INDICE

I set (iterable, mutable)

set in py3kdoc

i set sono insiemi NON ordinati di oggetti NON duplicati, anche di tipo diverso!!

i set non possono contenere tutti i tipi di oggetti ma solo oggetti hashable, cioé oggetti che:

- possano essere confrontati uno con l'altro
- abbiano un id intero univoco che non cambia durante tutta la vita dell'oggetto (il valore restituito dalla funzione builtin hash)

Ad esempio tutti i tipi non-mutable di Python sono hashable (ma non solo):

In [150]:
```
a=2345
hash(a)
```

Out[150]: 2345

In [151]:
```
a=43.3
hash(a)
```

Out[151]: 1503225489

mentre le liste no ...

In [152]:
```
a=[1,2,3]
hash(a)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-152-6f718f63b227> in <module>()
      1 a=[1,2,3]
----> 2 hash(a)

TypeError: unhashable type: 'list'
```

La digressione sugli oggetti hashable è riportata solo per dare indicazioni utili a riconoscere l'errore citato sopra, che si verifica quando si cerca di creare un set che contiene elementi non compatibili con i set(l'interprete segnala un `TypeError` ed il messaggio indica `unhashable type`)

Per creare i set

literals

Si usano le parentesi graffe:

In [153]:
```
#i doppioni sono rimossi
s1={1,2,3,3,4,1}
print(s1)
```

```
{1, 2, 3, 4}
```

il set sopra contiene tutti elementi dello stesso tipo ma posso creare anche set di oggetti di tipo differente (anche se meno frequente)

In [154]:
```
s4 = {1,2,3,'a','pippo'}
print(s4)
```

```
{'a', 'pippo', 2, 3, 1}
```

la funzione set

O la funzione builtin set che accetta una sequenza come parametro:

In [155]:
```
#anche in questo caso i doppioni sono rimossi
s2=set([1,1,1,1,2,3,3,4,1])
print(s2)
```

```
{1, 2, 3, 4}
```

creo il set dei caratteri di una stringa:

In [156]:
```
s3=set('abracadabra')
print(s3)
```

```
{'a', 'r', 'b', 'c', 'd'}
```

la funzione membro copy

posso creare una copia di un set con la funzione membro copy:

In [157]:
```
s4=s3.copy()
s4==s3 , s4 is s3
```

Out[157]: (True, False)

set comprehension

per creare nuovi set esiste una set comprehension

In [158]:
```
numeri = list(range(1,100))
resti = {n%5 for n in numeri}
print(resti)
```

```
{0, 1, 2, 3, 4}
```

come per le liste è possibile inserire una condizione if:

In [159]:
```
numeri = list(range(1,100))
resti = {n%5 for n in numeri if n%5 != 1}
print(resti)
```

```
{0, 2, 3, 4}
```

oppure usare l'opereatore ternario nella prima espressione

In [160]:
```
{c.upper() if c not in 'aeiou' else c*3 for c in 'zspippolatorez' if c != 'z'}
```

Out[160]: set(['aaa', 'ooo', 'L', 'P', 'S', 'R', 'T', 'eee', 'iii'])

Modifica dei set

Aggiunta di un elemento con add

In [161]:
```
#aggiunta di un elemento
s3.add('z')
print(s3)
```

```
{'a', 'c', 'b', 'd', 'r', 'z'}
```

In [162]:
```
#aggiungo un elemento stringa
s3.add('una')
print(s3)
```

```
{'a', 'c', 'b', 'd', 'una', 'r', 'z'}
```

Aggiunta elementi con update

in questo caso la stringa non viene aggiunta per intero ma viene interpretata come set:

In [163]:
```
print(s3)
s3.update("una")
print(s3)
```

```
{'a', 'c', 'b', 'd', 'una', 'r', 'z'}
{'a', 'c', 'b', 'd', 'n', 'una', 'r', 'u', 'z'}
```

in generale posso aggiungere ad un set tutti qli elementi (uno per uno) di un'altra sequenza (una stringa è interpretata come una sequenza di caratteri).

In [164]:
```
s3.update(range(5))
s3
```

Out[164]: set(['a', 0, 'c', 'b', 4, 'd', 3, 1, 2, 'n', 'una', 'r', 'u', 'z'])

In [165]:
```
s3.update(['x',67.8])
s3
```

Out[165]: set(['a', 0, 'c', 'b', 4, 'd', 3, 1, 67.8, 2, 'n', 'una', 'r', 'u', 'x', 'z'])

Rimozione di elementi con remove

remove segnala un errore se l'elemento da togliere non c'e'

In [166]:
```
s3.remove('z')
s3
```

Out[166]: set(['a', 0, 'c', 'b', 4, 'd', 3, 1, 67.8, 2, 'n', 'una', 'r', 'u', 'x'])

rimozione di elementi con discard

discard invece rimuove l'elemento solo se c'e' e non segnala alcun errore altrimenti

In [167]:
```
s3.discard('xxx')
s3
```

Out[167]: set(['a', 0, 'c', 'b', 4, 'd', 3, 1, 67.8, 2, 'n', 'una', 'r', 'u', 'x'])

estrazione casuale di elementi con pop

pop rimuove un elemento a caso dal set e segnala errore se il set e' vuoto

In [168]:
```
print(s3)
print(s3.pop())
print(s3)
```

```
{'a', 0, 'c', 'b', 4, 'd', 3, 1, 67.8, 2, 'n', 'una', 'r', 'u', 'x'}
a
{0, 'c', 'b', 4, 'd', 3, 1, 67.8, 2, 'n', 'una', 'r', 'u', 'x'}
```

cancellazione di un set con clear

In [169]:
```
#cancella tutti gli elementi del set
s3.clear()
```

Operazioni sui set

numero di elementi nel set con len

In [170]:
```
s3=set('abracadabra')
#numero di elementi
len(s3)
```

Out[170]: 5

test di appartenenza

di un elemento ad un set

In [171]:
```
#test di appartenenza
'a' in s3
```

Out[171]: True

di un set in un altro set

In [172]:
```
n=11
s  = set(range(n))     #numeri da zero a 10
sp = set(range(0,n,2)) #numeri pari da zero a 10
sd = s - sp            #numeri dispari da zero a 10
s,sp,sd
```

Out[172]:
```
(set([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]),
 set([0, 2, 4, 6, 8, 10]),
 set([1, 3, 9, 5, 7]))
```

attenzione: l'operatore in non funziona su interi set

In [173]:
```
sp in s
```

Out[173]: False

issubset

la funzione membro
set1.issubset(set2)
restituisce True se tutti gli elementi di set1 sono anche in set2

In [174]:
```
sd.issubset(s),sp.issubset(sd)
```

Out[174]: (True, False)

issuperset

set1.issuperset(set2)
restituisce True se tutti gli elementi di set 2 sono anche in set1

In [175]:
```
s.issuperset(sp)
```

Out[175]: True

insiemistica

I set supportano le operazioni tipiche della insiemistica. Vediamole con degli esempi.

In [176]:
```
s1=set('abcdefgh')
s2=set('abcd')
s3=set('cde')
s4=set('1234')
```

Unione

restituisce un nuovo set

In [177]:
```
s1 | s2 | s3 | s4 , s1
```

Out[177]:
```
(set(['a', 'c', 'b', 'e', 'd', 'g', 'f', 'h', '1', '3', '2', '4']),
 set(['a', 'c', 'b', 'e', 'd', 'g', 'f', 'h']))
```

oppure

In [178]:
```
set.union(s1,s2,s3,s4), s1
```

```
Out[178]: (set(['a', 'c', 'b', 'e', 'd', 'g', 'f', 'h', '1', '3', '2', '4']),
           set(['a', 'c', 'b', 'e', 'd', 'g', 'f', 'h']))
```

anche questa restituisce un nuovo set, nonostante sia una funzione membro di s1:

```
In [179]: s1.union(s2,s3,s4)   ,   s1

Out[179]: (set(['a', 'c', 'b', 'e', 'd', 'g', 'f', 'h', '1', '3', '2', '4']),
           set(['a', 'c', 'b', 'e', 'd', 'g', 'f', 'h']))
```

Intersezione

```
In [180]: s2.intersection(s3)

Out[180]: set(['c', 'd'])
```

oppure

```
In [181]: s2 & s3

Out[181]: set(['c', 'd'])
```

differenza

```
In [182]: s1.difference(s2)

Out[182]: set(['h', 'e', 'g', 'f'])
```

oppure

```
In [183]: s1-s2

Out[183]: set(['h', 'e', 'g', 'f'])
```

differenza simmetrica (xor)

elementi che sono o dell'uno o dell'altro ma non di entrambi

```
In [184]: print(s1)
          print(s2)
          print(s1^s2)

          {'a', 'c', 'b', 'e', 'd', 'g', 'f', 'h'}
          {'a', 'c', 'b', 'd'}
          {'e', 'g', 'f', 'h'}
```

... etc etc ... vedere set in py3kdoc

esempio con l'insiemistica

```
In [185]: rubrica1 =
          {('Nicola',345678798),('Antonio',345678798),('Ascanio',345558798),('Armando',345678798)}
          rubrica2 =
          {('Nicola',345678798),('Fernando',345678798),('Ascanio',345558798),('Riccardo',3452348798)}

In [186]: rubrica1 | rubrica2

Out[186]: set([('Nicola', 345678798),
               ('Fernando', 345678798),
               ('Ascanio', 345558798),
               ('Riccardo', 3452348798),
               ('Antonio', 345678798),
               ('Armando', 345678798)])

In [187]: rubrica1 & rubrica2

Out[187]: set([('Nicola', 345678798), ('Ascanio', 345558798)])
```

Iterazione sui set

L'iterazione è possibile ma non c'è alcun controllo sull'ordiname di iterazione sugli elementi.

In [188]:
```
for c in set('abcdefghilmnopq'):
    print(c)
```

```
a
c
b
e
d
g
f
i
h
m
l
o
n
q
p
```

I dizionari (***dict***, mutable,iterable)

dict in py3kdoc

- I dizionari sono, in gergo informatico, ARRAY ASSOCIATIVI.
- In altre parole sono oggetti che mettono in relazione delle `chiavi` con dei `valori`
- le chiavi devono necessariamente essere oggetti *NON MODIFICABILI IN-PLACE* (per la precisione devono essere hashable, vedi l'inizio del paragrafo sui set)

Creare dizionari

dizionari vuoti

Per creare un dizionario vuoto uso le `parentesi graffe`:

In [189]:
```
emptydict = {}
print(emptydict)
```

```
{}
```

Oppure la funzione builtin dict, senza argomenti:

In [190]:
```
emptydict = dict()
print(emptydict)
```

```
{}
```

creare dizionari non vuoti con parentesi graffe

Con parentesi graffe e coppie `chiave:valore` separate da virgole

In [191]:
```
rubricatel = {'gino' : '333-45602345' ,
              'aldo' : '055-332322',
              'renato' : '???' ,
              'stringa nome' : 'stringa numero'}
print(rubricatel)
```

```
{'aldo': '055-332322', 'renato': '???', 'gino': '333-45602345', 'stringa nome': 'stringa numero'}
```

creare dizionari non vuoti usando la funzione builtin dict

dict e argomenti passati per nome

In [192]:
```
#con argomenti passati per nome, il nome e' interpretato come stringa
a = dict(one=1, two=2)
print(a)
```

```
{'two': 2, 'one': 1}
```

dict e un altro dizionario

In [193]:
```
#copio un dizionario
#indovinate perche' e' diverso da b=a?
b = dict(a)
print(b is a)
print(b==a)
```

```
False
True
```

ma alla precedente forma e' possibile aggiungere altri argomenti passati per nome (vedi sopra)

In [194]:
```
c = dict(a,tre=3)
print(c)
```

```
{'tre': 3, 'two': 2, 'one': 1}
```

dict e una sequenza di coppie (chiave,valore)

In [195]:
```
listadicoppie = [
                    (1,'uno'),#tupla di due elementi
                    (2,'due'),
                    [3,'tre'],#lista di due elementi
                    [4,'quattro']
                ]

e = dict(listadicoppie)
print(e)
```

```
{1: 'uno', 2: 'due', 3: 'tre', 4: 'quattro'}
```

In [196]:
```
#questo NON funziona
#f = dict(listadicoppie,5='cinque')
```

In [197]:
```
listadicoppie.append([5,'cinque'])
f=dict(listadicoppie)
print(f)
```

```
{1: 'uno', 2: 'due', 3: 'tre', 4: 'quattro', 5: 'cinque'}
```

DOMANDA:
date le due liste:

In [198]:
```
punti   = list(range(10))#lista dei possibili punteggi maturati in un gioco
penalty = [p**2 for p in punti]#lista delle possibili penalita' associate ai punteggi
```

cosa faccio con la seguente istruzione?

In [199]:
```
d=dict(zip(punti,reversed(penalty)))
```

Dictionary comprehension

Come per le liste ed i set esiste la possibilità di creare dizionari con la sintassi della ... comprehension

In [200]:
```
chiavi = ['a','b','c']
valori = [12,1,'pippo']
```

```
d = {c:v for c,v in zip(chiavi,valori)}
print(d)
```

```
{'a': 12, 'c': 'pippo', 'b': 1}
```

anche in questo caso è supportata la clausola if

In [201]:
```
{c:i for i,c in enumerate(list('questa è una stringa')) if c != ' '}
```

Out[201]:
```
{'a': 19,
 'e': 2,
 'g': 18,
 'i': 16,
 'n': 17,
 'q': 0,
 'r': 15,
 's': 13,
 't': 14,
 'u': 9,
 'è': 7}
```

NON sembra supportato l'operatore ternario (non so se è un baco):

In [202]:
```
{c:i*2 if (c in 'aeiou') else c:i**3 for i,c in enumerate(list('questa è una stringa')) if c !=
' '}
```

```
  File "<ipython-input-202-87cf370f0ebd>", line 1
    {c:i*2 if (c in 'aeiou') else c:i**3 for i,c in enumerate(list('questa è una stringa')) if c != ' '}
                                       ^
SyntaxError: invalid syntax
```

inserimento ed accesso agli elementi di un dizionario

Le parentesi quadrate associato ai dizionari hanno un duplice uso/significato.

Inserimento e modifica di elementi (a sx dell'operatore =)

Se usate a sinistra dell' = in una espressione di assegnamento inserisco un nuovo elemento nel dizionario:

In [203]:
```
print(d)
```

```
{'a': 12, 'c': 'pippo', 'b': 1}
```

In [204]:
```
d['centoventi']=120
print(d)
```

```
{'a': 12, 'c': 'pippo', 'b': 1, 'centoventi': 120}
```

oppure definisco un nuovo valore per una chiave esistente (se la chiave gia' c'e'):

In [205]:
```
d['centoventi']=12
print(d)
```

```
{'a': 12, 'c': 'pippo', 'b': 1, 'centoventi': 12}
```

accesso al valore associato alla chiave (in una generica espressione ed in particolare a dx dell'operatore di assegnamento =)

Se utilizzate in una generica espressione, restituiscono il valore associato alla chiave:

In [206]:
```
d['quattro']+d['centoventi']
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-206-2416a29730ee> in <module>()
----> 1 d['quattro']+d['centoventi']
```

```
KeyError: 'quattro'
```

In questo caso se utilizzo una chiave inesistente, viene segnalato un errore:

```
In [207]: #d['cinquantacinque']
```

accesso al valore con get(chiave)

Una possibile alternativa e' quella di utilizzare il metodo get che, nel caso la chiave non e' nel dizionario restituisce None o un valore di default specificato:

```
In [208]: print(d.get('quattro'))
          print(d.get('cinquantacinque'))
          print(d.get('cinquantacinque','non c\'e\''))
```

```
None
None
non c'e'
```

Eliminazione elementi in un dizionario

```
In [209]: print(d)
```

```
{'a': 12, 'c': 'pippo', 'b': 1, 'centoventi': 12}
```

del

```
In [210]: del d['a']
          print(d)
```

```
{'c': 'pippo', 'b': 1, 'centoventi': 12}
```

ma se la chiave non c'è viene segnalato un key error

```
In [211]: del d['a']
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-211-4f6db43d86e0> in <module>()
----> 1 del d['a']

KeyError: 'a'
```

pop

Se si vuole essere sicuri che non sia segnalato un errore (nel caso la chiave non ci sia) usare pop:

```
In [212]: d.pop('undici','None')
```

```
Out[212]: 'None'
```

Viste dei dizionari

Le cosiddette viste dei dizionari sono fornite dalle seguenti funzioni membro:

- dict.keys()
- dict.values()
- dict.items()

The objects returned by dict.keys(), dict.values() and dict.items() are view objects. They provide a dynamic view on the dictionary's entries,

which means that when the dictionary changes, the view reflects these changes.

In [213]:
```
d=dict(uno=1,due=2,tre=3,quattro=4,cinque=5,sei=6,sette=7,otto=8)
d
```

Out[213]:
```
{'cinque': 5,
 'due': 2,
 'otto': 8,
 'quattro': 4,
 'sei': 6,
 'sette': 7,
 'tre': 3,
 'uno': 1}
```

dict.keys()

restituisce un contenitore con tutte le chiavi del dizionario

In [214]:
```
#restituisce una sequenza con tutte le chiavi del dizionario
k=d.keys()
print(k)
```

```
dict_keys(['quattro', 'tre', 'due', 'cinque', 'otto', 'sette', 'sei', 'uno'])
```

offre una visione dinamica del dizionario:

In [215]:
```
print(k)
#inserisco un nuovo elemento
d['undici']=11
#k tiene conto dell'aggiunta
print(k)
```

```
dict_keys(['quattro', 'tre', 'due', 'cinque', 'otto', 'sette', 'sei', 'uno'])
dict_keys(['quattro', 'tre', 'due', 'cinque', 'otto', 'undici', 'sette', 'sei', 'uno'])
```

le viste sono iterabili

In [216]:
```
for k in d.keys():
    print(k)
```

```
quattro
tre
due
cinque
otto
undici
sette
sei
uno
```

la sequenza restituita da dict.keys() ha le caratteristiche dei set (ad esempio supporta gli operatori logici)

In [217]:
```
print(d.keys() & {'ventidue' , 'due' , 'quattro'})
print(d.keys() | {'ventidue' , 'due' , 'quattro'})
```

```
{'quattro', 'due'}
{'ventidue', 'quattro', 'tre', 'due', 'cinque', 'otto', 'undici', 'sette', 'sei', 'uno'}
```

dict.values()

analogamente la funzione membro values() restituisce i valori del dizionario

In [218]:
```
d.values()
```

Out[218]:
```
dict_values([4, 3, 2, 5, 8, 11, 7, 6, 1])
```

In [219]:
```
d.keys()
```

```
Out[219]: dict_keys(['quattro', 'tre', 'due', 'cinque', 'otto', 'undici', 'sette', 'sei', 'uno'])
```

NB: NON e' un caso che l'ordine dei valori restituito da dict.values() sia corrispondente a quello delle chiavi restituite da dict.keys()

ovviamente i valori restituiti da dict.values() NON hanno la caratteristica dei set (i valori non sono necessariamente hashable)

dict.items()

restituisce una sequenza di coppie chiave,valore:

```
In [220]: d.items()
```

```
Out[220]: dict_items([('quattro', 4), ('tre', 3), ('due', 2), ('cinque', 5), ('otto', 8), ('undici', 11), ('sette',
          7), ('sei', 6), ('uno', 1)])
```

il risultato e' simile a quello che si otterrebbe con:

```
In [221]: list(zip(d.keys(),d.values()))
```

```
Out[221]: [('quattro', 4),
           ('tre', 3),
           ('due', 2),
           ('cinque', 5),
           ('otto', 8),
           ('undici', 11),
           ('sette', 7),
           ('sei', 6),
           ('uno', 1)]
```

test di appartenenza a dizionari

il costrutti `in` testa l'appartenenza di un elemento alle `chiavi` di un dizionario:

```
In [231]: d = {hex(i):i for i in range(10)}
          d
```

```
Out[231]: {'0x0': 0,
           '0x1': 1,
           '0x2': 2,
           '0x3': 3,
           '0x4': 4,
           '0x5': 5,
           '0x6': 6,
           '0x7': 7,
           '0x8': 8,
           '0x9': 9}
```

```
In [232]: '0x0' in d
```

```
Out[232]: True
```

```
In [234]: 0 in d
```

```
Out[234]: False
```

Iterazione sui dizionari

al solito usiamo degli esempi:

```
In [222]: d = {i:oct(i) for i in range(10)}
          d
```

```
Out[222]: {0: '0o0',
           1: '0o1',
```

```
 2: '0o2',
 3: '0o3',
 4: '0o4',
 5: '0o5',
 6: '0o6',
 7: '0o7',
 8: '0o10',
 9: '0o11'}
```

iterazione diretta

iterare direttamente su un dizionario equivale ad iterare sulle chiavi:

In [223]:
```
for i in d:
    print(i)
```

```
0
1
2
3
4
5
6
7
8
9
```

dove nel corpo del ciclo for posso anche riferirmi ai valori

In [224]:
```
for i in d:
    print(i,' --> ',d[i])
```

```
0  -->  0o0
1  -->  0o1
2  -->  0o2
3  -->  0o3
4  -->  0o4
5  -->  0o5
6  -->  0o6
7  -->  0o7
8  -->  0o10
9  -->  0o11
```

iterazione sulle chiavi con keys()

In [225]:
```
for i in d.keys():
    print(i,' --> ',d[i])
```

```
0  -->  0o0
1  -->  0o1
2  -->  0o2
3  -->  0o3
4  -->  0o4
5  -->  0o5
6  -->  0o6
7  -->  0o7
8  -->  0o10
9  -->  0o11
```

iterazione sui valori con values()

In [226]:
```
for v in d.values():
    print(v)
```

```
0o0
0o1
0o2
```

```
0o3
0o4
0o5
0o6
0o7
0o10
0o11
```

iterazione su chiavi e valori con items()

In [227]:
```
for i,v in d.items():
    print(i,' --> ',v)
```

```
0  -->  0o0
1  -->  0o1
2  -->  0o2
3  -->  0o3
4  -->  0o4
5  -->  0o5
6  -->  0o6
7  -->  0o7
8  -->  0o10
9  -->  0o11
```