liste

June 5, 2023

0.1 Liste

La lista è la struttura dati più importante in Python, e serve a contenere una sequenza di valori. La sintassi è simile a quella delle tuple, ma invece delle parentesi tonde si usano le quadre.

Anche per le liste possiamo calcolarne la lunghezza con `len`, possiamo indicizzarle, concatenarle, ripeterle.

Ricordate che le posizioni sono indicizzate a partire da `0`!

Pertanto una lista `x=["A","B","C"]` ha lunghezza `3`, la posizione `0` contiene `"A"` e la posizione `2` contiene `"C"`.

[1]:
```
x = [ 2, 4, 5, "stringa", 3.4]

print(x)
print(len(x))
```

```
[2, 4, 5, 'stringa', 3.4]
5
```

[2]:
```
print(x[0])
print(x[1])
print(x[2])
print(x[3])
print(x[4])
```

```
2
4
5
stringa
3.4
```

[5]:
```
print ( 3 * x)
```

```
[2, 4, 5, 'stringa', 3.4, 2, 4, 5, 'stringa', 3.4, 2, 4, 5, 'stringa', 3.4]
```

[4]:
```
print( x + [1,1,1] )
```

```
[2, 4, 5, 'stringa', 3.4, 1, 1, 1]
```

0.1.1 Ma sono come le tuple! Allora a che servono le liste?

Le liste possono **essere modificate**! Al contrario dei numeri, delle stringhe, dei booleani e **delle tuple**, la lista è la primo tipo di dato che vediamo che può essere modificato. Si può

- modificarne i valore nelle singole posizioni;
- inserire o eliminare un valore alla fine con `append` e `pop`;
- inserire o eliminate un valore alla posizione i-esima con `insert` e `pop`;.

[7]:
```
x = ["a","casa",21,"trenta",5]
print(x)
```

```
['a', 'casa', 21, 'trenta', 5]
```

[8]:
```
# Modifica di un valore
x[2] = 3.6
print(x)
```

```
['a', 'casa', 3.6, 'trenta', 5]
```

[6]:
```
# Elimina l'ultimo valore
x.pop()
print(x)
```

[6]:
```
# Aggiunge un valore in coda
x.append("gatto")
print(x)
```

[9]:
```
# Inserisce valori (traslando i seguenti verso destra)
x.insert(1,"A")
print(x)
x.insert(3,"B")
print(x)
```

```
['a', 'A', 'casa', 3.6, 'trenta', 5]
['a', 'A', 'casa', 'B', 3.6, 'trenta', 5]
```

[14]:
```
# Elimina valori (traslando i seguenti verso sinistra)
x.pop(2)
print(x)
```

```
['A', 'casa']
```

[10]:
```
x.pop(0)
print(x)
```

```
['A', 'casa', 'B', 3.6, 'trenta', 5]
```

e ancora… - rovesciare la lista con `reverse`; - aggiungere tanti valori con `extend`.

[19]:
```
x = ["a","casa",21,"trenta",5]
print(x)
```

```
['a', 'casa', 21, 'trenta', 5]
```

[20]:
```
x.reverse()
print(x)
```

```
[5, 'trenta', 21, 'casa', 'a']
```

[21]:
```
x.extend([1,2,3,3,2,1])
print(x)
```

```
[5, 'trenta', 21, 'casa', 'a', 1, 2, 3, 3, 2, 1]
```

Fate attenzione alla differenza tra `append` e `extend`.

[22]:
```
x = [1,2,3,4]
y = [1,2,3,4]
```

[23]:
```
x.append([5,6,7,8])
print(x)
print(len(x))
```

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

[24]:
```
y.extend([5,6,7,8])
print(y)
print(len(y))
```

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

0.1.2 La lista vuota

Una lista è vuota se ha lunghezza zero, e si denota `[]`.

[25]:
```
x = []
print(x)
print(len(x))
```

```
[]
0
```

0.1.3 Riferimenti multipli alla stessa lista

Osservate il seguente codice

[27]:
```
x = ['a','b','c']
y = x
print(x)
print(y)
```

```
['a', 'b', 'c']
['a', 'b', 'c']
```

[28]:
```
x[0] = 'A'
print(x)
print(y)
```

```
['A', 'b', 'c']
['A', 'b', 'c']
```

Abbiamo modificato solamente `x`, ma **anche** la lista `y` è cambiata! Come mai?

In realtà l'assegnamento `y = x` non copia la lista in `x` in `y`, ma copia il suo **riferimento**. Non entriamo nei dettagli tecnici, basti sapere che adesso abbiamo due variabili che si riferiscono alla stessa lista, e che quindi ogni modifica fatta ad `x` verrà osservata anche in `y`.

Accade qualcosa di simile anche per tuple e stringhe, ma poiché questi tipi di dati non possono essere modificati, non ci sono sorprese di questo tipo.

0.1.4 Ordinare una lista

Quando una lista è costituita da tutti elementi confrontabili tra loro, allora la funzione predefinita `sorted` può ordinarla.

Notate che la lista originale non viene modificata: `sorted` ne restituisce una nuova. Se invece vogliamo che sia la lista originale ad essere modificata, dobbiamo usare il metodo `sort`. Il metodo `sort` non restituisce valore ma ordina la lista su cui viene applicato.

[29]:
```
x = ["casa", "gatto", "abaco"]
y = sorted(x)
print(x)
print(y)
```

```
['casa', 'gatto', 'abaco']
['abaco', 'casa', 'gatto']
```

[30]:
```
x.sort()
print(x)
```

```
['abaco', 'casa', 'gatto']
```

Esercizio: si possono convertire trasformare alcuni tipi di dati in un lista con la funzione di conversione `list`, simile a come abbiamo fatto per altre conversioni. Che succede se a `list` diamo come argomento - un numero intero, - una stringa, - una tupla?

0.1.5 Riassumendo

Abbiamo visto - il tipo di dato **lista**; - come si **crea** una lista; - le operazioni principali sulla lista, **comuni anche a tuple e stringhe**; - varie operazioni che **modificano** la lista; - l'effetto di **multipli riferimenti** ad una lista; - come **ordinare** liste con `sorted` e il metodo `sort`.