Le liste

Una lista in Python è un oggetto con metodi propri che, come una stringa, contiene una sequenza di elementi ordinati in base a un indice che inizia col valore 0.

A differenza di una stringa:

- gli elementi di una lista possono essere dati qualsiasi, anche di tipo diverso tra loro (in una stringa abbiamo solo una sequenza di singoli caratteri)
- a differenza di una stringa, la lista può essere modificata.

La sintassi per dichiarare una lista è:

```
nomeLista=[elemento1, elemento2, elemento3 ...]
```

Poiché una lista è una struttura dati dinamica, ossia si può modificare, spesso è utile creare una lista vuota con la dichiarazione

```
listavuota = []
```

e aggiornarla in seguito.

Esempio (1)

```
lista = ['Luca', 5.5, 'Anna', 7]   #lista che contiene due stringhe, un dato di tipo int (7) e un dato di tipo float (5.5)

for elemento in lista:
     print(elemento, type(elemento))   #stampa i singoli dati e il tipo di dato
```

Data una lista, per selezionare l'elemento della lista nella posizione 2 (la terza) si usa `nomeLista[2]`.

Per assegnare un dato alla posizione 2 della lista: `nomeLista[2]=dato`

Ricordare che la lista si può modificare: assegnando dei nuovi dati, i vecchi dati memorizzati vengono persi.

Si può creare una lista a partire da una stringa con il metodo di stringa **`split()`**:

```
stringavoti = 'Luca 5.5 Anna 7'
listavoti = stringavoti.split()
```

Con la funzione **`len()`**, applicata prima alla stringa e poi alla lista, si può verificare che la stringa è formata da 15 elementi, mentre la lista è formata da 4 elementi.

Con l'**operatore di ripetizione *** si crea una lista contenente una serie di dati identici:

```
lista1 = ['nero']*5
```

crea la lista

```
['nero', 'nero', 'nero', 'nero', 'nero']
```

Con la funzione **`list()`** si crea una lista di caratteri a partire da una stringa:

```
lista2 = list('Pacinotti')
```

crea la lista

```
['P', 'a', 'c', 'i', 'n', 'o', 't', 't', 'i']
```

Con la funzione **`list()`** unita alla funzione **`range()`** si creano valori legati da una relazione matematica.

Ricordando che la funzione range ha come argomenti:

- il valore iniziale da cui parte il conteggio;

- il valore finale prima del quale ci si ferma;

- il passo, ossia l'incremento a ogni iterazione;

con

```
lista3 = list(range(0, 5, 1))
```

si crea una lista contenente i numeri da 0 a 4.

Si possono lasciare sottintesi il valore iniziale e il passo:

```
lista4 = list(range(5))
```

crea una lista uguale a `lista3`.

Con:

```
lista5 = list(range(5, 51, 5)
```

si crea una lista contenente la tabellina del 5.

Si può creare una lista unendo due liste tra loro con l'**operatore +:**

```
lista6 = [1, 2, 3, 4]
lista7 = [5, 6, 7, 8, 9]
lista8 = lista6 + lista7
```

La lista 8 sarà `[1, 2, 3, 4, 5, 6, 7, 8, 9]`.

L'ultima opzione è raramente necessaria perché in Python le liste sono dinamiche, posssono cioè variare di dimensione durante le elaborazioni.

Sono definite le funzioni di massimo **`max()`**, minimo **`min()`**,somma dei dati **`sum()`**:

`min()` per liste di soli numeri e per liste di sole stringhe(nel secondo caso, fornisce il primo elemento in ordine alfabetico);

`max()` per liste di soli numeri e per liste di sole stringhe (nel secondo caso, fornisce l'ultimo elemento in ordine alfabetico);

`sum()` solamente per liste di numeri.

Liste annidate e matrici

Una lista annidata è una lista contenuta come elemento in un'altra lista.

Ad esempio, la lista

```
listaA = [10, 20, 30, [40, 50]]
```

contiene come lista annidata la lista `[40, 50]` nella posizione 3.

Il dato `40` è l'elemento 0 della lista annidata, quindi il suo indice è `[3][0]`.

Nella tabella sono riportati i dati della `listaA` e gli indici corrispondenti:

Indice	[0]	[1]	[2]	[3]	
				[0]	[1]
Dato	10	20	30	40	50

Come altro esempio:

```
listaB = [11, 12, 13, [14, [15, [16, 17]]]
```

La `listaB` contiene come lista annidata nella posizione 3 la lista `[14, [15, [16, 17]]]`,
che a sua volta contiene nella sua posizione 1 la lista `[15, [16, 17]]]`,
che a sua volta contiene nella posizione 1 la lista `[16, 17]`.

L'indice del dato `15` è `[3][1][0]`.

L'indice del dato `16` è `[3][1][1][0]`.

Indice	[0]	[1]	[2]	[3]			
				[0]	[1]		
					[0]	[1]	
						[0]	[1]
Dato	11	12	13	14	15	16	17

Una matrice è una lista che contiene solamente altre liste, e queste hanno lo stesso numero di elementi.

È utile lavorare con matrici quando si ha a che fare con serie ripetute di dati, simili a quelle delle tabelle di un database. Le matrici consentono di memorizzare ed elaborare più facilmente i dati.

Esempio di matrice:

```
studenti = [['Marco', 17, 'IIIR'], ['Matteo', 17, 'IIIR'], ['Luca', 18, 'IIIS']]
```

	Dato[0]	Dato[1]	Dato[2]
studenti[0]	'Marco'	17	'IIIR'
studenti[1]	'Matteo'	17	'IIIR'
studenti[2]	'Luca'	18	'IIIS'

Si possono individuare i nomi degli studenti in una lista annidata usando un ciclo `while`, selezionando i valori della colonna 0 della matrice. Il primo indice si riferisce alle liste annidate: si fa variare, tenendo fisso il secondo.

```
indice=0
while indice<len(studenti):
      print(studenti[indice] [0], end=' ' )
      indice+=1
```

(Usiamo il comando `end=' '` per stampare più dati su una stessa riga.)

Metodi dell'oggetto lista, da utilizzare con la sintassi `oggetto.metodo(argomenti)`:

`append(`*`valore`*`)` inserisce il valore tra parentesi alla fine della lista;

`insert(`*`indice, valore`*`)` inserisce il valore nella posizione indicata;

`pop(`*`indice`*`)` elimina il valore nella posizione indicata;

`remove(`*`valore`*`)` elimina il valore considerato;

`clear()` elimina tutti i valori dalla lista.

Tuple

Una tupla ha caratteristiche simili a quelle di una lista: è un oggetto con metodi propri che contiene una serie di dati ordinati in base a un indice.

Una tupla non può essere modificata.

- L' elaborazione di una tupla è più veloce rispetto a quella di una lista.

- I dati memorizzati nella tupla sono al sicuro: questo è importante nella programmazione web-oriented.

Mentre per dichiarare una lista si usano le parentesi quadre [], per definire una tupla usano le parentesi tonde (), che possono anche essere omesse. La tupla contenente gli elementi 'Mario', 18, 'tre' si può dichiarare quindi in due modi :

```
tupla1 = 'Mario', 18, 'tre'    oppure    tupla1 = ('Mario', 18, 'tre')
```

Per dichiarare una tupla composta da un solo elemento, si deve inserire una virgola:

```
tupla2 = 'Mario',      oppure          tupla2 = ('mario',)
```

Con la funzione **`tuple()`** si possono convertire in tuple altri oggetti come liste o stringhe:

```
tupla3 = tuple('abc')
```

a partire da una stringa, e

```
tupla3 = tuple(['a', 'b', 'c'])
```

a partire da una lista, sono equivalenti a dichiarare

```
tupla3 = ('a', 'b', 'c').
```

Con le tuple si possono cercare elementi con il metodo **`index()`** e cercare ricorrenze con il metodo **`count()`**, si possono operare slicing e individuare valori minimi e massimi con **`min()`** e **`max()`.**

Non si possono invece usare i metodi dinamici visti per le liste come `append(), pop(), sort(), reverse()` che modificherebbero la tupla.

Una funzione importante è **`zip()`**, che accetta come argomenti degli oggetti come stringhe o liste e restituisce i valori corrispondenti sotto forma di tuple.

Ad esempio:

```
nome = ['Matteo', 'Marco', 'Luca', 'Giovanni']

cognome = ['Rossi', 'Bianchi', 'Neri', 'Verdi']

tuplacerniera=zip(nome,cognome)

for elemento in tuplacerniera:
      print(elemento)
```

L’esempio precedente stampa le tuple `('Matteo', 'Rossi'), ('Marco', 'Bianchi'), ('Luca', 'Neri'), ('Giovanni', 'Rossi').`

Se una delle liste avesse un numero inferiore di elementi rispetto all’altra? Verrebbe stampato un numero di tuple pari alla lunghezza della llista più corta.

Si possono incernierare anche più di due liste:

```
nome = ['Matteo', 'Marco', 'Luca', 'Giovanni']
cognome = ['Rossi', 'Bianchi', 'Neri', 'Verdi']
età = [16, 17, 18, 19]
tuplacerniera2 = zip(nome,cognome, età)
for elemento in tuplacerniera2:
     print(elemento)
```

I dizionari

In Python un dizionario:

- è un oggetto con metodi propri, che può essere modificato;

- contiene un insieme di dati indicizzati ma non ordinati.

A differenza delle strutture di dati viste sino a ora, gli elementi di un dizionario non hanno un indice numerico progressivo e automatico.

Nei dizionari gli indici, detti chiavi, sono scelti dal programmatore e possono essere stringhe alfanumeriche.

La sintassi per dichiarare un dizionario è:

```
nomeDizionario={
                chiaveA:valoreA,
                chiaveB:valoreB,
                ...,
                }
```

Vediamo un esempio di dizionario con quattro elementi:

```
dizionario={
             'nome':'Matteo',
             'cognome':'Rossi',
             'voto':4.5,
             4:'fine',
            }
```

Le chiavi del dizionario sono tre stringhe (`'nome'`, `'cognome'`, `'voto'`) e un numero `(4)`.

Per stampare l'intero dizionario: `print(dizionario)`

Per stampare un singolo valore: indicare tra parentesi quadre la relativa chiave.

Oltre alla dichiarazione con parentesi graffe, si può usare la funzione **`dict()`**, con sintassi:

```
nomeDizionario = dict(chiaveA='valoreA', chiaveB = 'valoreB', ...)
```

In questo caso la chiave NON VA RACCHIUSA TRA APICI, neppure quando è una stringa, e il valore è assegnato con l'operatore = anziché con i due punti.

Per visualizzare separatamente chiavi e valori di un dizionario: ricorrere a cicli iterativi oppure ai metodi `values()` e `keys()` dell'oggetto dizionario.

Esempio

```
dizionario={'nome':'Matteo', 'cognome':'Rossi', voto:'3.5',}


print('stampa delle chiavi:', end=' ')
for x in dizionario:
     print(x, end=' ')
print('\nstampa dei valori:\, end='')
for x in dizionario:
     print(dizionario[x], end='')

print('\ncon il metodo keys():', dizionario.keys())

print('con il metodo values():', dizionario.values())
```

Nei cicli for la variabile su cui si itera assume automaticamente il valore di tutte le chiavi del dizionario.

Modificare i contenuti di un dizionario

```
dizionario={'nome':'Matteo', 'cognome':'Rossi', 'voto':3.5,}
valori=dizionario.values()
chiavi=dizionario.keys()
print('Prima:\n', valori)
print('', chiavi)

dizionario['nome']='Paolo'
dizionario['anni']=18
print('Dopo:\n', valori)
print(chiavi)
```

`len()` restituisce il numero delle chiavi

`pop()` elimina la chiave passata come argomento insieme al valore corrispondente

`items()`: restituisce un oggetto vista contenente le coppie chiave-valore sotto forma di tuple.

Esempio

Creiamo un programma che identifichi quante lettere appaiono in un testo e quante volte ciascuna ricorre.

```
frase='Antonio Pacinotti'
dizionario={}
for lettera in frase:
     if lettera not in dizionario:
          dizionario[lettera]=1
     else:
```

```
            dizionario[lettera]+=1
for lettera_presente in dizionario.items():
      print(lettera_presente)
```

Nel codice:

viene definita una stringa frase, viene creato il dizionario vuoto, con un ciclo if si controlla se una determinata lettere è presente nel dizionario: se è assente, viene memorizzata come chiave del dizionario e si assegna il numero 1; se è già presente, si incrementa di una unità il corrispondente valore.

Siccome la frase inserita è una stringa, anche il carattere spazio bianco è considerato nel conteggio.

I set (insiemi)

I set sono delle strutture di dati con delle caratteristiche:
- oggetto con metodi propri, che può essere modificato;
- contiene un insieme di dati, anche di tipo diverso, non indicizzati e non ordinati;
- contiene dati univoci, cioè non può avere elementi con valori uguali tra loro.

```
nomeset={valore1, valore2, ..., valoreN}
```

In alternativa si può usare la funzione set() per trasformare un altro oggetto iterabile in un set.
Il set è una struttura di dati ispirata al concetto matematico di insieme: numerosi metodi associati all'oggetto insieme sono dedicati al confronto insiemistico.

Esempio:

```
frase='Antonio Pacinotti'
caratteri=set(frase)
dizionario=dict.fromkeys(caratteri, 0)
for lettera in frase:
     dizionario[lettera]+=1
for letterapresente in diz.items():
     print(letterapresente)
```

Viene creato nella seconda riga un insieme contenente i caratteri della frase; ogni carattere presente nella stringa compare una sola volta nell'insieme; viene poi creato il dizionario con il metodo fromkeys() dell'oggetto dict, cha ha due argomenti: la chiave e il valore assegnato alle singole chiavi.

Il programma è più conciso rispetto a quello visto in precedenza ed evita l'uso della struttura condizionale if…else: l'elaborazione sarà più veloce, si risparmierà il tempo necessario per rileggere ogni volta l'intero dizionario e stabilire se contenga o meno la lettera in esame .

Modificare i contenuti di un set

`add()` aggiunge un valore

`update()` aggiunge al set i valori di un altro set, eliminando gli eventuali doppioni

`pop()` elimina dal set un valore scelto a caso e restituisce il valore eliminato

`remove()` rimuove un valore specifico passato come argomento

`clear()` rimuove tutti i valori producendo un set vuoto

Ricordare: gli elementi del set non sono ordinati, quindi a ogni esecuzione l'output della stampa degli elementi di un set sarà diverso.

ESEMPIO

```
colori={‘bianco’, ‘rosso’, ‘verde’, giallo’, ‘blu’}
colori2={‘viola’, ‘verde’, ‘bianco’, rosso’}
colori.update(colori2)
colori.add(‘rosa’}
colori.pop()
colori.remove{‘viola’}
colori.clear{}
```

Metodi di insiemi:

```
insiemeA.intersection(insiemeB)
insiemeA.difference(insiemeB
insiemeA.symmetric_difference(insiemeB)
insiemeA.isdisjoint(insiemeB)
insiemeA.issubset(insiemeB)
insiemeA.issuperset(insiemeB)
```