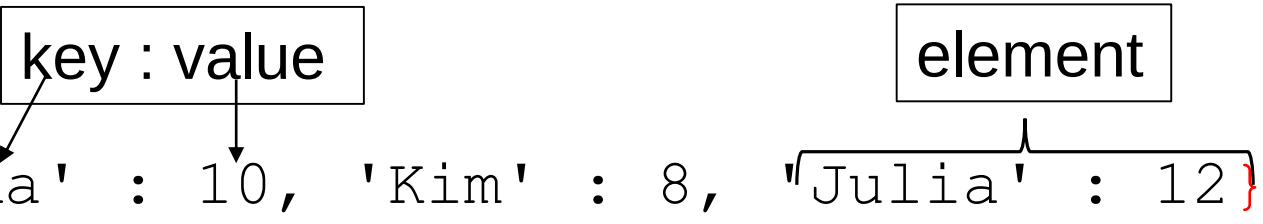


Dictionary (lexikon)

En *dictionary* är en slags tabell där man använder en söknyckel (*key*) för att komma åt information (*value*). Kallas även avbildning (avbildar *key* på *value*)



```
pers = { 'Agda' : 10, 'Kim' : 8, 'Julia' : 12 }
```

Notera {}, krullparenteserna, måsvingarna

Variabeln `pers` är av typen *dictionary* och består av tre element. Varje element består av ett par som består av: *key* och *value*, där *key* i detta exempel är en sträng och *value* är ett heltal.

Ett alternativ är att använda inbyggda funktionen **dict**:

```
pers2 = dict('Agda' = 10, 'Kim' = 8, 'Julia' = 12)
```

Utskrifterna `print(pers)` och `print(pers2)` ger samma resultat:

```
{ 'Agda' : 10, 'Kim' : 8, 'Julia' : 12 }
```

Ett annat exempel:

Tre element, där key är en sträng och value är en lista

```
students = { 'kim' : [10,12] , 'saga' : [17] , 'leo' : [] }
```

Komma åt element, görs med [] eller metoden get:

```
x = students['kim']      # alt: x = students.get('kim')
```

```
print(x)  # eller alt: print(students['kim'])
```

Ger utskrift: [10, 12]

```
x = students['torsten'] #blir fel, ty key 'torsten' finns ej
```

```
students['pia'] = [14,2]  # Lägga in nytt element:key,value
```

```
students['kim'] = [9]     # Nytt value för key
```

```
students['leo'].append(7) # Lägga till value
```

```
students['saga'].append(14)
```

```
print(students) # ger
```

```
{ 'kim': [9] , 'saga': [17, 14] , 'leo': [7] , 'pia': [14,2] }
```

Notera: *key* i ett lexikon fungerar som ett slags index för listor. Men är ej ordnade med heltal 0, 1, 2 ... Utan kan vara godtyckliga strängar och heltal

Ta bort ett element

```
students.pop('kim')    # alt: del students['kim']  
print(students)       # ger {'saga': [17, 14], 'leo': [7], 'pia': [14, 2]}
```

Loopa igenom en dictionary, skriva ut alla key's:

```
for x in students:  
    print(x)
```



saga
leo
pia

Loopa genom alla keys i dictionary, skriva ut alla values:

```
for x in students:  
    print(students[x])
```



[17, 14]
[7]
[14, 2]

Alt till ovan mha metoden **values**:

```
for x in students.values() :  
    print(x)
```

Loopa igenom en dictionary, skriva ut både key's och values med metoden **items**:

```
for x, y in students.items(): # metoden items ger en tuple  
    print(x, y)
```

x=key, y=value

saga	[17, 14]
leo	[7]
pia	[14, 2]

Kolla om key finns i dictionary:

```
if 'leo' in students:  
    print('leo is one key')  
else:  
    print('leo is not one key')
```

Längden av en dictionary, dvs antal element:

```
print(len(students))
```

→ 3

Ta bort alla element:

```
students.clear()  
print(len(students))
```

→ 0

```
d = dict([('a', 6), ('b', 88)])
```

```
print(d)
```

```
{'a': 6, 'b': 88}
```

Slå ihop två listor till en dictionary mha inbyggda funktionen **zip**:

```
per = ['Blom', 'Ek', 'Berg']
```

```
nr = [201, 204, 201]
```

```
lst = list(zip(per, nr)) # Inbyggda funktionen list
```

```
print(lst)
```

```
[('Blom', 201), ('Ek', 204), ('Berg', 201)]
```

```
tpl = tuple(zip(per, nr))
```

```
print(tpl)
```

```
((('Blom', 201), ('Ek', 204), ('Berg', 201)))
```

```
d = dict(zip(per, nr))
```

```
print(d)
```

```
{'Blom': 201, 'Ek': 204, 'Berg': 201}
```

Omvänd ordning key och value:

```
d = dict(zip(nr, per))
```

```
print(d)
```

```
{201: 'Berg', 204: 'Ek'}
```

201: 'Blom'
är ej med. Varför?

Större exempel, frekvens av ord, [wordCount.py](#)

```
s = 'Min hatt den har tre kanter tre kanter har min hatt \
    och har den ej tre kanter så är den ej min hatt'

s = s.lower()          # Alla versaler görs om gemener
lst = s.split()         # Gör en lista med alla ord, map blanka

d = {}                 # alt dict(), skapa en tom dictionary, inbyggd fkn

for w in lst:           # För varje ord w i lst
    if w in d:          # Om ord finns i d (in-operatorn)
        d[w] = d[w]+1  # Öka dess värde med 1, w är nyckel
    else:
        d[w] = 1       # Annars, nytt ord, sätt värdet 1
print(d) # ger:
```

\ → texten
fortsätter
nästa rad

```
{'min': 3, 'hatt': 3, 'den': 3, 'har': 3, 'tre': 3, 'kanter': 3,
'och': 1, 'ej': 2, 'så': 1, 'är': 1}
```

Sortera ordfrekvensen, men kan ej sortera direkt på en dictionary. Kan dock sortera map på listor. En lösning, gör först en lista med tuplar av d, så här

```
[(3, 'min'), (3, 'hatt'), (3, 'den'), (3, 'har'), ...]
```

```
lst2 = [(d[key], key) for key in d] # tuplarna blir (d[key], key)
```

Sortera listan lst2 map värdet av första elementet i tupeln

```
lst3 = sorted(lst2, key=f) # inbyggd fkn sorted  
print(lst3) # ger:
```

```
def f(e):  
    return e[0]
```

```
[(1, 'och'), (1, 'så'), (1, 'är'), (2, 'ej'), (3, 'min'), (3,  
'hatt'), (3, 'den'), (3, 'har'), (3, 'tre'), (3, 'kanter')]
```

Om man vill sortera i fallande ordning:

```
lst3 = sorted(lst2, key=f, reverse=True)
```

Några exempel:

```
dict1 = {'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5}
```

Nycklarna i en dictionary till en lista, metoden *keys*:

```
lst1 = list(dict1.keys()) # Inbyggda fkn list
print(lst1) # ['a', 'b', 'c', 'd', 'e']
```

Värdena i en dictionary till en lista, metoden *values*:

```
lst2 = list(dict1.values())
print(lst2) # [1, 2, 3, 4, 5]
```

Paret (nyckel, värde) till en lista, metoden *items*:

```
lst3 = list(dict1.items())
print(lst3) # Notera att elementen blir tuplar pga par
# [('a', 1), ('b', 2), ('c', 3), ('d', 4), ('e', 5)]
```



```
dict1 = {'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5}
```

Dictionary comprehension, exempel (jmf list-comprehension):

Gör en ny dictionary utgående från `dict1` men med value dubblerade:

```
dict2 = {k:v*2 for (k,v) in dict1.items() }
```

Loopa genom alla par (nyckel, värde), dvs `(k,v)` i `dict1`

Bilda nya par i `dict2`

Nycklarna i dictionary måste vara ***immutable***, dvs kan vara tal, strängar, tupler. Men ej listor, ty de är ***mutable***.

En dictionary är **ej ordnad**, som en lista med index.

Exempel:

```
d = {0: 'A', 2: 'B', 3: 'C'}  
print(d) # {0: 'A', 2: 'B', 3: 'C'}
```

```
d[1] = 'T' # Ny nyckel och värde  
print(d) # {0: 'A', 2: 'B', 3: 'C', 1: 'T'}
```

Jämför med lista

```
lst = ['A', 'B', 'C']  
lst.append('T') # ['A', 'B', 'C', 'T']  
lst.insert(1, 'S') # ['A', 'S', 'B', 'C', 'T']
```