Python
Parte 3: le collezioni di dati

Mattia Cozzi
cozzimattia@gmail.com

a.s. 2024/2025

Contenuti

Collezioni di dati (1)

Esistono diverse collezioni di dati in Python, che hanno proprietà diverse e sintassi differenti:

- liste: collezioni ordinate e modificabili; ammettono duplicati:

```
lista1 = ["Mela","Banana","Arancia","Uva"]
```

  Se una collezione è ordinata, accedo ai suoi elementi con un indice: `lista1[0]` indica il primo elemento, ecc.

- tuple: collezioni ordinate e immutabili; ammettono duplicati:

```
tupla1 = ("Mela","Banana","Arancia","Uva")
```

Collezioni di dati (2)

- set: collezioni non ordinate e quindi senza indici; non ammettono duplicati:

```
set1 = {"Mela","Banana","Arancia","Uva"}
```

- dictionary: collezioni ordinate e modificabili; non ammettono duplicati; i dati sono nella forma `chiave:valore`:

```
dizionario1 = {
  "marca":   "Ford",
  "modello": "Mustang",
  "anno":    1964
}
```

Liste

La lista è la collezione di dati più flessibile in Python, perché è ordinata (e quindi indicizzata), modificabile e ammette duplicati.

Possiamo creare una lista con oggetti dello stesso tipo, oppure con oggetti di tipo diverso (interi, float, stringhe, bool, ecc.).

Per scoprire quanti elementi contiene una lista:

```
1 lista1 = ["Mela","Banana","Arancia","Uva"]
2 print(len(lista1)) #restituisce il numero di elem. della lista
```

Stampa degli elementi di una lista (1)

Con un semplice ciclo **while** possiamo stampare gli elementi di una lista, accedendo ad essi con un indice:

```
1  frutti = ["Mela", "Banana", "Arancia", "Uva", "Pera", "Fragola",
   ↪  "Ananas", "Mango", "Kiwi", "Mirtillo"]
2  i = 0
3  while i < len(frutti):
4    print(frutti[i])
5    i += 1
```

In modo ancora più semplice:

```
1  frutti = ["Mela", "Banana", "Arancia", "Uva", "Pera", "Fragola",
   ↪  "Ananas", "Mango", "Kiwi", "Mirtillo"]
2  for x in frutti:
3    print(x)
```

Stampa degli elementi di una lista (2)

Possiamo anche accedere agli elementi di una lista dal fondo usando, come abbiamo visto per le stringhe, indici negativi.

```
1  frutti = ["Mela", "Banana", "Arancia", "Uva", "Pera", "Fragola",
   ↪  "Ananas", "Mango", "Kiwi", "Mirtillo"]
2  i = -1
3  while i > -len(frutti)-1:
4    print(frutti[i])
5    i -= 1
```

Stampa degli elementi di una lista (3)

Per accedere a porzioni di lista, facciamo come per le stringhe.

```
1  frutti = ["Mela", "Banana", "Arancia", "Uva", "Pera", "Fragola",
   ↪  "Ananas", "Mango", "Kiwi", "Mirtillo"]
2
3  print(frutti[:3]) #stampa gli elementi con indice da 0 a 2
4
5  print(frutti[7:]) #stampa gli elementi con indice da 7 alla fine
6
7  print(frutti[5:8]) #stampa gli elementi con indice da 5 a 7
```

Esercizi

1. Crea una lista di almeno 8 elementi e scrivi l'algoritmo che li stampa tutti.
2. Crea una lista di almeno 8 elementi e scrivi l'algoritmo che li stampa tutti in ordine inverso.
3. Crea una lista di almeno 8 elementi e scrivi l'algoritmo che li stampa tutti con davanti un numero crescente.
4. Crea una lista di almeno 8 elementi e scrivi l'algoritmo che stampa solo il secondo, il quarto, il sesto, ecc.
5. Crea una lista con un numero dispari di elementi e scrivi l'algoritmo che stampa i tre valori centrali. L'algoritmo deve funzionare per tutte le liste con elementi dispari.

Modificare elementi

Per cambiare il valore di un elemento, mi basta riassegnare il valore con la sintassi che già conosciamo:

```
1  frutti = ["Mela", "Banana", "Arancia", "Uva", "Pera", "Fragola",
   ↪  "Ananas", "Mango", "Kiwi", "Mirtillo"]
2
3  frutti[2] = "Mandarino"
4  #in questo modo, "Arancia" diventa "Mandarino"
```

Inserire elementi

Possiamo inserire elementi in coda o in qualsiasi altra posizione.

```
1  frutti = ["Mela","Banana","Arancia"]
2  frutti.append("Uva")
3  #per aggiungere un nuovo elemento sul fondo
4
5  frutti.insert(2, "Fragola")
6  #per inserire in posizione 2, spostando in avanti gli elementi
   ↪  successivi
7
8  nuoviFrutti = ["Cocco","Avocado"]
9  frutti.extend(nuoviFrutti)
10 #per fare un append di una intera lista ad un'altra
```

Eliminare elementi

Per eliminare elementi, esistono diversi metodi:

```
1  frutti = ["Mela", "Banana", "Arancia", "Uva", "Pera", "Fragola",
   ↪  "Ananas", "Mango", "Kiwi", "Mirtillo"]
2
3  frutti.remove("Pera") #per eliminare uno specifico elemento
4
5  frutti.pop()          #elimina l'ultimo elemento della lista
6
7  frutti.pop(3)         #elimina l'elemento di indice 3
8
9  frutti.clear()        #svuota la lista
10
11 del frutti            #elimina la lista
12
13 del frutti[2]         #elimina l'elemento di indice 2
```

Esercizi

6. Crea una lista di almeno 5 elementi, stampala e poi creane una nuova con gli elementi in ordine inverso. Stampa anche quella.
7. Crea una lista di 5 elementi tutti nulli. Chiedi poi all'utente di inserire dei valori nella lista e calcola la somma di tutti i valori inseriti. Mostra come output "La somma di X, Y, Z, ... è W".
8. Crea una lista con un numero di elementi a piacere, stampala e crea un algoritmo che chieda all'utente cosa vuole fare: modificare un elemento, aggiungerne uno in una certa posizione oppure eliminarlo. Esegui l'operazione che l'utente ha scelto e mostra la stringa modificata.

Ordinare gli elementi

Il metodo `sort()` ordina in ordine alfabetico o numerico gli elementi di una lista.

```
1  frutti = ["Mela", "Banana", "Arancia", "Uva", "Pera", "Fragola",
   ↪  "Ananas", "Mango", "Kiwi", "Mirtillo"]
2
3  frutti.sort()
4  #ordina in ordine alfabetico
5
6  frutti.sort(reverse=True)
7  #ordina in ordine alfabetico inverso
```

Esercizi

9. Chiedi all'utente da quanti elementi deve essere formata una certa lista. Scrivi successivamente un ciclo per farla riempire con valori scelti dall'utente. Quando la lista è completa, mostrala ordinata in ordine alfabetico.
10. Scrivi un algoritmo che permetta di aggiungere elementi ad una lista fino a che l'utente non digita "finito". Mostra poi la lista ottenuta in ordine alfabetico e in ordine alfabetico inverso, con un numero davanti ad ogni elemento che ne indichi la posizione nella lista.

Copiare una lista

Per creare una copia di una lista, non basta fare:

```
1 frutti1 = ["Mela","Banana","Arancia"]
2
3 frutti2 = frutti1
```

Dobbiamo invece usare il metodo `copy()` oppure il costruttore[1] `list()`:

```
1 frutti1 = ["Mela","Banana","Arancia"]
2
3 frutti2 = frutti1.copy()
4 #oppure
5 frutti3 = list(frutti1)
```

[1] In informatica, un costruttore è una funzione che crea nuovi oggetti.

Verificare la presenza di un elemento

Per controllare se un certo elemento di una lista esiste, possiamo usare la seguente tecnica:

```
1  frutti = ["Mela", "Banana", "Arancia", "Uva", "Pera", "Fragola",
   ↪  "Ananas", "Mango", "Kiwi", "Mirtillo"]
2
3  if "Ananas" in frutti:
4    print("Trovato!")
```

oppure:

```
1  frutti = ["Mela", "Banana", "Arancia", "Uva", "Pera", "Fragola",
   ↪  "Ananas", "Mango", "Kiwi", "Mirtillo"]
2
3  print("Fragola" in frutti)
```

Esercizi

11. Crea una lista che contiene dieci elementi. Chiedi poi all'utente di inserire una stringa e controlla se tale elemento è presente nella lista. Mostra poi a schermo una riga che dice se l'elemento è presente o meno.

count() e index()

Il metodo count() ci permette di contare quante volte un certo valore compare in una lista:

```
1  frutti = ["Mela", "Banana", "Arancia", "Mela", "Pera", "Fragola",
   ↪  "Ananas", "Mango", "Mela", "Mirtillo"]
2  x = frutti.count("Mela")
3  print(x)
```

Se vogliamo invece sapere la posizione di un certo elemento, usiamo il metodo index():

```
1  frutti = ["Mela", "Banana", "Arancia", "Uva", "Pera", "Fragola",
   ↪  "Ananas", "Mango", "Kiwi", "Mirtillo"]
2  x = frutti.index("Pera")
3  #se un termine compare più volte, restituisce solo la prima
   ↪  occorrenza
4  print(x)
```

Esercizi

12. Crea una lista che contiene dieci elementi, anche ripetuti. Chiedi poi all'utente di inserire una stringa e controlla se tale elemento è presente nella lista. Se è presente, scrivi un algoritmo che conta quante volte quell'elemento si presenta nella lista, usando la funzione `count()`.

Unire più liste

Per unire tra loro più liste, posso agire in diversi modi:

```
1  frutti1 = ["Mela","Banana","Arancia"]
2  frutti2 = ["Uva","Pera","Fragola"]
3
4  frutti3 = frutti1 + frutti2
5  #metodo più diretto, che crea una nuova lista
6
7  for x in frutti2:   #per ogni x di frutti2
8    frutti1.append(x) #aggiungi x in fondo a frutti1
9
10 frutti1.extend(frutti2)
11 #appendi la lista frutti2 in fondo a frutti1
```

▸ Elenco completo dei metodi delle liste

Esercizi

13. Crea due liste rispettivamente da 3 e 5 elementi. Uniscile con il metodo che preferisci e stampa la lista completa ottenuta.
14. Crea un algoritmo che permetta ad un utente di creare una lista e aggiungerci elementi finché preferisce. Chiedi successivamente se si vuole che venga mostrato il minimo o il massimo valore presenti nella lista. In base alla risposta fornita, mostra l'elemento corretto.

Tuple

Le tuple in Python sono collezioni ordinate (e quindi indicizzate) ma non possono essere modificate una volta create.

```
1 colori = ("giallo","blu","verde","rosa","nero","rosso")
```

Come per le liste, posso trovare la dimensione di una tupla con `len()` e accedere ai suoi elementi grazie al loro indice (anche negativo).

Essendo immutabili, il seguente codice darà errore:

```
1 colori = ("giallo","blu","verde","rosa","nero","rosso")
2 colori[2] = "azzurro"
```

Un piccolo trucco

Prova a valutare che cosa fa il seguente codice:

```
1  colori = ("giallo","blu","verde","rosa","nero","rosso")
2
3  colori2 = list(colori)  #list() è il costruttore per le liste
4
5  colori2[2] = "azzurro"
6
7  colori = tuple(colori2) #tuple() è il costruttore per le tuple
8
9  print(colori)
```

Allo stesso modo posso aggiungere o eliminare elementi.

Spacchettare una tupla

Spacchettare una tupla significa assegnare a diverse variabili i singoli valori contenuti nella tupla.

```
1  colori = ("giallo","blu","verde")
2  (col1, col2, col3) = colori
3  print(col1)
4  print(col2)
5  print(col3)
```

Stampare gli elementi di una tupla

Ci comportiamo esattamente come abbiamo fatto per le liste:

```
1 colori = ("giallo","blu","verde","rosa","nero","rosso")
2 for x in colori:
3   print(x)
```

oppure:

```
1 colori = ("giallo","blu","verde","rosa","nero","rosso")
2 for i in range(len(colori)):
3   print(colori[i])
```

o ancora:

```
1 colori = ("giallo","blu","verde","rosa","nero","rosso")
2 while i < len(colori):
3   print(colori[i])
4   i += 1
```

Esercizi

15. Crea una tupla a tuo piacimento e creane poi una copia identica, per poi stamparla a schermo.
16. Crea una tupla di valori numerici a tuo piacimento e crea poi un'altra tupla che contenga solo i valori della prima maggiori di 5. Mostra entrambe le tuple a schermo.

Altre operazioni sulle tuple

L'unico metodo per unire più tuple è creare una nuova tupla che ne fonde altre due:

```
1 colori1 = ("giallo","blu","verde")
2 colori2 = ("rosa","nero","rosso")
3 colori3 = colori1 + colori2
```

Per le tuple valgono inoltre i metodi `count()` e `index()` che abbiamo visto per le liste.

Possiamo inoltre cercare se un elemento è presente in una tupla abbiamo come fatto per le liste.

Esercizi

17. Crea una tupla a tuo piacimento e chiedi una stringa all'utente. Controlla se la stringa è presente nella tupla e mostra il risultato della ricerca a schermo.

Insiemi

I set (insiemi) in Python sono collezioni non ordinate (e quindi non indicizzate). È possibile aggiungere o eliminare dati, ma i dati non possono essere modificati. Non sono ammessi duplicati.

```
1  gatti = {"Spooky","Milky","Maya","Olivia","Nikolaj","Fuego"}
```

Posso trovare la dimensione di un set con `len()` ma non posso accedere ai suoi elementi con un indice.

Se provo a stampare un set, ogni volta il suo contenuto viene mostrato casualmente.

Il costruttore è ovviamente `set()`.

Accedere agli elementi di un set

L'unico modo per accedere agli elementi di un set è usare un loop:

```
1  gatti = {"Spooky","Milky","Maya","Olivia","Nikolaj","Fuego"}
2
3  for x in gatti:
4    print(x)    #elementi mostrati in ordine casuale!
```

Per controllare se un valore è presente:

```
1  gatti = {"Spooky","Milky","Maya","Olivia","Nikolaj","Fuego"}
2  print("Fuego" in gatti) #restituisce True se il valore è trovato
```

Esercizi

18. Crea un insieme con quanti elementi preferisci e stampane due volte il contenuto, dimostrando così che gli insiemi non sono ordinati.
19. Crea un insieme con quanti elementi preferisci, chiedi all'utente una stringa, controlla se è presente nell'insieme e stampa il risultato della ricerca.

Aggiungere elementi

Non ha senso aggiungere un elemento in una certa posizione (in fondo o altrove), perché gli elementi di un set non sono ordinati.

Possiamo soltanto aggiungere un elemento ad un set:

```
1  gatti = {"Spooky","Milky","Maya"}
2  gatti.add("Fuego")
```

Eliminare elementi (1)

Abbiamo due metodi per eliminare elementi:

```
1 gatti = {"Spooky","Milky","Maya","Olivia","Nikolaj","Fuego"}
2 gatti.remove("Maya")
3 #ritorna errore se provo a eliminare un valore non presente
```

oppure:

```
1 gatti = {"Spooky","Milky","Maya","Olivia","Nikolaj","Fuego"}
2 gatti.discard("Maya")
3 #NON ritorna errore se provo a eliminare un valore non presente
```

Eliminare elementi (2)

Con il metodo `pop()` possiamo eliminare un elemento casuale dal set:

```
1 gatti = {"Spooky","Milky","Maya","Olivia","Nikolaj","Fuego"}
2 eliminato = gatti.pop()
3 #il metodo pop ritorna il valore eliminato dal set
4 print("Ho eliminato " + eliminato)
5 print(gatti)
```

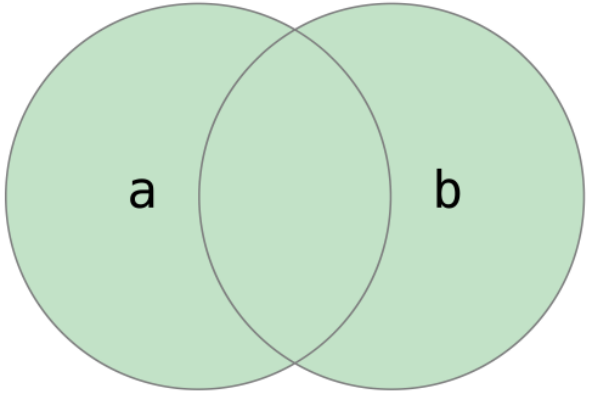
Per eliminare tutti gli elementi:

```
1 gatti = {"Spooky","Milky","Maya","Olivia","Nikolaj","Fuego"}
2 gatti.clear()
3 print(gatti)
```

Esercizi

20. Crea un insieme con quanti elementi preferisci e mostralo all'utente. Chiedi poi "Che cosa vuoi fare?".
    - Se l'utente digita "A",allora dovrà poter aggiungere un elemento, avendo controllato che esso esista.
    - Se l'utente digita "D",allora dovrà poter eliminare un elemento, avendo controllato che esso esista.
    - Se l'utente digita "S",dovrà essere mostrato l'insieme.
    - Se l'utente digita "X",l'algoritmo deve salutare e terminare.
    - Se l'utente digita altro, dovrà essere mostrato un messaggio di errore e si dovrà chiedere una nuova immissione.

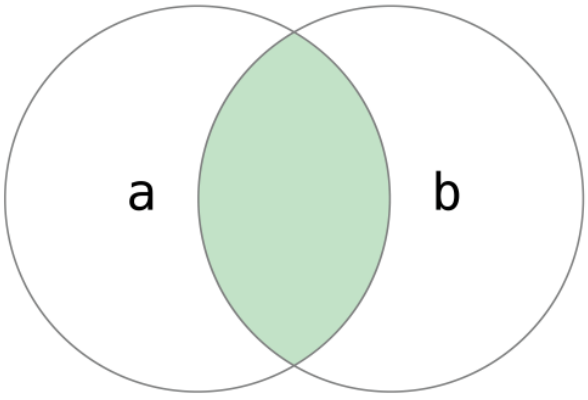
Operazioni insiemistiche: unione



Se il secondo set ha qualche valore identico al primo, il valore comparirà una sola volta nel set ottenuto.

```
1  gatti1 = {"Spooky","Milky","Maya"}
2  gatti2 = {"Olivia","Nikolaj","Fuego"}
3  gattiTot = gatti1.union(gatti2)
4  #crea l'insieme unione
5
6  #OPPURE
7  gatti1.update(gatti2)
8  #aggiunge a gatti1 tutti gli elementi di gatti2
9  #che non sono già presenti in gatti1, AGGIORNANDO l'insieme
```

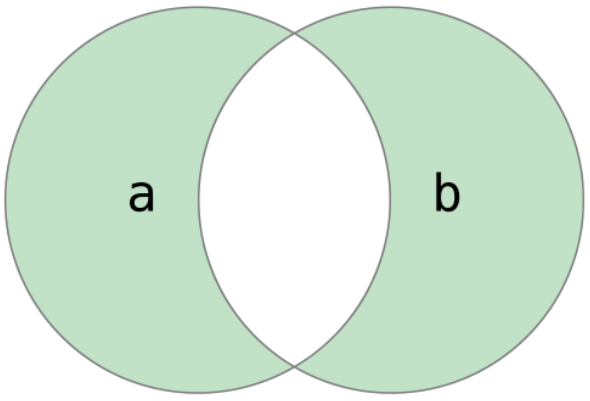
Operazioni insiemistiche: insersezione



L'intersezione tra insiemi contiene tutti gli elementi comuni ai due insiemi.

```
1  gatti1 = {"Spooky","Milky","Maya"}
2  gatti2 = {"Milky","Maya","Olivia"}
3  gattiInters = gatti1.intersection(gatti2)
4  #crea l'insieme intersezione
5
6  #OPPURE
7  gatti1.intersection_update(gatti2)
8  #elimina da gatti1 gli elementi non comuni a gatti2
9  #cioè AGGIORNA il contenuto di gatti1 con l'intersezione
```

Operazioni insiemistiche: differenza simmetrica



La differenza simmetrica, in insiemistica, genera un insieme che contiene solo gli elementi non comuni ai due insiemi di partenza.

```
1  gatti1 = {"Spooky","Milky","Maya"}
2  gatti2 = {"Milky","Maya","Olivia"}
3  gattiDiff = gatti1.symmetric_difference(gatti2)
4  #crea l'insieme differenza
5
6  #OPPURE
7  gatti1.symmetric_difference_update(gatti2)
8  #aggiorna gatti1 alla differenza simmetrica con gatti2
```

Esercizi

21. Crea due insiemi, con eventualmente elementi in comune. Stampa prima gli insiemi e poi la loro unione.
22. Crea due insiemi disgiunti e stampali. Crea poi la loro unione e stampala. Concludi poi con un algoritmo che controlla se la somma delle cardinalità del primo e del secondo insieme è uguale alla cardinalità dell'insieme unione.
23. Crea due insiemi con almeno un elemento in comune. Stampa prima gli insiemi e poi la loro intersezione.

Esercizi

24. Crea due insiemi disgiunti e scrivi un algoritmo che confermi che l'intersezione tra essi è vuota.
25. Crea due insiemi, eventualmente con elementi in comune, e scrivi un algoritmo che confermi che la cardinalità della loro differenza simmetrica è uguale alla somma delle cardinalità dei due insiemi meno il doppio della cardinalità dell'intersezione tra essi.

Altri metodi per gli insiemi

Per controllare se un insieme è contenuto in un altro insieme:

```
1  gatti1 = {"Spooky","Milky","Maya"}
2  gatti2 = {"Milky","Maya"}
3  x = gatti2.issubset(gatti1) #ritorna True oppure False
4  print(x)
```

Per controllare se un insieme contiene un altro insieme:

```
3  y = gatti1.issuperset(gatti2) #ritorna True oppure False
4  print(y)
```

Per controllare se due insiemi hanno elementi in comune:

```
3  z = gatti1.isdisjoint(gatti2) #ritorna True oppure False
4  print(z)
```

Esercizi

26. Crea un insieme di 3 elementi. Creane poi una copia e fai in modo che l'utente vi possa aggiungere due elementi. Controlla poi che il secondo insieme contenga il primo.
27. Crea un insieme di 5 elementi e un altro insieme con 2 soli elementi scelti tra quelli del primo. Crea poi la loro unione e controlla che la sua cardinalità sia esattamente 5.

Dictionaries

I dictionaries in Python sono collezioni ordinate (e quindi indicizzate), modificabili, che non ammettono duplicati.

Hanno una struttura particolare rispetto alle altre collezioni viste, perché il loro contenuto è formato da coppie `chiave:valore`.

```
1  prodotto = {
2    "nome":        "Latte",
3    "categoria":   "latticini",
4    "peso":        100,
5    "codice":      11358,
6  }
7  #posso, come al solito, usare le funzioni type() e len()
```

Sono l'equivalente degli oggetti in altri linguaggi di programmazione orientati agli oggetti (OOP).

Accedere agli elementi (1)

```
1  prodotto = {
2    "nome":        "Latte",
3    "categoria":   "latticini",
4    "peso":        100,
5    "codice":      11358,
6  }
7  print(prodotto["nome"])
8  #per stampare un singolo valore
9
10 #OPPURE
11 for x in prodotto:
12     print(x + ": " +  str(prodotto[x]))
13 #per stampare tutti i valori
```

Da questo esempio capiamo che l'indice fa riferimento alle chiavi, mentre `dict[key]` fa riferimento al valore della chiave `key` nell'oggetto `dict`.

Accedere agli elementi (2)

Per accedere agli elementi di un dict, esistono anche altri metodi:

```
1  prodotto = {
2    "nome":        "Latte",
3    "categoria":   "latticini",
4    "peso":        100,
5    "codice":      11358,
6  }
7  x = prodotto.get("categoria")
8  print(x)
```

Per ottenere tuple con tutte le chiavi e tutti i valori:

```
7  x = prodotto.keys()
8  y = prodotto.values()
9  print(x)
10 print(y)
```

Accedere agli elementi (3)

Posso anche ottenere una lista di tuple che associano ogni chiave al suo valore:

```
prodotto = {
  "nome":        "Latte",
  "categoria":   "latticini",
  "peso":        100,
  "codice":      11358,
}
x = prodotto.items()
print(x)
```

Otteniamo infatti:

```
dict_items([('nome', 'Latte'), ('categoria', 'latticini'),
↪  ('peso', 100), ('codice', 11358)])
```

Esercizi

28. Crea un dizionario che descriva la composizione di un profumo o di un prodotto cosmetico. Stampa poi a schermo la lista chiavi:valori.
29. Realizza un dizionario che esprima le caratteristiche, ingredienti compresi, di un profumo o di un prodotto cosmetico. Mostra poi i dati del prodotto formattando l'output in modo chiaro e gradevole.
30. Crea due dizionari per due prodotti, in modo simile all'esercizio precedente. Scegli poi due chiavi numeriche (come il peso, una percentuale, il prezzo o un codice) e scrivi un algoritmo che controlli, per i due prodotti creati, quale dei due ha il valore maggiore per la chiave prescelta. Dovrà mostrare "Il valore [NomeChiave] è più alto per [prodotto1/prodotto2]".

Controllare le chiavi

Per controllare se qualche chiave è presente:

```
prodotto = {
  "nome":        "Latte",
  "categoria":   "latticini",
  "peso":        100,
  "codice":      11358,
}
print("peso" in Persona) #ritorna True o False
```

Modificare gli elementi

Essendo i dict delle strutture ordinate e modificabili, possiamo accedere ai dati e modificarli.

```
1  prodotto = {
2    "nome":        "Latte",
3    "categoria":   "latticini",
4    "peso":        100,
5    "codice":      11358,
6  }
7  prodotto["peso"] = 150
8  #OPPURE
9  prodotto.update({"codice": 12345})
10 print(prodotto)
```

Aggiungere elementi

Con la sintassi che già conosciamo e usando delle nuove chiavi, possiamo anche aggiungere elementi:

```
1  prodotto = {
2    "nome":        "Latte",
3    "categoria":   "latticini",
4    "peso":        100,
5    "codice":      11358,
6  }
7  prodotto["prezzo"] = 1.99
8  #OPPURE
9  prodotto.update({"in_frigo": True})
```

Eliminare elementi

```
1  prodotto = {
2    "nome":        "Latte",
3    "categoria":   "latticini",
4    "peso":        100,
5    "codice":      11358,
6    "prezzo":      1.99,
7    "in_frigo":    True,
8  }
9  prodotto.pop("prezzo")     #elimina un elemento
10
11 prodotto.popitem()         #elimina l'ultimo elemento
12
13 prodotto.clear             #svuota il dizionario
14
15 del prodotto["categoria"] #elimina un elemento
16
17 del prodotto               #elimina l'intero dizionario
```

Esercizi

31. Crea un dict per un oggetto a tua scelta. Mostra poi tutte le chiavi presenti all'utente e chiedi se vuole modificare qualche valore. Se sì, crea un algoritmo che permetta all'utente di cambiare il valore di una chiave.
32. Crea un dict per un oggetto a tua scelta. Mostra poi tutte le chiavi presenti all'utente e chiedi se vuole Operare sui dati. Se sì, chiedi quale operazione vuole compiere: aggiunta, eliminazione o modifica di dati. Crea un algoritmo che permetta all'utente di eseguire l'operazione selezionata. Una volta eseguita, deve essere mostrato il dict modificato.

Esercizi

33. Comportati come nell'esercizio precedente, ma crea tre dict invece di uno e, all'inizio dell'algoritmo, chiedi all'utente su qualche dei tre vuole operare.

Copiare dizionari

Possiamo creare nuovi dizionari con il metodo `copy()` o con il costruttore per i dictionaries.

```
prodotto = {
  "nome":        "Latte",
  "categoria":   "latticini",
  "peso":        100,
  "codice":      11358,
}
prodotto2 = prodotto1.copy()
#OPPURE
prodotto3 = dict(prodotto)
```

Dizionari annidati

Un elemento di un dizionario può essere a sua volta un dizionario:

```
prodotto = {
  "nome":        "Latte",
  "categoria":   "latticini",
  "peso":        100,
  "codice":      11358,
  "scadenza":    {
                   "anno":   2025,
                   "mese":   "luglio",
                   "giorno": 12,
                 },
}
print(prodotto)

#per estrarre i valori di un dict annidato:
x = prodotto["scadenza"]["mese"]
print(x)
```

Esercizi

34. Realizza un dizionario relativo ad un profumo o ad un prodotto cosmetico. Usa un dizionario annidato per fornire la composizione dettagliata. Mostra i dati a schermo, con una formattazione gradevole e leggibile.
35. Crea 3 dizionari come descritto nell'esercizio precedente. Una delle chiavi deve essere un codice (da 1 a 3). Chiedi poi all'utente di inserire un numero da 1 a 3 (altrimenti viene mostrato un errore) e in base al codice inserito, mostra le informazioni del prodotto con un output gradevole e leggibile.