2023

## 0.1 Stringhe, Liste, Tuple: sono sequenze

Stringhe, liste e tuple sono **differenti** tipi di dati, ma hanno molte **operazioni** in comune. Tutti e tre fanno parte della famiglia delle **sequenze**, ovvero collezioni di dati disposte in ordine:

- la stringa è una sequenza di **caratteri**;
- la tupla è una sequenza di valori **non modificabile**;
- la lista è una sequenza di valori **modificabile**.

Sono tre **tipi diversi** e ognuno ha caratteristiche peculiari, tuttavia hanno operazioni in comune come ad esempio - calcolare la lunghezza con `len`; - leggere il valore alla posizione i-esima.

### 0.1.1 Copiare segmenti

Le cellule nel nostro organismo sanno quanto è importante copiare pezzi di sequenze. In python questa è un'operazione possibile su stringhe, liste e tuple.

[2]:
```
testo = "abcdefghijklmnopqrstuvwxyz"
```

[3]:
```
# Il primo e l'ultimo carattere
print(testo[0], testo[25])
```

```
a z
```

[4]:
```
# Dalla posizione 0 (inclusa) alla posizione 5 (esclusa).
print( testo[:5]  )
```

```
abcde
```

[5]:
```
# Dalla posizione 10 (inclusa) alla posizione finale
print( testo[10:] )
```

```
klmnopqrstuvwxyz
```

[6]:
```
# Dalla posizione 7 (inclusa) alla 17 (esclusa)
print ( testo[7:17] )
```

```
hijklmnopq
```

Dall'esempio vedete la sintassi generale:

```
seq[a:b]
```

è una **nuova** sequenza che contiente i valori dalla posizione `a` **inclusa** alla posizione `b` **esclusa** contenuti nella sequenza `seq`.

**Esercizio:** provate questa sintassi dei valori per `a` e `b` - che sono numeri; - che non sono numeri; - `a` minore di `b`; - `a` maggiore di `b`; - `a` minore di `0`; - `b` maggiore di `len(seq)`.

Esistono sintassi alternative, che permettono di omettere uno o entrambi i limiti: - `seq[:b]` allora la posizione iniziale (inclusa) è assunta essere `0`; - `seq[a:]` allora la posizione finale (esclusa) è assunta essere `len(seq)`; - `seq[:]` una copia di tutta la sequenza.

Adesso vediamo un esempio su una lista

[7]:
```
L = ['zero', 'uno', 'due', 'tre', 'quattro', 'cinque']
```

[8]:
```
print( L[3: ])
```

```
['tre', 'quattro', 'cinque']
```

[9]:
```
print( L[ :4])
```

```
['zero', 'uno', 'due', 'tre']
```

[10]:
```
print( L[1:4])
```

```
['uno', 'due', 'tre']
```

[11]:
```
print( L[:])
```

```
['zero', 'uno', 'due', 'tre', 'quattro', 'cinque']
```

Fate attenzione all'asimmetria: **il limite destro è escluso**.

**Copia di una lista** La sintassi aapena vista permette di effettuare una vera copia di una lista, cosa che un semplice assegnamento non fa.

[17]:
```
L = ['zero', 'uno', 'due', 'tre', 'quattro', 'cinque']
```

`X` è un secondo riferimento alla stessa lista `L`

[18]:
```
X = L
Y = L[:]
```

Modifico la lista `L`

[19]:
```
L[2] = 'DUE'
```

La lista `X` è cambiata (perché di fatto è la lista `L`). La lista `Y` non è stata toccata.

[20]:
```
print(X)
print(Y)
```

```
['zero', 'uno', 'DUE', 'tre', 'quattro', 'cinque']
['zero', 'uno', 'due', 'tre', 'quattro', 'cinque']
```

### 0.1.2 Sequenza di numeri progressivi: `range`

In Python esiste un tipo di sequenza speciale, che serve a rappresentare gli intervalli di numeri interi.

[21]:
```
X = range(10)
print("La lunghezza di X è",len(X))
```

```
La lunghezza di X è 10
```

[22]:
```
print(X[0])
print(X[1])
print(X[9])
```

```
0
1
9
```

[23]:
```
# X non è né una tupla, né una lista
print(X)
```

```
range(0, 10)
```

[24]:
```
# Ma può essere convertita
print(list(X))
print(tuple(X))
```

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

Quindi `range(N)` produce la sequenza dei numeri interi non negativi fino a `N`, escluso. È anche possibile definire la sequenza per un intervallo che inizi da un numero diverso da `0`. La sintassi `range(a,b)` crea un intervallo da `a` (incluso) a `b` (escluso).

[25]:
```
X = range(-3,13)
l = len(X)
print("La lunghezza di X è",l)
```

```
La lunghezza di X è 16
```

[27]:
```
print(X[0])
print(X[1])
print(X[l-1])
```

```
-3
-2
12
```

[28]:
```
# X non è né una tupla, né una lista
print(X)
```

```
range(-3, 13)
```

[29]:
```
# Ma può essere convertita
print(list(X))
print(tuple(X))
```

```
[-3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
(-3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12)
```

La sequenza di tipo `range` sarà molto utile per i cicli `for`.

### 0.1.3 Ricerca nelle sequenze. Operatore `in` e metodo `index`.

Possiamo effettuare una ricerca in una sequenza con l'operatore `in`.

[30]:
```
x = [1,4,7,9]
print( 10 in x)
print(  9 in x)
```

```
False
True
```

[31]:
```
y = ("gatto","cane","delfino")

if "cavallo" in y:
    print("Il mio animale preferito")
elif "gatto" in y:
    print("Mi piacciono i gatti")
else:
    print("Che peccato")
```

```
Mi piacciono i gatti
```

L'operatore `in` dà luogo ad una importante **differenza** tra le stringhe e le altre sequenze. L'espressione booleana `x in S` si comporta diversamente se `S` è una stringa: restituisce `True` quando `x` è una **sottostringa** di `S`, e `False` altrimenti.

[32]:
```
S = "un bel testo evocativo"
L = list(S)
print(S)
print(L)
```

[35]:
```
print("bel testo" in S)
print("bel testo" in L)
```

```
True
False
```

[34]:
```
print("o" in S)
print("o" in L)
```

```
True
True
```

L'operatore `in` ci dice se un elemento (o una sottostringa) è nella sequenza, ma non ci dice in che posizione. Per questo possiamo usare il metodo `index`.

[36]:
```
S = "un bel testo evocativo"
L = list(S)
```

[37]:
```
print(S)
print(L)
```

[38]:
```
print(S.index("bel testo"))
print(S.index("o"))
print(L.index("o"))
```

```
3
11
11
```

**Esercizio:** Che succede se la ricerca effettuata col metodo index non va a buon fine, ovvero si cerca qualcosa che non è nella sequenza?

### 0.1.4 Riassunto

Abbiamo visto - che liste, tuple e stringhe sono tutte **sequenze**; - che si possono copiare **segmenti** di sequenze; - come fare una **copia di una lista**; - il tipo `range`, che produce sequenze di numeri interi progressivi; - la **ricerca** di un elemento in una sequenza; - la ricerca di una **sottostringa**.