UNIVERSITÀ di VERONA

# Linguaggio di Programmazione Python

05/04/2022

Giulio Mazzi

giulio.mazzi@univr.it

# Liste – breve recap

Le liste sono **sequenze di valori**. Possono contenere elementi eterogenei (e.g., possono combinare stringhe e interi)

```
terna = [23, 72, 8]
primari = ['giallo', 'magenta', 'ciano']
voti = [27, 30, 24, 'idoneo']
```

Le liste sono **mutabili.**

```
l = [1, 2, 3]
h = l
print(l, h)

l = l + [4]
print(l, h)
```

```
[1, 2, 3] [1, 2, 3]
[1, 2, 3, 4] [1, 2, 3]
```

# Operazioni su Liste (e tipi sequenziali)

| Operation | Result |
|---|---|
| `x in s` | `True` if an item of *s* is equal to *x*, else `False` |
| `x not in s` | `False` if an item of *s* is equal to *x*, else `True` |
| `s + t` | the concatenation of *s* and *t* |
| `s * n` or `n * s` | equivalent to adding *s* to itself *n* times |
| `s[i]` | *i*th item of *s*, origin 0 |
| `s[i:j]` | slice of *s* from *i* to *j* |
| `s[i:j:k]` | slice of *s* from *i* to *j* with step *k* |
| `len(s)` | length of *s* |
| `min(s)` | smallest item of *s* |
| `max(s)` | largest item of *s* |
| `s.index(x[, i[, j]])` | index of the first occurrence of *x* in *s* (at or after index *i* and before index *j*) |
| `s.count(x)` | total number of occurrences of *x* in *s* |

# Slice

Abbiamo visto che si può accedere a un elemento della lista usando le parentesi quadre.

Esistono Però alcune utili varianti:

- Usando indici negativi, conto a partire dal fondo. L'ultimo elemento è -1, il penultimo –2...
- Possiamo ritagliare una fetta (*slice*) di lista usando il formalismo `l[start:end]`. Se ometto uno dei due valori, prendo l'inizio/la fine.
- Possiamo prendere solo alcuni elementi specificando il passo. Per esempio `l[1:6:2]` prende gli elementi 1, 3 e 5 (parto da uno, aggiungo il passo di 2, mi fermo quando arrivo a una posizione ≥6)

```
>>> l = [1, 2, 3, 4, 5, 6]
>>> l[0], l[1]
(1, 2)
>>> l[-1], l[-2]
(6, 5)
>>> l[1:3]
[2, 3]
>>> l[2:]
[3, 4, 5, 6]
>>> l[:4]
[1, 2, 3, 4]
>>> l[:-2]
[1, 2, 3, 4]
>>> l[1:6:2]
[2, 4, 6]
```

# Slice - assegnamento

Se assegno un valore a una sottolista, modifico la lista originale.

```
>>> l = [0, 1, 2, 3, 4]
>>> l[3:5] = [0, 0, 0, 0, 0]
>>> print(l)
[0, 1, 2, 0, 0, 0, 0, 0]
>>>
```

Se assegno una sottolista a un altro elemento, creo una copia. Posso copiare un intera lista usando l[:] (oppure l.copy())

```
>>> l = [1, 2, 3]
>>> a = l
>>> print(a is l)
True
>>> b = l[:]
>>> print(b is l)
False
>>> c = l.copy()
>>> print(c is l)
False
```

# In, index, count

- Il metodo **`in`** (risp. **`not in`**) ritorna True se un elemento è presente (risp. non presente) in una lista

- Il metodo `l.index(val)` mi ritorna l'indice della prima occorrenza di val dentro la lista l. Ritorna un errore se l'elemento non è presente.

- Il metodo `l.count(val)` conta il numero di occorrenze di val nella lista l

```
>>> l = [1, 2, 3]
>>> 2 in l
True
>>> 4 not in l
True
>>> l.index(2)
1
>>> l.index(4)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: 4 is not in list
>>> l.count(2)
1
>>> l.count(4)
0
```

# Modificare una lista

- Abbiamo visto che con **`append`** possiamo aggiungere un elemento lista, modificandola direttamente (no copie)
- Possiamo usare `l.clear()` per rimuovere tutti gli elementi
- Con `l.insert(val, pos)` possiamo inserire il valore `val` in posizione `pos`
- Con `l.delete(pos)` possiamo rimuovere l'elemento in posizione `pos`
- Con `l.sort()` possiamo ordinare gli elementi in una lista (la lista non deve essere eterogenea!)

Potete trovare una descrizione completa nella documentazione:

https://docs.python.org/3/library/stdtypes.html#sequence-types-list-tuple-range

# List comprehension

Python ci offre un modo compatto per inizializzare I valori di una lista. Possiamo includere in una sola riga un intero ciclo che inizializza ogni elemento della lista. Per esempio

```
>>> l = [ x**2 for x in range(1,6) ]
>>> print(l)
[1, 4, 9, 16, 25]
```

Inizializza ogni elemento della lista al quadrato di un numero da uno a cinque.

Possiamo combinarlo a un if inline per filtrare gli elementi

```
>>> l2 = [ x for x in range(20) if x % 4 == 0 ]
>>> print(l2)
[0, 4, 8, 12, 16]
```

# List comprehension - dettagli

In generale, la sintassi è:

```
newlist = [expression for item in iterable if condition == True]
```

Se uso una lista come iterable, questa non viene modificata, il risultato è una nuova lista.

Ogni elemento generato da expression è unico. Posso usare questo metodo per inizializzare delle liste di liste senza problemi

```
>>> l2 = [[] for _ in range(10)]
>>> print(l2)
[[], [], [], [], [], [], [], [], [], []]
>>> l2[0].append(1)
>>> print(l2)
[[1], [], [], [], [], [], [], [], [], []]
```