

# Liste in Python

---



# Tipi di Dato Composti

**Finora abbiamo discusso i tipi di dato semplici in Python**

...Ma abbiamo accennato che esistono anche tipi di dato composti

Un **tipo di dato composto** è una aggregazione di tipi di dato semplice

**In questa categoria ricadono:**

- Le **collezioni**

- ...Di cui esistono diversi tipi

- La **classi**

- ...Ovvero tipi di dato astratti, definibili dall'utente

**Discuteremo le collezioni in questa lezione, le classi tra qualche settimana**



# Liste

## Il tipo di collezione per eccellenza in Python è la lista

Proviamo a dare una definizione che non sia troppo tecnica:

Una lista è una **sequenza mutabile di oggetti**

## Si può definire la lista con la notazione

```
<lista> ::= "[" <espressioni> "]"  
<espressioni> ::= | <espressione> | <espressione> {, <espressione>}
```

- In questo caso "[" e "]" indicano le parentesi quadre (letterali)
- Tra le quadre si può inserire una sequenza di espressioni
- ...Separate da virgola
- In alternativa, si può lasciare la sequenza di espressioni vuota



# Definizione di Liste

## Vediamo qualche esempio di definizione di lista

Una lista con 3 numeri:

```
In [70]: [1, 3, 7]
```

```
Out[70]: [1, 3, 7]
```

Una lista con un solo numero:

```
In [71]: [1.3]
```

```
Out[71]: [1.3]
```

Una lista vuota:

```
In [72]: []
```

```
Out[72]: []
```



# Definizione di Liste

## Qualche altro esempio

Le espressioni possono essere eterogenee (i.e. di tipo diverso)

```
In [73]: [1, 1.7, True]
```

```
Out[73]: [1, 1.7, True]
```

È possibile assegnare una lista ad una variabile:

```
In [74]: l1 = [1, 2, 3]  
l1
```

```
Out[74]: [1, 2, 3]
```

Le espressioni possono non essere semplici:

```
In [75]: [2*3, 1+8, pow(2, 3)]
```

```
Out[75]: [6, 9, 8]
```



# Operatore di Costruzione Liste

Tecnicamente, la notazione "[...]" è un **operatore**

- Le espressioni racchiuse tra parentesi quadre sono i suoi argomenti
- ...Quindi vengono valutate **prima della costruzione** della lista

Nell'esempio appena fatto:

```
In [76]: [2*3, 1+8, pow(2, 3)]
```

```
Out[76]: [6, 9, 8]
```

- Prima vengono valutate le espressioni  $2*3$ ,  $1+8$  e  $\text{pow}(2, 3)$
- ...Quindi i valori denotati vengono usati per costruire una lista

L'operatore "[...]" **denota una lista** come risultato\_



# Operatore di Costruzione di Liste

**Un corollario importante: le espressioni in una lista possono essere liste**

Per esempio, è perfettamente valido scrivere:

```
In [77]: [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

```
Out[77]: [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

- Del resto, le espressioni una lista possono essere composte
- ...Ed il simbolo "[...]" è un operatore!

**Nell'esempio, abbiamo costruito una sorta di matrice**

- ...Ma si possono usare liste di lunghezza diversa
- ...Ed innestarle arbitrariamente

```
In [78]: [1, [2, 3], [[7, 8], 3]]
```

```
Out[78]: [1, [2, 3], [[7, 8], 3]]
```



# Accesso a Liste

Per accedere ad una lista, si usa la sintassi seguente:

```
<lista>["<espressione>"]
```

Dove <espressione> specifica l'indice di interesse all'interno della sequenza

## Vediamo qualche esempio

- Il primo elemento ha indice 0 e gli altri a seguire

```
In [79]: a = [7, 2, 9]
          print(a[0])
          print(a[2])
```

7

9





# Accesso a Liste

Se accediamo oltre l'indice massimo (lunghezza - 1), otteniamo un errore

```
In [80]: a = [7, 8, 9]
         print(a[3])
```

```
-----
IndexError                                Traceback (most recent call last)
Cell In[80], line 2
      1 a = [7, 8, 9]
----> 2 print(a[3])

IndexError: list index out of range
```

Per conoscere la lunghezza di una lista, si usa la funzione `len(<lista>)`

```
In [81]: a = [7, 8, 9]
         print(len(a))
```

3

 `len` funziona con ogni tipo di collezione

# Accesso a Liste

In Python, si possono **usare come indici numeri negativi**

In questo caso, -1 è l'indice dell'ultimo elemento e così via

```
In [82]: a = [7, 8, 9]
         print(a[-1])
         print(a[-2])
```

9

8

## Gli indici devono però essere numeri interi (o "slices")

```
In [83]: a = [7, 8, 9]
         a[1.3]
```

-----  
TypeError

Traceback (most recent call last)

Cell In[83], line 2

1 a = [7, 8, 9]

----> 2 a[1.3]



TypeError: list indices must be integers or slices, not float

# Operatore di Indicizzazione

**Il simbolo "[...]", quando usato per accedere ad una lista...**

...È considerato come un operatore e si chiama **operatore di indicizzazione**

- Quindi la notazione `<lista>[<indice>]` è una espressione composta
- I cui argomenti sono **la lista** e **l'indice**
- ...E che denota il valore indicizzato

```
In [84]: a = [7, 8, 9]
print(a[1]) # lista ed indice come espressioni semplici
print(a[1+1]) # lista semplice, indice come espressione composta
print([1, 4, 5][0]) # lista come espressione composta!
```

8

9

1

- Nell'ultimo esempio, `[1, 4, 5]` denota una lista

- ...Cui viene applicato l'operatore di indicizzazione con argomento 0



# Assegnamento di Elementi

**È possibile accedere ad un elemento in scrittura**

- In pratica, se l'operatore di indicizzazione è usato a sx del segno "="
- ...La posizione corrispondente della lista è trattata **come una variabile**

```
In [85]: a = [7, 8, 9]
print(f'Prima: {a}')
a[0] = 1
print(f'Dopo: {a}')
```

```
Prima: [7, 8, 9]
Dopo: [1, 8, 9]
```

In questo modo si può modificare un elemento della lista

**Intuitivamente:**

- Potete trattare una lista
- ...Come una **sequenza di variabili**



# Accesso Mediante "Slice"

**Si può selezionare una sottosequenza da una lista (i.e. una sotto-lista)**

...Usando un costrutto chiamato **slice**

- In particolare, uno slice specifica una sequenza di indici
- Si usa la sintassi seguente:

```
<primo indice incluso>:<primo indice escluso>
```

**Vediamo subito un esempio:**

```
In [86]: l = [2, 4, 6, 8, 10]
         l[0:3]
```

```
Out[86]: [2, 4, 6]
```

- Iniziamo a prendere elementi dall'indice 0

- ...E ci fermiamo all'indice 3 (escluso)



# Accesso Mediante "Slice"

## È possibile "saltare" indici

...Usando la notazione seguente:

```
<primo indice incluso>:<primo indice escluso>:<passo>
```

- Il primo indice incluso <primo indice incluso>
- Il secondo è <primo indice incluso> + <passo>
- Il terzo è <primo indice incluso> + <passo> + <passo>
- ...E così via finché non viene raggiunto o superato <primo indice escluso>

## Vediamo un esempio:

```
In [87]: l = [2, 4, 6, 8, 10, 11, 8, 3]
         l[0:5:2] # seleziona gli indici 0, 2 e 4
```

```
Out[87]: [2, 6, 10]
```



# Accesso Mediante "Slice"

**È possibile omettere alcuni degli argomenti dello slice**

- Se si omette il primo indice da includere, si assume sia 0:

```
In [88]: l = [2, 4, 6, 8, 10, 11, 8, 3]
         l[:5]
```

```
Out[88]: [2, 4, 6, 8, 10]
```

- Se si omette il primo indice da escludere
- ...Si assume sia la lunghezza della lista (i.e. si arriva alla fine)

```
In [89]: l = [2, 4, 6, 8, 10, 11, 8, 3]
         l[3:]
```

```
Out[89]: [8, 10, 11, 8, 3]
```



# Operatore di Indicizzazione

## L'operatore di indicizzazione

- Funziona per molti tipi di collezione (come vedremo)
- ...Ed anche per le stringhe!

## In questo caso denota una stringa

...Costituita dal carattere indicizzato

```
In [90]: s = 'Ciao, mondo!'
         s[0]
```

```
Out[90]: 'C'
```

Si possono usare anche gli slice:

```
In [91]: s = 'Ciao, mondo!'
         s[0:4]
```

```
Out[91]: 'Ciao'
```





# Rimozione ed Aggiunta di Valori

## È possibile eliminare elementi in una lista

Si può usare l'istruzione `del`

```
In [92]: l = [1, 2, 3]
         print(l)
         del l[0]
         print(l)
```

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

- `del` è in grado di eliminare anche le normali variabili
- ...Ma ha diverse controindicazioni

In questo corso, eviteremo di utilizzarlo

**Infine, si possono aggiungere elementi ad una lista**

...Ma vedremo come farlo tra qualche settimana



# Operatori per Liste

## Alcuni operatori hanno un significato particolare per le liste

L'operatore `+`, se applicato a due liste, le **concatena**

```
In [93]: l1 = [1, 2]
         l2 = [3, 4]
         l1 + l2
```

```
Out[93]: [1, 2, 3, 4]
```

- L'operatore `*`, se applicato ad una lista `l` e ad un numero intero `n`
- ...Ripete la lista `l` per `n` volte

```
In [94]: l1 * 3
```

```
Out[94]: [1, 2, 1, 2, 1, 2]
```



# Operatore `in`

## Si può verificare se una lista contenga un elemento

...Mediante l'operatore `in`, che ha la sintassi:

```
<espressione> in <espressione lista>
```

- Dove `<espressione>` indica l'elemento da cercare
- ...Ed `<espressione lista>` dove cercarlo

Restituisce `True` se l'elemento viene trovato

## Vediamo un paio di esempi:

```
In [95]: l = [2, 4, 6, 8]
print(8 in l)
print(5 in l)
```

```
True
False
```

