Laboratorio 02

Programmazione - CdS Matematica

Marco Virgulin
4 Novembre 2014

Università degli Studi di Padova

Spesso è necessario utilizzare *contenitori* di oggetti.

Spesso è necessario utilizzare *contenitori* di oggetti.
Per esempio per creare elenchi di oggetti o informazione strutturata.

Spesso è necessario utilizzare *contenitori* di oggetti.
Per esempio per creare elenchi di oggetti o informazione strutturata.
I contenitori sono quindi strutture dati che raggruppano altri oggetti:

Spesso è necessario utilizzare *contenitori* di oggetti.
Per esempio per creare elenchi di oggetti o informazione strutturata.
I contenitori sono quindi strutture dati che raggruppano altri oggetti:

- Tuple

```
>>> data_nascita=15,"Aprile",1452
>>> # oppure:
>>> data_nascita=(15,"Aprile",1452)
```

- Liste

```
>>> lista=[123, "abcd", 1.84, (2.0,"lista")]
```

- Dizionari
- Insiemi

Una tupla è una sequenza ordinata di oggetti **non sostituibili**.

Una tupla è una sequenza ordinata di oggetti **non sostituibili**.

Una lista rappresenta una sequenza ordinata di oggetti (anche eterogenei).

Una tupla è una sequenza ordinata di oggetti **non sostituibili**.

Una lista rappresenta una sequenza ordinata di oggetti (anche eterogenei).
La lista è mutabile:

- gli oggetti di una lista possono essere sostituiti,
- la lunghezza di una lista puó essere modificata aggiungendo o rimuovendo elementi.

Esercizio Svolto

Siano:

```
>>> v1 = 1
>>> v2 = 'a'
```

Usare una tupla per scambiare i valori in modo rapido.

Esercizio Svolto

Siano:

```
>>> v1 = 1
>>> v2 = 'a'
```

Usare una tupla per scambiare i valori in modo rapido.

```
>>> v1,v2 = v2,v1
```

Esercizio Svolto

Siano:

```
>>> v1 = 1
>>> v2 = 'a'
```

Usare una tupla per scambiare i valori in modo rapido.

```
>>> v1,v2 = v2,v1
>>> v1
```

Esercizio Svolto

Siano:

```
>>> v1 = 1
>>> v2 = 'a'
```

Usare una tupla per scambiare i valori in modo rapido.

```
>>> v1,v2 = v2,v1
>>> v1
'a'
```

Esercizio Svolto

Siano:

```
>>> v1 = 1
>>> v2 = 'a'
```

Usare una tupla per scambiare i valori in modo rapido.

```
>>> v1,v2 = v2,v1
>>> v1
'a'
>>> v2
```

Esercizio Svolto

Siano:

```
>>> v1 = 1
>>> v2 = 'a'
```

Usare una tupla per scambiare i valori in modo rapido.

```
>>> v1,v2 = v2,v1
>>> v1
'a'
>>> v2
1
```

Operazioni sulle sequenze

Per tutte le sequenze (liste, stringhe e tuple) è possibile:

Operazioni sulle sequenze

Per tutte le sequenze (liste, stringhe e tuple) è possibile:

- accedere agli elementi di una lista tramite l'operatore di indicizzazione [],

```
>>> lista[1]
```

Operazioni sulle sequenze

Per tutte le sequenze (liste, stringhe e tuple) è possibile:

- accedere agli elementi di una lista tramite l'operatore di indicizzazione [],

```
>>> lista[1]
```

- ottenere una sottolista di una lista attraverso l'operatore di slicing [begin:end] o striding [begin:end:step],

```
>>> lista[1:3]
>>> lista[1::2]
```

Operazioni sulle sequenze

Per tutte le sequenze (liste, stringhe e tuple) è possibile:

- accedere agli elementi di una lista tramite l'operatore di indicizzazione [],

```
>>> lista[1]
```

- ottenere una sottolista di una lista attraverso l'operatore di slicing [begin:end] o striding [begin:end:step],

```
>>> lista[1:3]
>>> lista[1::2]
```

- concatenare due liste ottenendo una nuova lista (operatore +),

```
>>> lista+lista
```

Operazioni sulle sequenze

Per tutte le sequenze (liste, stringhe e tuple) è possibile:

- accedere agli elementi di una lista tramite l'operatore di indicizzazione [],

```
>>> lista[1]
```

- ottenere una sottolista di una lista attraverso l'operatore di slicing [begin:end] o striding [begin:end:step],

```
>>> lista[1:3]
>>> lista[1::2]
```

- concatenare due liste ottenendo una nuova lista (operatore +),

```
>>> lista+lista
```

- generare un oggetto lista in cui si ripetono n volte gli elementi di una lista (operatore *),

```
>>> lista*2
```

Operazioni sulle liste

Per tutte le liste è possibile:

Operazioni sulle liste

Per tutte le liste è possibile:

- aggiungere elementi,

```
>>> elle = ['a',1,('b',3+1j)] # lista di esempio
>>> elle.append('nuovo')      # aggiunge un elemento alla fine
>>> elle.extend([1,2])        # concatena un'altra lista alla fine
```

Operazioni sulle liste

Per tutte le liste è possibile:

- aggiungere elementi,

```
>>> elle = ['a',1,('b',3+1j)] # lista di esempio
>>> elle.append('nuovo')      # aggiunge un elemento alla fine
>>> elle.extend([1,2])        # concatena un'altra lista alla fine
```

- invertire l'ordine degli elementi,

```
>>> elle.reverse()            # inverte l'ordine degli elementi
```

Operazioni sulle liste

Per tutte le liste è possibile:

- aggiungere elementi,

```
>>> elle = ['a',1,('b',3+1j)] # lista di esempio
>>> elle.append('nuovo')      # aggiunge un elemento alla fine
>>> elle.extend([1,2])        # concatena un'altra lista alla fine
```

- invertire l'ordine degli elementi,

```
>>> elle.reverse()            # inverte l'ordine degli elementi
```

- inserire un elemento in un certo indice,

```
>>> elle.insert(1,'inserito') # inserisce nell'indice 1
```

Operazioni sulle liste

Per tutte le liste è possibile:

- aggiungere elementi,

```
>>> elle = ['a',1,('b',3+1j)] # lista di esempio
>>> elle.append('nuovo')      # aggiunge un elemento alla fine
>>> elle.extend([1,2])        # concatena un'altra lista alla fine
```

- invertire l'ordine degli elementi,

```
>>> elle.reverse()            # inverte l'ordine degli elementi
```

- inserire un elemento in un certo indice,

```
>>> elle.insert(1,'inserito') # inserisce nell'indice 1
```

- eliminare elementi,

```
>>> del elle[5]               # elimina l'elemento di indice 5
>>> elle
[2,'inserito',1,'nuovo',('b',3+1j),'a']
```

Operazioni sulle liste

Per tutte le liste è possibile:

- aggiungere elementi,

```
>>> elle = ['a',1,('b',3+1j)] # lista di esempio
>>> elle.append('nuovo')      # aggiunge un elemento alla fine
>>> elle.extend([1,2])        # concatena un'altra lista alla fine
```

- invertire l'ordine degli elementi,

```
>>> elle.reverse()            # inverte l'ordine degli elementi
```

- inserire un elemento in un certo indice,

```
>>> elle.insert(1,'inserito') # inserisce nell'indice 1
```

- eliminare elementi,

```
>>> del elle[5]               # elimina l'elemento di indice 5
>>> elle
[2,'inserito',1,'nuovo',('b',3+1j),'a']
```

Esercizio

Data a = [1,2,3] si modifichi a in modo da ottenere a = [4,2,3].

Esercizio

Data a = [1,2,3] si modifichi a in modo da ottenere a = [4,2,3].

```
>>> a = [1,2,3]
```

Esercizio

Data a = [1,2,3] si modifichi a in modo da ottenere a = [4,2,3].

```
>>> a = [1,2,3]
>>> a[0] = 4
```

Esercizio

Data a = [1,2,3] si modifichi a in modo da ottenere a = [4,2,3].

```
>>> a = [1,2,3]
>>> a[0] = 4
>>> a
```

Esercizio

Data a = [1,2,3] si modifichi a in modo da ottenere a = [4,2,3].

```
>>> a = [1,2,3]
>>> a[0] = 4
>>> a
>>> [4, 2, 3]
```

Esercizio

Data a = [1,2,3,4,5], utilizzare l'operatore di slicing e l'assegnazione in modo da ottenere a = [3,4,5].

Esercizio

Data a = [1,2,3,4,5], utilizzare l'operatore di slicing e l'assegnazione in modo da ottenere a = [3,4,5].

```
>>> a = [1,2,3,4,5]
```

Esercizio

Data a = [1,2,3,4,5], utilizzare l'operatore di slicing e l'assegnazione in modo da ottenere a = [3,4,5].

```
>>> a = [1,2,3,4,5]
>>> a = a[2:]
```

Esercizio

Data a = [1,2,3,4,5], utilizzare l'operatore di slicing e l'assegnazione in modo da ottenere a = [3,4,5].

```
>>> a = [1,2,3,4,5]
>>> a = a[2:]
>>> a
```

Esercizio

Data a = [1,2,3,4,5], utilizzare l'operatore di slicing e l'assegnazione in modo da ottenere a = [3,4,5].

```
>>> a = [1,2,3,4,5]
>>> a = a[2:]
>>> a
>>> [3, 4, 5]
```

Operatore di formato per le stringhe: un operatore per generare una stringa con un formato dato.

Operatori di formato

Operatore di formato per le stringhe: un operatore per generare una stringa con un formato dato. Un primo esempio:

```
>>> "I numeri %d e %d sono multipli di %d"%(8,4,2)
```

Operatori di formato

Operatore di formato per le stringhe: un operatore per generare una stringa con un formato dato. Un primo esempio:

```
>>> "I numeri %d e %d sono multipli di %d"%(8,4,2)
'I numeri 8 e 4 sono multipli di 2'
```

Operatori di formato

Operatore di formato per le stringhe: un operatore per generare una stringa con un formato dato. Un primo esempio:

```
>>> "I numeri %d e %d sono multipli di %d"%(8,4,2)
'I numeri 8 e 4 sono multipli di 2'
```

Ecco alcuni tipi utili:

- **Interi**: carattere (c), ottale (o), decimale (d), esadecimale (x, X), ...

Operatori di formato

Operatore di formato per le stringhe: un operatore per generare una stringa con un formato dato. Un primo esempio:

```
>>> "I numeri %d e %d sono multipli di %d"%(8,4,2)
'I numeri 8 e 4 sono multipli di 2'
```

Ecco alcuni tipi utili:

- **Interi**: carattere (c), ottale (o), decimale (d), esadecimale (x, X), ...
- **Float**: notazione esponenziale (e, E), fixed point (f), ...

Operatori di formato

Operatore di formato per le stringhe: un operatore per generare una stringa con un formato dato. Un primo esempio:

```
>>> "I numeri %d e %d sono multipli di %d"%(8,4,2)
'I numeri 8 e 4 sono multipli di 2'
```

Ecco alcuni tipi utili:

- **Interi**: carattere (c), ottale (o), decimale (d), esadecimale (x, X), ...
- **Float**: notazione esponenziale (e, E), fixed point (f), ...

```
>>> x = 0.123456789
>>> "Il valore di x troncato: %.1f  %.5f  %.9f" % (x,x,x)
```

Operatori di formato

Operatore di formato per le stringhe: un operatore per generare una stringa con un formato dato. Un primo esempio:

```
>>> "I numeri %d e %d sono multipli di %d"%(8,4,2)
'I numeri 8 e 4 sono multipli di 2'
```

Ecco alcuni tipi utili:

- **Interi**: carattere (c), ottale (o), decimale (d), esadecimale (x, X), ...
- **Float**: notazione esponenziale (e, E), fixed point (f), ...

```
>>> x = 0.123456789
>>> "Il valore di x troncato: %.1f  %.5f  %.9f" % (x,x,x)
'Il valore di x troncato: 0.1 0.12346 0.123456789'
```

Operatori di formato

Operatore di formato per le stringhe: un operatore per generare una stringa con un formato dato. Un primo esempio:

```
>>> "I numeri %d e %d sono multipli di %d"%(8,4,2)
'I numeri 8 e 4 sono multipli di 2'
```

Ecco alcuni tipi utili:

- **Interi**: carattere (c), ottale (o), decimale (d), esadecimale (x, X), ...
- **Float**: notazione esponenziale (e, E), fixed point (f), ...

```
>>> x = 0.123456789
>>> "Il valore di x troncato: %.1f  %.5f  %.9f" % (x,x,x)
'Il valore di x troncato: 0.1 0.12346 0.123456789'
```

- **Stringa**: rappresentazione data da str (s).

Esercizio svolto

Dato x=42, visualizzarlo usando gli operatori di formato: carattere, decimale, ottale e esadecimale.

Esercizio svolto

Dato x=42, visualizzarlo usando gli operatori di formato: carattere, decimale, ottale e esadecimale.

```
>>> x = 42
>>> "%c %d %o %x" % (x,x,x,x)
```

Esercizio svolto

Dato x=42, visualizzarlo usando gli operatori di formato: carattere, decimale, ottale e esadecimale.

```
>>> x = 42
>>> "%c %d %o %x" % (x,x,x,x)
'* 42 52 2a'
```

Esercizio

Importare il modulo math e mettere in P il valore di pigreco (math.pi)

Esercizio

Importare il modulo math e mettere in P il valore di pigreco (math.pi)

```
>>> import math
```

Esercizio

Importare il modulo math e mettere in P il valore di pigreco (math.pi)

```
>>> import math
>>> P = math.pi
```

Esercizio

Importare il modulo math e mettere in P il valore di pigreco (math.pi)

```
>>> import math
>>> P = math.pi
>>> P
3.141592653589793
```

Esercizio

Importare il modulo math e mettere in P il valore di pigreco (math.pi)

```
>>> import math
>>> P = math.pi
>>> P
3.141592653589793
```

Stampare il valore di P con 2,3 e 4 decimali. (Come avviene il taglio dei decimali?)

Esercizio

Importare il modulo math e mettere in P il valore di pigreco (math.pi)

```
>>> import math
>>> P = math.pi
>>> P
3.141592653589793
```

Stampare il valore di P con 2,3 e 4 decimali. (Come avviene il taglio dei decimali?)

```
>>> "Pigreco con 2 decimali: %.2f" % P
'Pigreco con due decimali: 3.14'
>>> "Pigreco con 3 decimali: %.3f" % P
'Pigreco con due decimali: 3.142'
>>> "Pigreco con 4 decimali: %.4f" % P
'Pigreco con due decimali: 3.1416'
```

Range

Un utile comando per generare una lista di valori è il comando *range(stop)*, *range(start, stop)*, *range(start, stop, step)*.

Un utile comando per generare una lista di valori è il comando *range(stop)*, *range(start, stop)*, *range(start, stop, step)*.

```
>>> range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Range

Un utile comando per generare una lista di valori è il comando *range(stop)*, *range(start, stop)*, *range(start, stop, step)*.

```
>>> range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> range(6,10)
[6, 7, 8, 9]
```

Range

Un utile comando per generare una lista di valori è il comando *range(stop)*, *range(start, stop)*, *range(start, stop, step)*.

```
>>> range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> range(6,10)
[6, 7, 8, 9]
>>> range(6,10,2)
[6, 8]
```

Esercizio

Generare la lista di tutti i numeri multipli di 7 compresi tra 21 e 56 inclusi.

Esercizio

Generare la lista di tutti i numeri multipli di 7 compresi tra 21 e 56 inclusi.

```
>>> range(21,57,7)
```

Esercizio

Generare la lista di tutti i numeri multipli di 7 compresi tra 21 e 56 inclusi.

```
>>> range(21,57,7)
[21, 28, 35, 42, 49, 56]
```

Esercizio

Generare la lista di tutti i numeri multipli di 7 compresi tra 21 e 56 inclusi.

```
>>> range(21,57,7)
[21, 28, 35, 42, 49, 56]
```

Generare la lista di tutti i numeri pari multipli di 13 compresi tra 10 e 99 inclusi.

Esercizio

Generare la lista di tutti i numeri multipli di 7 compresi tra 21 e 56 inclusi.

```
>>> range(21,57,7)
[21, 28, 35, 42, 49, 56]
```

Generare la lista di tutti i numeri pari multipli di 13 compresi tra 10 e 99 inclusi.

```
>>> range(13,100,13)
```

Esercizio

Generare la lista di tutti i numeri multipli di 7 compresi tra 21 e 56 inclusi.

```
>>> range(21,57,7)
[21, 28, 35, 42, 49, 56]
```

Generare la lista di tutti i numeri pari multipli di 13 compresi tra 10 e 99 inclusi.

```
>>> range(13,100,13)
[13, 26, 39, 52, 65, 78, 91]
```

Esercizio

Generare la lista di tutti i numeri multipli di 7 compresi tra 21 e 56 inclusi.

```
>>> range(21,57,7)
[21, 28, 35, 42, 49, 56]
```

Generare la lista di tutti i numeri pari multipli di 13 compresi tra 10 e 99 inclusi.

```
>>> range(13,100,13)
[13, 26, 39, 52, 65, 78, 91]
>>> range(13,100,13)[1::2]
```

Esercizio

Generare la lista di tutti i numeri multipli di 7 compresi tra 21 e 56 inclusi.

```
>>> range(21,57,7)
[21, 28, 35, 42, 49, 56]
```

Generare la lista di tutti i numeri pari multipli di 13 compresi tra 10 e 99 inclusi.

```
>>> range(13,100,13)
[13, 26, 39, 52, 65, 78, 91]
>>> range(13,100,13)[1::2]
[26, 52, 78]
```

Immutabili?


```
>>> h = ['uno', 'due']
>>> t = (h[0], h[1], h)
>>> t
```

Immutabili?


```
>>> h = ['uno', 'due']
>>> t = (h[0], h[1], h)
>>> t
('uno', 'due', ['uno', 'due'])
```

Immutabili?

```
>>> h = ['uno', 'due']
>>> t = (h[0], h[1], h)
>>> t
('uno', 'due', ['uno', 'due'])
>>> t[0] = 1
```

Immutabili?


```
>>> h = ['uno', 'due']
>>> t = (h[0], h[1], h)
>>> t
('uno', 'due', ['uno', 'due'])
>>> t[0] = 1
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

Immutabili?

```
>>> h = ['uno', 'due']
>>> t = (h[0], h[1], h)
>>> t
('uno', 'due', ['uno', 'due'])
>>> t[0] = 1
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>> h[0] = 3
>>> h[1] = 4
>>> t
```

Immutabili?


```
>>> h = ['uno', 'due']
>>> t = (h[0], h[1], h)
>>> t
('uno', 'due', ['uno', 'due'])
>>> t[0] = 1
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>> h[0] = 3
>>> h[1] = 4
>>> t
('uno', 'due', [3, 4])
```

Esercizio

Dati:

```
>>> x = 1
>>> y = 0
>>> t = (x,x+1,x+3,[y,3])
```

Esercizio

Dati:

```
>>> x = 1
>>> y = 0
>>> t = (x,x+1,x+3,[y,3])
```

Si eseguano le seguenti istruzioni:

```
>>> x = x * 4
>>> t[3][1] = x
>>> t[3].extend([x,y])
>>> y = 2
>>> t[3][3] += 1
```

Esercizio

Dati:

```
>>> x = 1
>>> y = 0
>>> t = (x,x+1,x+3,[y,3])
```

Si eseguano le seguenti istruzioni:

```
>>> x = x * 4
>>> t[3][1] = x
>>> t[3].extend([x,y])
>>> y = 2
>>> t[3][3] += 1
```

Nel caso in cui l'esecuzione vada a buon fine, dire come è composta la tupla t.

Esercizio

Dati:

```
>>> x = 1
>>> y = 0
>>> t = (x,x+1,x+3,[y,3])
```

Si eseguano le seguenti istruzioni:

```
>>> x = x * 4
>>> t[3][1] = x
>>> t[3].extend([x,y])
>>> y = 2
>>> t[3][3] += 1
```

Nel caso in cui l'esecuzione vada a buon fine, dire come è composta la tupla t.

```
>>> t
```

Esercizio

Dati:

```
>>> x = 1
>>> y = 0
>>> t = (x,x+1,x+3,[y,3])
```

Si eseguano le seguenti istruzioni:

```
>>> x = x * 4
>>> t[3][1] = x
>>> t[3].extend([x,y])
>>> y = 2
>>> t[3][3] += 1
```

Nel caso in cui l'esecuzione vada a buon fine, dire come è composta la tupla t.

```
>>> t
(1, 2, 4, [0, 4, 4, 1])
```

Liste e stringhe: split() e join()

Date le stringhe *s* e *sep*, *s.split(sep)* ritorna la lista delle parole presenti in *s* usando *sep* come separatore.

Liste e stringhe: split() e join()

Date le stringhe *s* e *sep*, *s.split(sep)* ritorna la lista delle parole presenti in *s* usando *sep* come separatore.

```
>>> s = "manca poco alla fine della lezione\n molto alla fine del corso"
```

Liste e stringhe: split() e join()

Date le stringhe *s* e *sep*, *s.split(sep)* ritorna la lista delle parole presenti in *s* usando *sep* come separatore.

```
>>> s = "manca poco alla fine della lezione\n molto alla fine del corso"
>>> s.split() # separatori impliciti: " " e "\n"
```

Liste e stringhe: split() e join()

Date le stringhe *s* e *sep*, *s.split(sep)* ritorna la lista delle parole presenti in *s* usando *sep* come separatore.

```
>>> s = "manca poco alla fine della lezione\n molto alla fine del corso"
>>> s.split() # separatori impliciti: " " e "\n"
['manca', 'poco', 'alla', 'fine', 'della', 'lezione', 'molto', 'alla', '
    fine', 'del', 'corso']
```

Liste e stringhe: split() e join()

Date le stringhe *s* e *sep*, *s.split(sep)* ritorna la lista delle parole presenti in *s* usando *sep* come separatore.

```
>>> s = "manca poco alla fine della lezione\n molto alla fine del corso"
>>> s.split() # separatori impliciti: " " e "\n"
['manca', 'poco', 'alla', 'fine', 'della', 'lezione', 'molto', 'alla', '
    fine', 'del', 'corso']
>>> s.split("\n")
```

Liste e stringhe: split() e join()

Date le stringhe *s* e *sep*, *s.split(sep)* ritorna la lista delle parole presenti in *s* usando *sep* come separatore.

```
>>> s = "manca poco alla fine della lezione\n molto alla fine del corso"
>>> s.split() # separatori impliciti: " " e "\n"
['manca', 'poco', 'alla', 'fine', 'della', 'lezione', 'molto', 'alla', '
    fine', 'del', 'corso']
>>> s.split("\n")
['manca poco alla fine della lezione', ' molto alla fine del corso']
```

Liste e stringhe: split() e join()

Date le stringhe *s* e *sep*, *s.split(sep)* ritorna la lista delle parole presenti in *s* usando *sep* come separatore.

```
>>> s = "manca poco alla fine della lezione\n molto alla fine del corso"
>>> s.split() # separatori impliciti: " " e "\n"
['manca', 'poco', 'alla', 'fine', 'della', 'lezione', 'molto', 'alla', '
    fine', 'del', 'corso']
>>> s.split("\n")
['manca poco alla fine della lezione', ' molto alla fine del corso']
>>> s.split('m')
```

Liste e stringhe: split() e join()

Date le stringhe *s* e *sep*, *s.split(sep)* ritorna la lista delle parole presenti in *s* usando *sep* come separatore.

```
>>> s = "manca poco alla fine della lezione\n molto alla fine del corso"
>>> s.split() # separatori impliciti: " " e "\n"
['manca', 'poco', 'alla', 'fine', 'della', 'lezione', 'molto', 'alla', '
    fine', 'del', 'corso']
>>> s.split("\n")
['manca poco alla fine della lezione', ' molto alla fine del corso']
>>> s.split('m')
['', 'anca poco alla fine della lezione\n ', 'olto alla fine del corso']
```

Liste e stringhe: split() e join()

La funzionalità inversa è ottenuta con il metodo delle stringhe join() che unisce una lista di stringhe.

Liste e stringhe: split() e join()

La funzionalità inversa è ottenuta con il metodo delle stringhe join() che unisce una lista di stringhe.
Anche se puó sembrare controintuitivo, il metodo join() è un metodo della classe str e **va applicato alla stringa che contiene il separatore**.

Liste e stringhe: split() e join()

La funzionalità inversa è ottenuta con il metodo delle stringhe join() che unisce una lista di stringhe.
Anche se puó sembrare controintuitivo, il metodo join() è un metodo della classe str e **va applicato alla stringa che contiene il separatore**.

```
>>> "".join(['a','b','c']) # senza separatore
```

Liste e stringhe: split() e join()

La funzionalità inversa è ottenuta con il metodo delle stringhe join() che unisce una lista di stringhe.
Anche se puó sembrare controintuitivo, il metodo join() è un metodo della classe str e **va applicato alla stringa che contiene il separatore**.

```
>>> "".join(['a','b','c']) # senza separatore
'abc'
```

Liste e stringhe: split() e join()

La funzionalità inversa è ottenuta con il metodo delle stringhe join() che unisce una lista di stringhe.
Anche se puó sembrare controintuitivo, il metodo join() è un metodo della classe str e **va applicato alla stringa che contiene il separatore**.

```
>>> "".join(['a','b','c']) # senza separatore
'abc'
>>> " ".join(['a','b','c']) # separatore spazio
```

Liste e stringhe: split() e join()

La funzionalità inversa è ottenuta con il metodo delle stringhe join() che unisce una lista di stringhe.
Anche se puó sembrare controintuitivo, il metodo join() è un metodo della classe str e **va applicato alla stringa che contiene il separatore**.

```
>>> "".join(['a','b','c']) # senza separatore
'abc'
>>> " ".join(['a','b','c']) # separatore spazio
'a b c'
```

Liste e stringhe: split() e join()

La funzionalità inversa è ottenuta con il metodo delle stringhe join() che unisce una lista di stringhe.
Anche se puó sembrare controintuitivo, il metodo join() è un metodo della classe str e **va applicato alla stringa che contiene il separatore**.

```
>>> "".join(['a','b','c']) # senza separatore
'abc'
>>> " ".join(['a','b','c']) # separatore spazio
'a b c'
>>> "\n".join(['a','b','c']) # separatore termine riga
```

Liste e stringhe: split() e join()

La funzionalità inversa è ottenuta con il metodo delle stringhe join() che unisce una lista di stringhe.
Anche se puó sembrare controintuitivo, il metodo join() è un metodo della classe str e **va applicato alla stringa che contiene il separatore**.

```
>>> "".join(['a','b','c']) # senza separatore
'abc'
>>> " ".join(['a','b','c']) # separatore spazio
'a b c'
>>> "\n".join(['a','b','c']) # separatore termine riga
'a\nb\nc'
```

Esercizio

Dati:

```
>>> s = "long a what remember never I"
>>> l = ['sentence','I made','up']
```

Ottenere: "I * never * remember * what * a * long * sentence * I made * up".
Suggerimento: usare split, reverse e join.

Esercizio

Dati:

```
>>> s = "long a what remember never I"
>>> l = ['sentence','I made','up']
```

Ottenere: "I * never * remember * what * a * long * sentence * I made * up".
Suggerimento: usare split, reverse e join.

```
>>> l1 = s.split()
```

Esercizio

Dati:

```
>>> s = "long a what remember never I"
>>> l = ['sentence','I made','up']
```

Ottenere: "I * never * remember * what * a * long * sentence * I made * up".
Suggerimento: usare split, reverse e join.

```
>>> l1 = s.split()
>>> l1.reverse()
```

Esercizio

Dati:

```
>>> s = "long a what remember never I"
>>> l = ['sentence','I made','up']
```

Ottenere: "I * never * remember * what * a * long * sentence * I made * up".
Suggerimento: usare split, reverse e join.

```
>>> l1 = s.split()
>>> l1.reverse()
>>> s = l1+l
```

Esercizio

Dati:

```
>>> s = "long a what remember never I"
>>> l = ['sentence','I made','up']
```

Ottenere: "I * never * remember * what * a * long * sentence * I made * up".
Suggerimento: usare split, reverse e join.

```
>>> l1 = s.split()
>>> l1.reverse()
>>> s = l1+l
>>> s = " * ".join(s)
```

Esercizio

Dati:

```
>>> s = "long a what remember never I"
>>> l = ['sentence','I made','up']
```

Ottenere: "I * never * remember * what * a * long * sentence * I made * up".
Suggerimento: usare split, reverse e join.

```
>>> l1 = s.split()
>>> l1.reverse()
>>> s = l1+l
>>> s = " * ".join(s)
>>> 'I * never * remember * what * a * long * sentence * I made * up'
```

Esercizio

Siano:

```
>>> s1 = "bu"
>>> s2 = "se"
>>> s3 = "te"
```

Usando s1, s2 e s3 comporre la stringa **bubusettete**.

Esercizio

Siano:

```
>>> s1 = "bu"
>>> s2 = "se"
>>> s3 = "te"
```

Usando s1, s2 e s3 comporre la stringa **bubusettete**.

```
>>> s1*2+s2+s3[0]+s3*2
```

Esercizio

Siano:

```
>>> s1 = "bu"
>>> s2 = "se"
>>> s3 = "te"
```

Usando s1, s2 e s3 comporre la stringa **bubusettete**.

```
>>> s1*2+s2+s3[0]+s3*2
'bubusettete'
```

Modifica di sequenze

Per oggetti sequenza di tipo mutabile è possibile anche utilizzare l'indicizzazione, lo slicing e lo striding per modificare delle sotto-sequenze.

Modifica di sequenze

Per oggetti sequenza di tipo mutabile è possibile anche utilizzare l'indicizzazione, lo slicing e lo striding per modificare delle sotto-sequenze.

```
>>> elle = [1,2,3,4,5]
>>> elle[2]=6
>>> elle
```

Modifica di sequenze

Per oggetti sequenza di tipo mutabile è possibile anche utilizzare l'indicizzazione, lo slicing e lo striding per modificare delle sotto-sequenze.

```
>>> elle = [1,2,3,4,5]
>>> elle[2]=6
>>> elle
[1,2,6,4,5]
```

Modifica di sequenze

Per oggetti sequenza di tipo mutabile è possibile anche utilizzare l'indicizzazione, lo slicing e lo striding per modificare delle sotto-sequenze.

```
>>> elle = [1,2,3,4,5]
>>> elle[2]=6
>>> elle
[1,2,6,4,5]
>>> elle[1:3]=[7,8,9]
>>> elle
```

Modifica di sequenze

Per oggetti sequenza di tipo mutabile è possibile anche utilizzare l'indicizzazione, lo slicing e lo striding per modificare delle sotto-sequenze.

```
>>> elle = [1,2,3,4,5]
>>> elle[2]=6
>>> elle
[1,2,6,4,5]
>>> elle[1:3]=[7,8,9]
>>> elle
[1,7,8,9,4,5]
```

Modifica di sequenze

Per oggetti sequenza di tipo mutabile è possibile anche utilizzare l'indicizzazione, lo slicing e lo striding per modificare delle sotto-sequenze.

```
>>> elle = [1,2,3,4,5]
>>> elle[2]=6
>>> elle
[1,2,6,4,5]
>>> elle[1:3]=[7,8,9]
>>> elle
[1,7,8,9,4,5]
>>> elle[1:5:2]=[10,11]
>>> elle
```

Modifica di sequenze

Per oggetti sequenza di tipo mutabile è possibile anche utilizzare l'indicizzazione, lo slicing e lo striding per modificare delle sotto-sequenze.

```
>>> elle = [1,2,3,4,5]
>>> elle[2]=6
>>> elle
[1,2,6,4,5]
>>> elle[1:3]=[7,8,9]
>>> elle
[1,7,8,9,4,5]
>>> elle[1:5:2]=[10,11]
>>> elle
[1,10,8,11,4,5]
```

Assegnazione o copia?

Quando assegnamo (un riferimento ad) una lista ad una variabile l1, id(l1) identifica la lista riferita da l1. L'istruzione:

```
>>> l2 = l1
```

assegna a l2 lo stesso riferimento di l1 (l1 e l2 puntano alla stessa lista).

Assegnazione o copia?

Quando assegnamo (un riferimento ad) una lista ad una variabile l1, id(l1) identifica la lista riferita da l1. L'istruzione:

```
>>> l2 = l1
```

assegna a l2 lo stesso riferimento di l1 (l1 e l2 puntano alla stessa lista).
Se questo non è il comportamento desiderato, possiamo clonare l1 tramite l'operatore di slicing:

```
>>> l2 = l1[:]
```

Ora l2 e l1 si riferiscono a due oggetti list diversi. Possiamo verificarlo confrontando id(l1) e id(l2).

Esercizio

Data la lista a = [1,2,3] creare b a partire da a in modo che, rimuovendo successivamente il secondo elemento di b si abbia a = [1,2,3] e b = [1,3].

Esercizio

Data la lista a = [1,2,3] creare b a partire da a in modo che, rimuovendo successivamente il secondo elemento di b si abbia a = [1,2,3] e b = [1,3].

```
>>> a = [1,2,3]
```

Esercizio

Data la lista a = [1,2,3] creare b a partire da a in modo che, rimuovendo successivamente il secondo elemento di b si abbia a = [1,2,3] e b = [1,3].

```
>>> a = [1,2,3]
>>> b = a[:]
```

Esercizio

Data la lista a = [1,2,3] creare b a partire da a in modo che, rimuovendo successivamente il secondo elemento di b si abbia a = [1,2,3] e b = [1,3].

```
>>> a = [1,2,3]
>>> b = a[:]
>>> del b[1]
```

Esercizio

Data la lista a = [1,2,3] creare b a partire da a in modo che, rimuovendo successivamente il secondo elemento di b si abbia a = [1,2,3] e b = [1,3].

```
>>> a = [1,2,3]
>>> b = a[:]
>>> del b[1]
>>> a,b
```

Esercizio

Data la lista a = [1,2,3] creare b a partire da a in modo che, rimuovendo successivamente il secondo elemento di b si abbia a = [1,2,3] e b = [1,3].

```
>>> a = [1,2,3]
>>> b = a[:]
>>> del b[1]
>>> a,b
>>> ([1, 2, 3], [1, 3])
```

Esercizio

Data la lista l1 = [[7,8,9],[5,5,5],[1,2,3]], costruire e modificare l2 a partire da l1 in modo che

- l1 sia [[7,8,9],[5,5,5],[1,2,3]]
- l2 sia [[1,2,3],[9,8,7]]

Esercizio

Data la lista l1 = [[7,8,9],[5,5,5],[1,2,3]], costruire e modificare l2 a partire da l1 in modo che

- l1 sia [[7,8,9],[5,5,5],[1,2,3]]
- l2 sia [[1,2,3],[9,8,7]]

```
>>> l1 = [[7,8,9],[5,5,5],[1,2,3]]
```

Esercizio

Data la lista l1 = [[7,8,9],[5,5,5],[1,2,3]], costruire e modificare l2 a partire da l1 in modo che

- l1 sia [[7,8,9],[5,5,5],[1,2,3]]
- l2 sia [[1,2,3],[9,8,7]]

```
>>> l1 = [[7,8,9],[5,5,5],[1,2,3]]
>>> l2 = [l1[0][:],l1[2][:]]
```

Esercizio

Data la lista l1 = [[7,8,9],[5,5,5],[1,2,3]], costruire e modificare l2 a partire da l1 in modo che

- l1 sia [[7,8,9],[5,5,5],[1,2,3]]
- l2 sia [[1,2,3],[9,8,7]]

```
>>> l1 = [[7,8,9],[5,5,5],[1,2,3]]
>>> l2 = [l1[0][:],l1[2][:]]
>>> l2.reverse()
```

Esercizio

Data la lista l1 = [[7,8,9],[5,5,5],[1,2,3]], costruire e modificare l2 a partire da l1 in modo che

- l1 sia [[7,8,9],[5,5,5],[1,2,3]]
- l2 sia [[1,2,3],[9,8,7]]

```
>>> l1 = [[7,8,9],[5,5,5],[1,2,3]]
>>> l2 = [l1[0][:],l1[2][:]]
>>> l2.reverse()
>>> l2[1].reverse()
```

Esercizio

Data la lista l1 = [[7,8,9],[5,5,5],[1,2,3]], costruire e modificare l2 a partire da l1 in modo che

- l1 sia [[7,8,9],[5,5,5],[1,2,3]]
- l2 sia [[1,2,3],[9,8,7]]

```
>>> l1 = [[7,8,9],[5,5,5],[1,2,3]]
>>> l2 = [l1[0][:],l1[2][:]]
>>> l2.reverse()
>>> l2[1].reverse()
>>> l1
```

Esercizio

Data la lista l1 = [[7,8,9],[5,5,5],[1,2,3]], costruire e modificare l2 a partire da l1 in modo che

- l1 sia [[7,8,9],[5,5,5],[1,2,3]]
- l2 sia [[1,2,3],[9,8,7]]

```
>>> l1 = [[7,8,9],[5,5,5],[1,2,3]]
>>> l2 = [l1[0][:],l1[2][:]]
>>> l2.reverse()
>>> l2[1].reverse()
>>> l1
[[7, 8, 9], [5, 5, 5], [1, 2, 3]]
```

Esercizio

Data la lista l1 = [[7,8,9],[5,5,5],[1,2,3]], costruire e modificare l2 a partire da l1 in modo che

- l1 sia [[7,8,9],[5,5,5],[1,2,3]]
- l2 sia [[1,2,3],[9,8,7]]

```
>>> l1 = [[7,8,9],[5,5,5],[1,2,3]]
>>> l2 = [l1[0][:],l1[2][:]]
>>> l2.reverse()
>>> l2[1].reverse()
>>> l1
[[7, 8, 9], [5, 5, 5], [1, 2, 3]]
>>> l2
```

Esercizio

Data la lista l1 = [[7,8,9],[5,5,5],[1,2,3]], costruire e modificare l2 a partire da l1 in modo che

- l1 sia [[7,8,9],[5,5,5],[1,2,3]]
- l2 sia [[1,2,3],[9,8,7]]

```
>>> l1 = [[7,8,9],[5,5,5],[1,2,3]]
>>> l2 = [l1[0][:],l1[2][:]]
>>> l2.reverse()
>>> l2[1].reverse()
>>> l1
[[7, 8, 9], [5, 5, 5], [1, 2, 3]]
>>> l2
[[1, 2, 3], [9, 8, 7]]
```

Data la matrice

$$A = \begin{bmatrix} A_{00} & A_{01} & A_{02} \\ A_{10} & A_{11} & A_{12} \\ A_{20} & A_{21} & A_{22} \end{bmatrix}$$

rappresentata come lista: $A = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]$).
Costruire B a partire da una copia profonda di A e modificare B per ottenere la trasposta di A, definita come:

$$B = \begin{bmatrix} A_{00} & A_{10} & A_{20} \\ A_{01} & A_{11} & A_{21} \\ A_{02} & A_{12} & A_{22} \end{bmatrix}$$

Data la matrice

$$A = \begin{bmatrix} A_{00} & A_{01} & A_{02} \\ A_{10} & A_{11} & A_{12} \\ A_{20} & A_{21} & A_{22} \end{bmatrix}$$

rappresentata come lista: $A = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]$).
Costruire B a partire da una copia profonda di A e modificare B per ottenere la trasposta di A, definita come:

$$B = \begin{bmatrix} A_{00} & A_{10} & A_{20} \\ A_{01} & A_{11} & A_{21} \\ A_{02} & A_{12} & A_{22} \end{bmatrix}$$

```
>>> A = [[1,2,3],[4,5,6],[7,8,9]]
```

Data la matrice

$$A = \begin{bmatrix} A_{00} & A_{01} & A_{02} \\ A_{10} & A_{11} & A_{12} \\ A_{20} & A_{21} & A_{22} \end{bmatrix}$$

rappresentata come lista: $A = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]$).
Costruire B a partire da una copia profonda di A e modificare B per ottenere la trasposta di A, definita come:

$$B = \begin{bmatrix} A_{00} & A_{10} & A_{20} \\ A_{01} & A_{11} & A_{21} \\ A_{02} & A_{12} & A_{22} \end{bmatrix}$$

```
>>> A = [[1,2,3],[4,5,6],[7,8,9]]
>>> B = [A[0][:],A[1][:],A[2][:]]
```

Data la matrice

$$A = \begin{bmatrix} A_{00} & A_{01} & A_{02} \\ A_{10} & A_{11} & A_{12} \\ A_{20} & A_{21} & A_{22} \end{bmatrix}$$

rappresentata come lista: $A = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]$).
Costruire B a partire da una copia profonda di A e modificare B per ottenere la trasposta di A, definita come:

$$B = \begin{bmatrix} A_{00} & A_{10} & A_{20} \\ A_{01} & A_{11} & A_{21} \\ A_{02} & A_{12} & A_{22} \end{bmatrix}$$

```
>>> A = [[1,2,3],[4,5,6],[7,8,9]]
>>> B = [A[0][:],A[1][:],A[2][:]]
>>> B[0][1],B[1][0]=B[1][0],B[0][1]
```

Data la matrice

$$A = \begin{bmatrix} A_{00} & A_{01} & A_{02} \\ A_{10} & A_{11} & A_{12} \\ A_{20} & A_{21} & A_{22} \end{bmatrix}$$

rappresentata come lista: $A = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]$).
Costruire B a partire da una copia profonda di A e modificare B per ottenere la trasposta di A, definita come:

$$B = \begin{bmatrix} A_{00} & A_{10} & A_{20} \\ A_{01} & A_{11} & A_{21} \\ A_{02} & A_{12} & A_{22} \end{bmatrix}$$

```
>>> A = [[1,2,3],[4,5,6],[7,8,9]]
>>> B = [A[0][:],A[1][:],A[2][:]]
>>> B[0][1],B[1][0]=B[1][0],B[0][1]
>>> B[0][2],B[2][0]=B[2][0],B[0][2]
```

Data la matrice

$$A = \begin{bmatrix} A_{00} & A_{01} & A_{02} \\ A_{10} & A_{11} & A_{12} \\ A_{20} & A_{21} & A_{22} \end{bmatrix}$$

rappresentata come lista: $A = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]$).
Costruire B a partire da una copia profonda di A e modificare B per ottenere la trasposta di A, definita come:

$$B = \begin{bmatrix} A_{00} & A_{10} & A_{20} \\ A_{01} & A_{11} & A_{21} \\ A_{02} & A_{12} & A_{22} \end{bmatrix}$$

```
>>> A = [[1,2,3],[4,5,6],[7,8,9]]
>>> B = [A[0][:],A[1][:],A[2][:]]
>>> B[0][1],B[1][0]=B[1][0],B[0][1]
>>> B[0][2],B[2][0]=B[2][0],B[0][2]
>>> B[2][1],B[1][2]=B[1][2],B[2][1]
```

Data la matrice

$$A = \begin{bmatrix} A_{00} & A_{01} & A_{02} \\ A_{10} & A_{11} & A_{12} \\ A_{20} & A_{21} & A_{22} \end{bmatrix}$$

rappresentata come lista: $A = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]$).
Costruire B a partire da una copia profonda di A e modificare B per ottenere la trasposta di A, definita come:

$$B = \begin{bmatrix} A_{00} & A_{10} & A_{20} \\ A_{01} & A_{11} & A_{21} \\ A_{02} & A_{12} & A_{22} \end{bmatrix}$$

```
>>> A = [[1,2,3],[4,5,6],[7,8,9]]
>>> B = [A[0][:],A[1][:],A[2][:]]
>>> B[0][1],B[1][0]=B[1][0],B[0][1]
>>> B[0][2],B[2][0]=B[2][0],B[0][2]
>>> B[2][1],B[1][2]=B[1][2],B[2][1]
>>> A
```

Data la matrice

$$A = \begin{bmatrix} A_{00} & A_{01} & A_{02} \\ A_{10} & A_{11} & A_{12} \\ A_{20} & A_{21} & A_{22} \end{bmatrix}$$

rappresentata come lista: $A = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]$).
Costruire B a partire da una copia profonda di A e modificare B per ottenere la trasposta di A, definita come:

$$B = \begin{bmatrix} A_{00} & A_{10} & A_{20} \\ A_{01} & A_{11} & A_{21} \\ A_{02} & A_{12} & A_{22} \end{bmatrix}$$

```
>>> A = [[1,2,3],[4,5,6],[7,8,9]]
>>> B = [A[0][:],A[1][:],A[2][:]]
>>> B[0][1],B[1][0]=B[1][0],B[0][1]
>>> B[0][2],B[2][0]=B[2][0],B[0][2]
>>> B[2][1],B[1][2]=B[1][2],B[2][1]
>>> A
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
>>> B
```

Data la matrice

$$A = \begin{bmatrix} A_{00} & A_{01} & A_{02} \\ A_{10} & A_{11} & A_{12} \\ A_{20} & A_{21} & A_{22} \end{bmatrix}$$

rappresentata come lista: $A = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]$).
Costruire B a partire da una copia profonda di A e modificare B per ottenere la trasposta di A, definita come:

$$B = \begin{bmatrix} A_{00} & A_{10} & A_{20} \\ A_{01} & A_{11} & A_{21} \\ A_{02} & A_{12} & A_{22} \end{bmatrix}$$

```
>>> A = [[1,2,3],[4,5,6],[7,8,9]]
>>> B = [A[0][:],A[1][:],A[2][:]]
>>> B[0][1],B[1][0]=B[1][0],B[0][1]
>>> B[0][2],B[2][0]=B[2][0],B[0][2]
>>> B[2][1],B[1][2]=B[1][2],B[2][1]
>>> A
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
>>> B
[[1, 4, 7], [2, 5, 8], [3, 6, 9]]
```