Tipi di dato complessi

Liste

Le **liste** sono usate per immagazzinare in una sola variabile più dati.

Caratteristiche principali:

- **Ordinate**: gli elementi hanno un ordine preciso basato sull'indice, che parte da 0. Questo ordine non cambia a meno che non venga modificato con metodi specifici.
- **Modificabili**: può aggiungere, cambiare o rimuovere elementi dopo aver creato una lista.
- **Permettono duplicati**: le liste possono contenere più elementi con lo stesso valore.
- **Tipi di dato**: possono contenere qualunque tipo di dato, anche mischiati!

```
lista = ["mela", "banana", "ciliegia", "kiwi", "melone"]
l = [] # Lista vuota
print(lista)
```

```
['mela', 'banana', 'ciliegia', 'kiwi', 'melone']
```

Perché può tornarci molto utile?

Esercizio 1

Detector di parole bannabili.

Supponiamo di avere una chat pubblica in un gioco online. Ogni commento racconta, sessista, volgare o scurrile deve essere bannato. Supponiamo che il gestore della chat voglia bannare ogni commento che possiede le parole:

- Uccidere
- Morte
- Sesso
- Schifo
- Stupido
- Nero
- Porco

Implementare un rilevatore di queste parole e bannare immediatamente l'utente che le utilizza, altrimenti stampare il messaggio.

Implementazione senza liste

```
# Definisco le parole bannabili

p1 = "uccidere"
p2 = "morte"
p3 = "sesso"
p4 = "schifo"
p5 = "stupido"
p6 = "nero"
p7 = "porco"

# Prendo in input una parola

messaggio = input("Scrivi il messaggioNa")

# Controllo se è bannabile e restituisco il risultato

if p1 == messaggio.lower() or p2 == messaggio.lower() or p3 == messaggio.lower() or \
    p4 == messaggio.lower() or p5 == messaggio.lower() or p6 == messaggio.lower() or \
    p7 == messaggio.lower():
    print("Sei bannato!")
else:
    print(messaggio)
```

```
Scrivi il messaggio
Cattivo
Cattivo
```

Notare se volessimo aggiungere una nuova parola da bannare, bisognerebbe aggiungere una nuova variabile in alto e un nuovo controllo nella condizione dell'if.

Implementazione con le liste

```
# Riprendiamo l'esercizio di prima ma usiamo le liste

bannabili = ["uccidere", "morte", "sesso", "schifo", "stupido", "nero", "porco"]

messaggio = input("Scrivi messaggioNa")

# Controllo se è bannabile e restituisco il risultato

if messaggio.lower() in bannabili:
    print("Sei bannato!")
else:
    print(messaggio)
```

```
Scrivi messaggio
Nero
Sei bannato!
```

Metodi

Accesso con indici

Per **accedere** alle liste si usano gli **indici**.

```
lista[1]
```

```
'banana'
```

```
lista[4]
```

```
'mela'
```

```
# Cambio il frutto con indice 1
lista[1] = "pera"
print(lista)
```

```
['mela', 'pera', 'ciliegia', 'kiwi', 'melone']
```

```
l1 = [8, 18, 28, 38, 48, 58]
l1[-1]
```

```
58
```

```
l1[2:4]
```

```
[28, 38]
```

```
l1[2:]
```

```
[28, 38, 48, 58]
```

Stringhe

NOTARE: anche le stringhe (ti tipi stringi) sono delle liste, quindi gli indici e tutti questi metodi di accesso valgono!

```
frase = "Che bella questa corsa, mi sto divertendo un sacco!"
frase[24:]
```

```
'mi sto divertendo un sacco!'
```

```
frase[:22]
```

```
'Che bella questa corsa'
```

```
frase[-1]
```

```
'!'
```

- stringa[start:stop:step]
- start -> Punto di partenza (vuoto significa "inizia dall'inizio").
- stop -> Punto di fine (vuoto significa "arriva fino alla fine").
- step -> Indica di quanti passi muoversi (se è -1, significa "vai all'indietro").

```
nome = "gianluca"
print(nome)
print(nome[::-1])

nome = "anna"
print(nome)
print(nome[::-1])
```

```
gianluca
aculnaig
anna
anna
```

Aggiungere elementi

Per **aggiungere** degli elementi nella lista si possono utilizzare dei metodi!

Metodo	Descrizione
append()	Aggiunge un elemento alla fine della lista.
insert()	Inserisce un elemento in una posizione specificata della lista.
extend()	Estende la lista aggiungendo gli elementi di un'altra iterabile (lista, tupla, ecc.).

```
thislist = ["mela", "banana", "ciliegia"]
thislist.append("arancia")
print(thislist)
```

```
['mela', 'banana', 'cherry', 'orange']
```

```
thislist = ["mela", "banana", "ciliegia"]
thislist.insert(1, "arancia")
print(thislist)
```

```
['apple', 'orange', 'banana', 'cherry']
```

```
thislist = ["mela", "banana", "ciliegia"]
tropical = ["ananas", "mango", "papaya"]
thislist.extend(tropical)
print(thislist)
```

```
['apple', 'banana', 'cherry', 'mango', 'pineapple', 'papaya']
```

Rimuovere elementi

Per **rimuovere** degli elementi nella lista si possono utilizzare dei metodi

Metodo	Descrizione
remove()	Rimuove l'elemento specifico, se ce ne sono di più rimuove solo il primo
pop()	Rimuove l'elemento all'indice specificato, se non si specifica l'elemento rimuove l'ultimo
clear()	Rimuove tutti gli elementi, la lista diventa vuota

```
thislist = ["mela", "banana", "ciliegia", "banana", "kiwi"]
thislist.remove("banana")
print(thislist)
```

```
['apple', 'cherry', 'banana', 'kiwi']
```

```
thislist = ["mela", "banana", "ciliegia"]
thislist.pop(1)
print(thislist)
```

```
['apple', 'cherry']
```

```
thislist.clear()
print(thislist)
```

```
[]
```

Miscellanea

Metodo	Descrizione
count()	Conta le occorrenze di uno specifico elemento
split()	**ATTENZIONE** si applica a una stringa, ritorna una lista con gli elementi creati dividendo la stringa tramite il separatore passato come parametro.
join()	**ATTENZIONE** si applica a una stringa, ritorna una stringa unendo tutti gli elementi della lista presa come parametro utilizzando la stringa come separatore

Funzione len(): non è un metodo ma è una funzione (che abbiamo già visto per le stringhe), permette di stampare il numero di elementi nella lista, come nelle stringhe restituisce il numero di caratteri da cui è composta.

```
# count()
thislist = ["mela", "banana", "ciliegia", "banana", "kiwi"]
thislist.count("banana")
```

```
2
```

```
# split()
stringa = "Ciao come stai"
parole = stringa.split(" ") # Divido per spazi
print(parole) # Lista di tre elementi
```

```
['Ciao', 'come', 'stai']
```

```
# join()
nuova_stringa = "_".join(parole)
print(nuova_stringa)
```

```
Ciao_come_stai
```

```
len(thislist)
```

```
5
```

```
stringa = "Ciao a tutti come stata?"
len(stringa)
```

```
24
```

Tuple

La tuple, come le liste, sono utilizzate per immagazzinare in una sola variabile più dati, ma hanno delle caratteristiche diverse.

- **Ordinate**: significa che gli elementi hanno un ordine definito, e tale ordine non cambierà.
- **Immutabili**: è che significa che non possono modificare, aggiungere o rimuovere elementi dopo che una tupla è stata creata.
- **Consentono duplicati**: poiché le tuple sono indicizzate, possono contenere elementi con lo stesso valore.
- **Tipi di dato**: possono contenere qualunque tipo di dato, anche mischiati!

In breve sono delle strutture identiche alle liste ma i valori immagazzinati non possono essere cambiati!

```
mytuple = ("mela", "banana", "ciliegia", "arancia", "kiwi", "melone", "mango")
print(mytuple)
```

```
('apple', 'banana', 'cherry', 'orange', 'kiwi', 'melon', 'mango')
```

Tuple con un solo elemento:

```
singleTuple = ("mela") # ATTENZIONE! non è una tupla
print(type(singleTuple))
```

```
<class 'str'>
```

```
singleTuple = ("mela",)
print(type(singleTuple))
```

```
<class 'tuple'>
```

Metodi

Accesso con indici

Per accedere agli elementi bisogna usare, come nelle liste, gli indici.

```
print(mytuple[1])
```

```
banana
```

Modifica

Come abbiamo detto, le tuple sono tipi di dato complesso *immutabile* però, si può barare!

Basta convertire una tuple in lista, cambiarne gli elementi e farla tornare una tuple. (Fa schifo)

```
x = ("mela", "banana", "kiwi")
print(x)
y = list(x)
y[1] = "arancia"
x = tuple(y)

print(x)
```

```
('mela', 'banana', 'kiwi')
('mela', 'arancia', 'kiwi')
```

Miscellanea

Dato che le operazioni che si possono fare con una tupla sono poche, i metodi sono molto pochi (Metodo | Descrizione | --- | --- | count() | Conta le occorrenze di uno specifico elemento)

Funzione len(): non è un metodo ma è una funzione (che abbiamo già visto per le string e per le list); permette di stampare il numero di elementi nella tupla.

```
mytuple = ("mela", "banana", "banana", "kiwi", "banana")
mytuple.count("banana")
```

```
3
```

```
len(mytuple)
```

```
5
```

Sets

Set in italiano si traduce con **insieme**. Anch'essi sono utilizzati per immagazzinare più valori in una singola variabile.

- Non ordinati: gli elementi in un set non hanno un ordine preciso e non possono essere usati con indici.
- Parzialmente immutabili: non puoi modificare i singoli elementi di un set, ma puoi aggiungere o rimuovere elementi.
- Niente duplicati: Un set non può contenere due elementi con lo stesso valore.
- Tipi di dato: possono immagazzinare diversi tipi di dato mischiati MA...

```
thisset = {"mela", "banana", "ciliegia", "mela"}
print(thisset)

{'ciliegia', 'mela', 'banana'}
```

```
# True e 1 sono considerati lo stesso valore
thisset = {"mela", "banana", "mango", 1, True, 1, 2}
print(thisset)

# False e 0 sono considerati lo stesso valore
thisset = {"mela", "banana", "mango", False, 0, True, 2}
print(thisset)
```

```
{1, 2, 'banana', 'mango', 'mela'}
{False, True, 2, 'banana', 'mango', 'mela'}
```

Metodi

Accesso (senza indici)

Come abbiamo detto non è possibile accedere ai singoli elementi tramite indice. L'unico modo è tramite dei cicli, loops, che vedremo più avanti.

Aggiungere elementi

Si possono inserire nuovi elementi attraverso dei metodi.

Metodo	Descrizione
add()	Inserisce un elemento all'interno dell'insieme (set)
update()	Aggiunge tutti gli elementi di una qualsiasi struttura dati all'interno dell'insieme (set)

```
thisset = {"mela", "banana", "ciliegia"}
print(thisset)
thisset.add("arancia")
print(thisset)

{'ciliegia', 'mela', 'banana'}
{'ciliegia', 'arancia', 'mela', 'banana'}
```

```
thisset = {"mela", "banana", "ciliegia"}
print(thisset)
tropical = {"ananas", "mango", "papaya"}
thisset.update(tropical)
print(thisset)
numeri = {9, 1, 2, 3, 4, 5}
thisset.update(numeri)
print(thisset)

{'ciliegia', 'mela', 'banana'}
{'ciliegia', 'banana', 'papaya', 'ananas', 'mela', 'mango'}
{9, 1, 2, 3, 4, 5, 'banana', 'mango', 'ciliegia', 'papaya', 'ananas', 'mela'}
```

Rimuovere elementi

Anche per la rimozione esistono dei metodi.

Metodo	Descrizione
remove()	Rimuove l'elemento passato come parametro, se non esiste all'interno dell'insieme *produce un errore*
discard()	Rimuove l'elemento passato come parametro, se non esiste all'interno dell'insieme **NON** *produce un errore*
pop()	Rimuove un elemento **random** (a caso) dall'insieme
clear()	Rimuove tutti gli elementi nell'insieme

```
thisset = {"mela", "banana", "ciliegia", "arancia", "kiwi"}
thisset.remove("banana")
print(thisset)

# Controllare errore
# thisset.remove("pera")

{'ciliegia', 'kiwi', 'mela', 'arancia'}
```

```
thisset.discard("kiwi")
print(thisset)

# Non restituisce un errore
thisset.discard("pera")
print(thisset)

{'ciliegia', 'mela', 'arancia'}
```

```
thisset = {"mango", "pera", "mela", "banana", "ciliegia"}
thisset.pop()
print(thisset)
print(thisset)

{'ciliegia', 'banana', 'mango', 'mela', 'pera'}
```

```
thisset.clear()
print(thisset)

set()
```

Miscellanea

Metodo	Descrizione
difference()	Restituisce un set con gli elementi presenti solo nel primo set (differenza).
intersection()	Restituisce un set con gli elementi comuni a entrambi i set.
subset()	Restituisce `True` se i due set non hanno elementi in comune.
union()	Restituisce un set con tutti gli elementi presenti in entrambi i set (unione).

Funzione len(): non è un metodo ma è una funzione), permette di stampare il numero di elementi nell'insieme.

```
# Inizializziamo due set
set1 = {1, 2, 3, 4, 5}
set2 = {3, 4, 5, 6}
set3 = {5, 6, 8, 9}

# difference()
print(set1.difference(set2))

# intersection()
print("Intersezione 1 e 2 -> {set1.intersection(set2)}")
# subset()
print(f"1 e 2 sono disgiunti -> {set1.isdisjoint(set2)}")

# intersection()
print("Intersezione 1 e 3 -> {set1.intersection(set3)}")
# subset()
print(f"1 e 3 sono disgiunti -> {set1.isdisjoint(set3)}")

# union()
print(set1.union(set2))
```

```
{1, 2}
Intersezione 1 e 2 -> {3, 4}
1 e 2 sono disgiunti -> False
Intersezione 1 e 3 -> set()
1 e 3 sono disgiunti -> True
{1, 2, 3, 4, 5, 6}
```

```
len(set1)
```

```
5
```