Listor

Endimensionella listor

Skivningar av listor

Funktioner för listor

Metoder för listor

Listor och referenser

Loopa igenom en lista

List comprehension (listbyggare)

Inläsning till lista

Zip och Map

Tvådimensionella listor

Tredimensionella listor

Endimensionella listor

En lista består *element* och varje element har en plats som anges av *index*. Det första elementet har index 0. Elementen kan vara av olika typ.

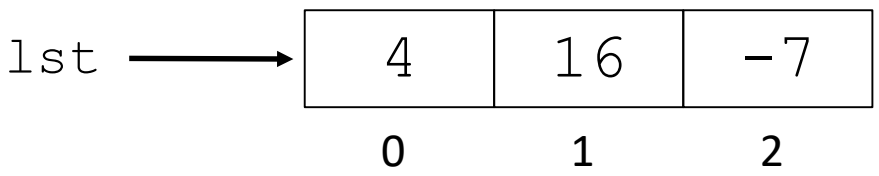
Exempel kod:

```
lst2 = []          # Skapa en variabel lst2 som är tom
lst = [4,16,-7]    # Skapa en variabel med namnet lst
print(lst)         # Skriv ut variabeln lst
print(type(lst)) # Skriv ut typen för variabeln lst
```

Utskrift:

```
[4, 16, -7]
<class 'list'>
```

Kan illusteras på följande sätt:



Variabeln `lst` har tre element, `lst[0], lst[1], lst[2],` alla är av typen `int`

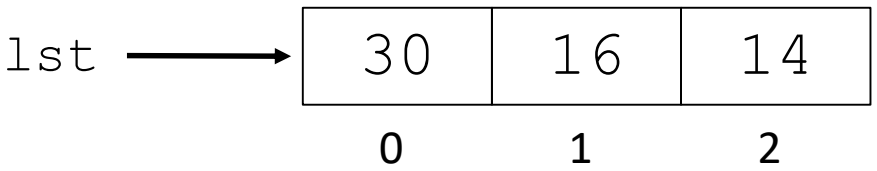
Hantering av enskilda element i en lista:

```
lst = [4,16,-7]
print(lst[1])          # skriv ut element nr 2, har index=1
lst[2]=14              # uppdatera element nr 3.
print(lst)             # Skriv ut lst
lst[0]=lst[1]+lst[2] # uppdatera första elementet
print(lst)             # Skriv ut lst
```

Utskrift:

```
16
[4, 16, 14]
[30, 16, 14]
```

Så här ser `lst` ut nu:



Legala index i en lista:

```
lst = [4,16,-7]   # Legala index är 0…2, ty lst har 3 element
lst[3]=99         # Men …
```

Ger felmeddelande:

```
    lst[3] = 99
IndexError: list assignment index out of range
```

Listans längd, dvs antal element, ges av funktionen **`len`**:

```
print(len(lst))   # Ger utskrift: 3
```

Ändra sista elementet, utan att "veta" listans längd:

```
lst[len(lst)-1]=8   # Alt 1
lst[-1]=8           # Alt 2, lite enklare
```

Ändra näst sista elementet, utan att "veta" listans längd:

```
lst[len(lst)-2]=8   # Alt 1
lst[-2]=8           # Alt 2, lite enklare
```

`len`, inbyggd funktion i Python

Exempel på tilldelningar:

```
lst = [0] * 5                    # Ger [0,0,0,0,0]

k = 7
lst = [k, 2*k, 3*k, 4*k]         # Ger [7, 14, 21, 28]
```

Skivningar

```
lst = [7, 14, 21, 28]

lst2 = lst[1:]                  # Ger [14, 21, 28]

lst2 = lst[1:3]                 # Ger [14, 21]

lst3 = lst2 + lst[0:1]          # Ger [14, 21,7]

lst = [1,2,3,4]
lst2 = lst[3:0:-1]              # Ger [4, 3, 2]

lst[0:2] = [14,99]              # Ger [14, 99, 3, 4]
```

Allmänt: `[forst:sist:steg]` kallas *skivning*

Metoder för listor, några exempel:

```
v1 = [9,14,6]
v1.append(23)       # [9,14,6,23]
v1.insert(1,17)     # [9,17,14,6,23]
x = v1.pop()        # [9,17,14,6], eller bara v1.pop()
v1.extend([15,3]) # [9,17,14,6,15,3]
x = v1.pop(2)       # [9,17,6,15,3]
v2 = v1.copy()
v2.sort()           # [3,6,9,15,17]

names = ['Kim', 'Bo', 'Torsten', 'Agda']
names.sort()                  # ['Agda', 'Bo', 'Kim', 'Torsten']
names.sort(reverse=True) # ['Torsten', 'Kim', 'Bo', 'Agda']
names.sort(key=len)           # ['Bo', 'Kim', 'Agda', 'Torsten']
```

Notera: *Metoder* anropas med punktnotation, ex. `v1.pop()`

Inbyggda funktioner för listor, några exempel:

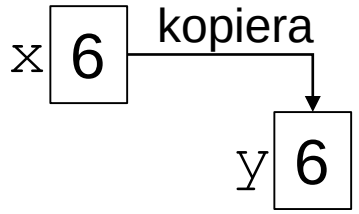
```
v3 = list('kim')  # ['k','i','m'] Gör en lista av strängen
v1 = [9,14,6]
x = max(v1) # 14
x = min(v1) # 6
x = sum(v1) # 29
v2 = sorted(v1) # v2 blir [6,9,14]
```

Notera: *Funktioner* anropas utan punktnotation

Listor och referenser

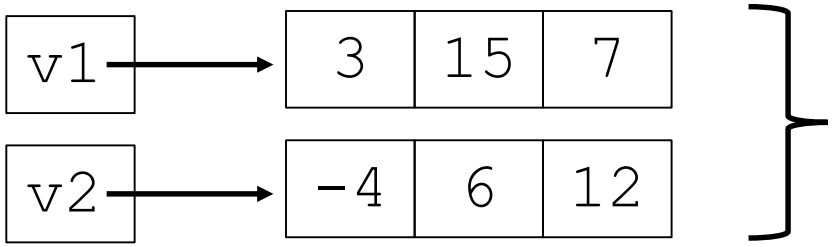
Kopiering mellan enkla variabler:

```
x = 6  # x är en int
y = x  # värdet 6 kopieras från x till y
print(x,y) # Ger 6 6
```



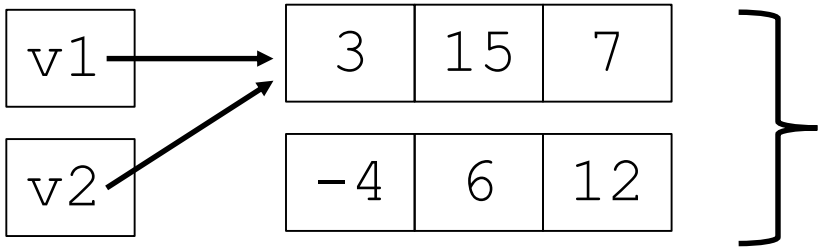

```
x = 3
print(x,y) # Ger 3 6
```


```
v1 = [3,15,7]
v2 = [-4,6,12]
```



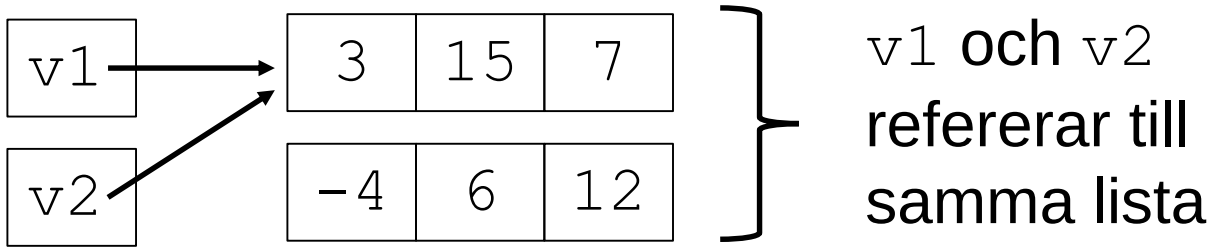
v1 och v2 refererar till varsin lista

```
v2 = v1
```



v1 och v2 refererar till samma lista

Forts…

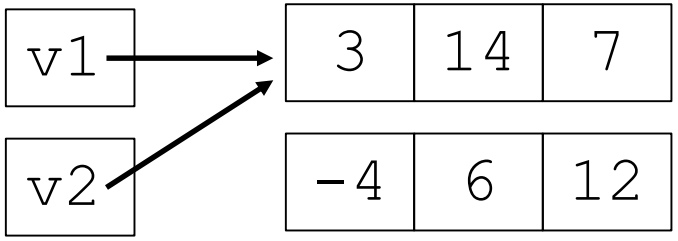


`v1` och `v2` refererar till samma lista

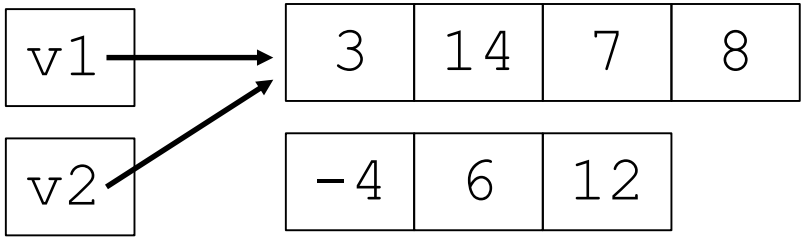
Kommer man åt listan `[-4,6,12]` ?

Vi har tappat bort listan som `v2` tidigare refererade till

`v1[1]=14`



`v2.append(8)`



Loopa (iterera) igenom en lista:

```
lst = [4,16,-7]
for el in lst:  # För varje element (el) i lst
    print(el)   # Skriv ut el
```

Utskrift:

```
4
16
-7
```

Alt 2 (mha funktionen enumerate)

```
for i, el in enumerate(lst): # För varje index och element
    print(i,el)   # Skriv ut index och element
```

Utskrift:

```
0 4
1 16
2 -7
```

Funktionen enumerate returnerar två värden för varje iteration

Alt 3:

```
for i, el in enumerate(lst,100): # Starta med index=100
    print(i,el)
```

Utskrift

```
100 4
101 16
102 -7
```

Alt 4:

```
for tpl in enumerate(lst): # För varje tuple (index,element)
    print(tpl)   # Skriv ut tuple (parvärdet)
```

Utskrift:

```
(0, 4)
(1, 16)
(2, -7)
```

List comprehension (listbyggare)

Skapa en lista `v2` genom att utgå från en befintlig lista `v1` och utföra ett visst uttryck på alla element i listan `v1`.

```
v1 = [1,2,3,4]
v2 = [x*x for x in v1]
# v2 blir [1,4,9,16]
```

För varje element `x` i listan `v1`: Beräkna `x*x` som elementens värde i listan `v2`

```
v2 = [x*x for x in v1 if x>2]
# v2 blir [9,16]
```

```
v2 = [x+4 for x in v1 if x%2==0]
# v2 blir [?]
```

Svar: `[6,8]`

```
v2 = [i*x for i,x in enumerate(v1)]
# v2 blir [0, 2, 6, 12]
```

För varje index (i) och element (x): Beräkna i*x

Listbyggare forts, utöka lista

```
v1 = [2, 3, 4]
v2 = [ [0]*3 + v1 + [0]*3 ]
# v2 blir [0, 0, 0, 2, 3, 4, 0, 0, 0]
```

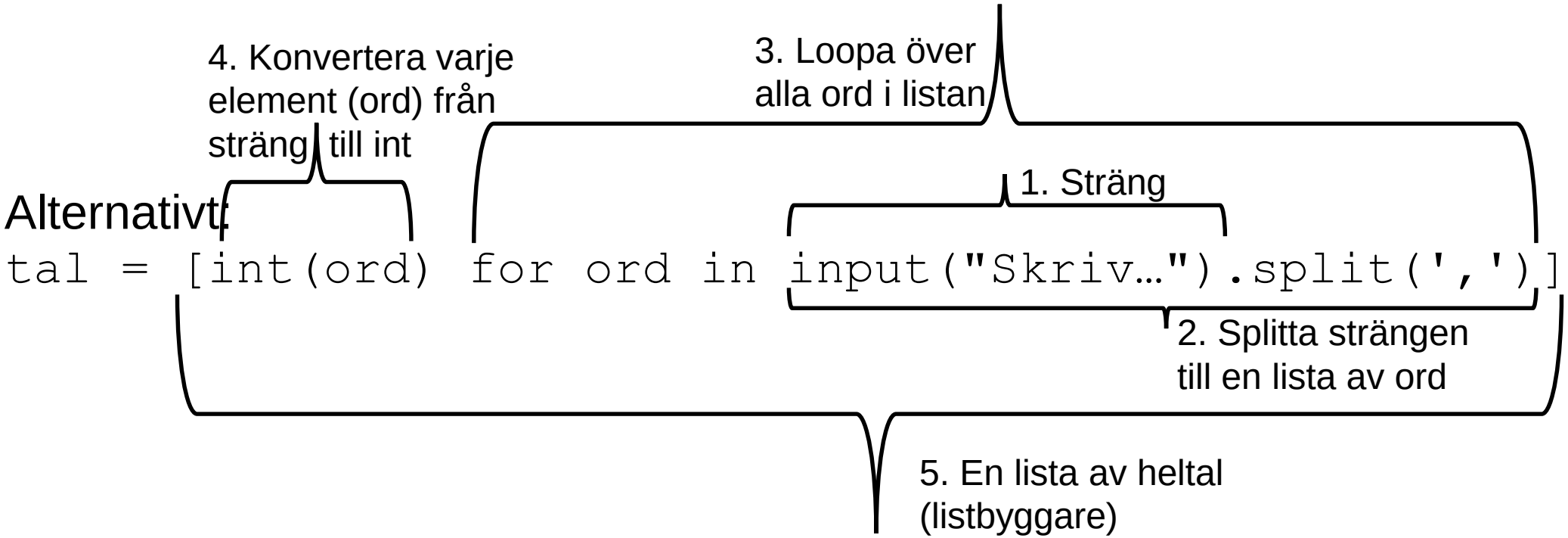
Inläsning till lista

Ett program som fungerar på följande sätt:

```
Skriv ett antal heltal, kommatecken mellan: 2, 1, 4, 3
Du skrev 4 tal
Max av talen: 4
Summan av talen: 10
```

```
s = input("Skriv ett antal heltal, kommatecken mellan:")
# s blir en sträng, exvis "1, 2, 3, 4"
s2 = s.split(',')# s2 blir en lista ['1', '2', '3', '4']
# Gör om alla ord (strängar) till heltal och lägg i en lista
tal = [int(ord) for ord in s2] # listbyggare
print("Du skrev", len(tal), "tal")  # Använder de inbyggda
print("Max av talen:", max(tal))     # funktionerna len,
print("Summan av talen:", sum(tal))  # max och sum
```

Det hela kan skrivas som en sats, där det utförs i fem steg, 1-5.




```
print("Du skrev", len(tal), "tal")
print("Max av talen", max(tal))
print("Summan av talen", sum(tal))
```

Exempel

Antag att vi har två listor med strängar

```
swedish= ['hund', 'hej', 'katt']
english= ['dog', 'hello', 'cat']
```

Vi vill skapa listan `longest` som väljer de längsta orden i varje par från de två listorna, dvs

```
longest = ['hund','hello','katt']
```

Vi har följande funktion till hjälp:

```
def longestString(str1, str2):
    """Returnerar den längsta strängen av två."""
    if len(str1) > len(str2):
       return str1
    else:
       return str2
```

Lösning 1, enkel loop

```
longest = []
for i in range(len(swedish)): # För varje par i listorna
    longest.append(longestString(swedish[i], english[i]))
```

- Behöver ett index (`i`) när vi faktiskt bara vill iterera över hela listan
- Lite problem att vi har TVÅ listor vi vill iterera över samtidigt

Lösning 2, med zip (inbyggd funktion)

```
longest = []
for (s1, s2) in zip(swedish, english):
    longest.append(longestString(s1, s2))
```

- `zip` skapar en iteration över par (tuplar) från två itererbara objekt (tänk tänderna i ett blixtlås som möts)

Lösning 3, listbyggare

```
longest = [longestString(s1,s2) for (s1,s2) in \
                                    zip(swedish,english)]
```

Lösning 4

```
longest = list(map(longestString, swedish, english))
```

- `map` är inbyggd funktion i python som kan ta flera itererbara objekt samtidigt (`swedish` och `english`) och var och en av dem bidrar då med en parameter till funktionen (`longestString`)

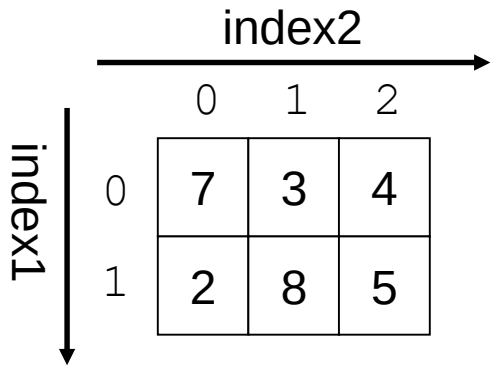
- Att använda färdiga verktyg från Python gör koden
  - Kortare
  - Mer entydig
  - Lättare att förstå (när man känner igen mönstren)

Tvådimensionella listor

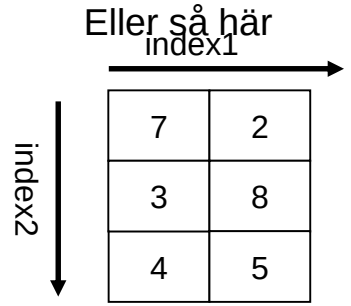
Notera: Yttre och inre par av hakparenteser

```
t = [ [7,3,4], [2,8,5] ]
print(t)  # ger [[7, 3, 4], [2, 8, 5]]
```

Kan ses som en tabell (matris):



Eller så här



Elementen är int:

1:a index 2:a index

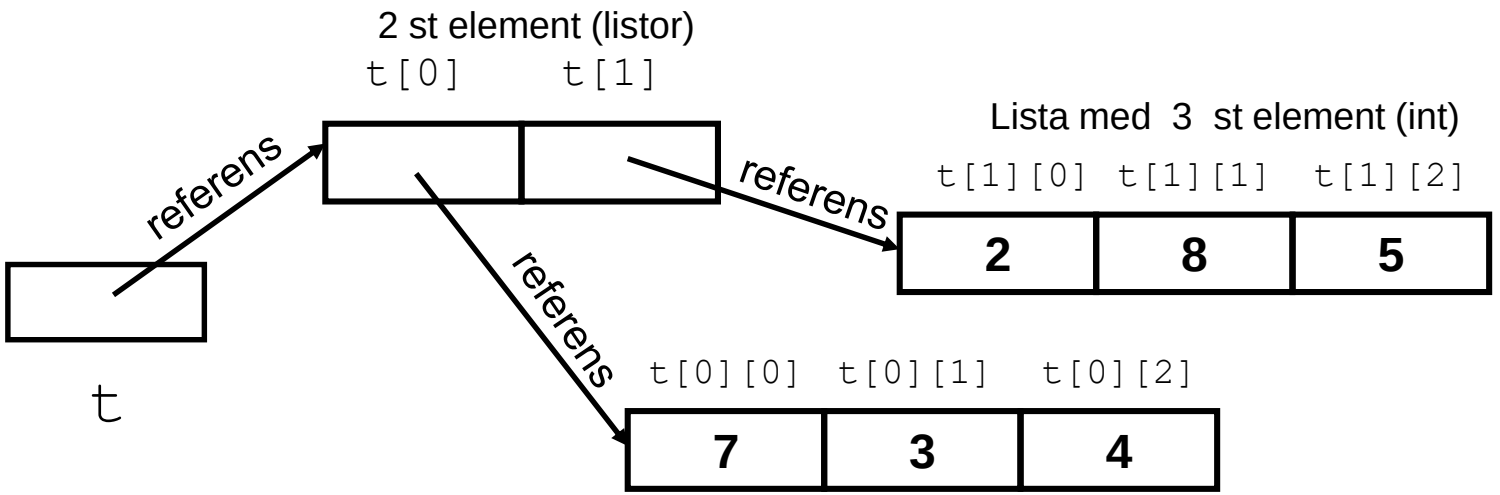
```
t[0][0]=7, t[0][1]=3, t[0][2]=4,
t[1][0]=2, t[1][1]=8, t[1][2]=5.
```

```
t = [ [7,3,4] , [2,8,5] ];
```

2 st element

Listan `t` har två ”element” `t[0]` resp. `t[1]` , där varje sådant element är en lista med ett index som i sin tur refererar till tre element som är heltal.

Så här kan det illusteras:



Skriv ut listan `t:`

```
print(t)        # [[7, 3, 4], [2, 8, 5]]
print(t[0])     # [7, 3, 4]
print(t[1])     # [2, 8, 5]
print(t[0][1]) # 3
```

Dubbelloop, loopa igenom alla element:

```
for i1 in range(0,2): # radloop
    for i2 in range(0,3): # kolumnloop
        print(t[i1][i2], end =' ')
    print() # Radbyte
```

Ger:

```
7 3 4
2 8 5
```

Alt dubbeloop:

```
for x in t: # rad
    for y in x: # kolumn
        print(y, end = ' ')
    print()
```

```
t = [ [7,3,4], [2,8,5] ]
```

Summera elementen

```
print(sum(t))  # Ger avbrott, kan ej summera 2D-listor
print(sum(t[0])# Ger 14, som är summan av första raden
```

```
s = 0
for x in t:
    for y in x:
        s = s + y
```

s blir 29

```
for x in t:
    s = s + sum(x) # summera summan av varje rad
```

```
s = sum( [sum(x) for x in v] )
```

Bygg upp 1D-listan: `[14,15]`. Därefter summan av dessa

Sudoku
En variabel som representerar en sudoku-spelplan (9x9 rutor):

Alla element sätts till noll, tomma rutor är noll.

```
sud = [[0,0,0,0,0,0,0,0,0,], … ]
```

Alt:

```
sud = [[0 for x in range(9)] for y in range(9)]
```

Sätt värdena 1-9 i översta raden:

```
for x in range(9):
    sud[0][x] = x+1
```

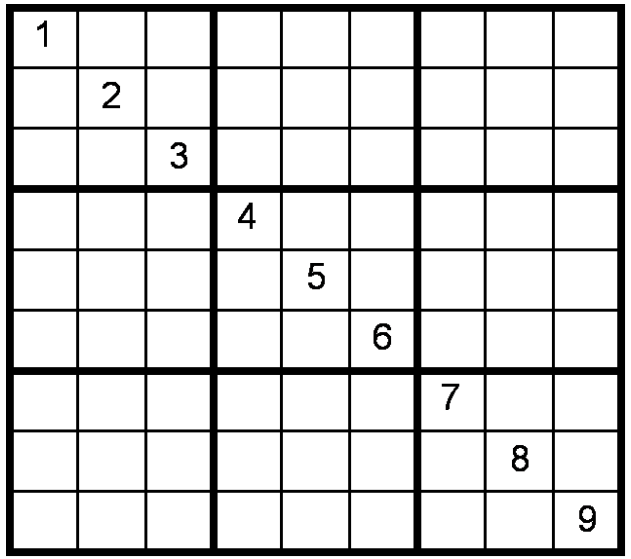
```
def printSud(sud):
   for r in range(9):
        print(sud[r])
```

Låtsas att det är nollor i alla tomma rutor i figuren till höger

1	2	3	4	5	6	7	8	9

Fundera på:

Fyll diagonalen med värdena 1-9.



Låtsas att det är nollor i alla tomma rutor i figuren till vänster

```
for x in range(9):
    sud[x][x] = x+1
```

Fundera på:
Lägg in värden 1-9 i tre av delar av spelplanen enligt nedan.

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

Att fylla **en** kvadrat är **ett** delproblem. Att fylla de tre kvadraterna är egentligen att göra samma sak (lösa samma problem) tre gånger, vilket kan uttryckas med en loop som snurrar tre varv, där varje varv består av en kvadratfyllning.

```
sud = [[0 for x in range(9)] for y in range(9)]
#  Fyll kvadranter
fill_qv(0,0,sud)
fill_qv(3,3,sud)
fill_qv(6,6,sud)
```

Lös ett sudoku

a) Skapa en sudoku-spelplan som fylls med slumpmässiga siffror 1-9 i alla rutor. Gör ingen kontroll av siffrorna.

b) Bygg på programmet: Räkna hur många rader som är godkända dvs innehåller alla siffror 1-9. (tips på räknare, se nästa sida)

c) Bygg på programmet: Räkna hur många kolumner som är godkända, dvs innehåller alla siffror 1-9.

d) Bygg på programmet: Räkna hur många av de 9 st 3x3 rutorna (kvadraterna) som är godkända.

e) Med räknarna i b)-d) kan du kontrollera ifall hela sudoku't är korrekt.

Gissningsvis är det ytterst sannolikt att det blir någon rad, kolumn eller kvadrat som bli godkänd.

Testa därför programmet genom att listan istället har fixa värden för alla element, istf slumpvalda, så att någon rad, kolumn eller kvadrat blir godkänd.

Tips på räknare för sudoku

Alt 1:

```
count = [0]*9 # Alla nio element får värdet 0.
```

`count` kan användas för att räkna antal element med värdet 1-9, om `count[0]` är antalet 1:or, `count[1]` är antalet 2:or, etc

Exempel:

```
dig=4;
count[dig-1] = count[dig-1] + 1;
```

Ovan sats ökar räknaren med ett för värdet 4

Alt 2 (fiffigare för sudoku):

```
exists = [False]*9 # Alla element är false.
```

Variabeln `exists` kan användas för att bokföra huruvida ett siffra (1-9) redan är valt. Om `exist[0]` är `true` betyder det att siffran 1 redan är vald.

Exempel:

```
int dig=4;
if (exists[dig-1]==true)…
```

så är siffran 4 redan vald

Annars så väljer vi siffran och gör:

```
exists[dig-1]=true
```

Dvs sätter att siffran 4 nu är vald.

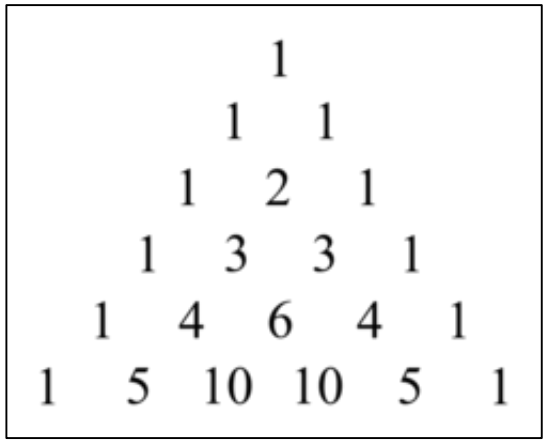
*-uppgifter

4. Skriv ett program som fyller ett från början tomt sudoku med nummer efter nummer på följande sätt:
   Börja med att slumpa en siffra (dig1) och placera dig1 i övrevänstra positionen, dvs i första raden, första kolumnen. Därefter slumpa nästa siffra (dig2), som ej får vara dig1, och placera i första raden, andra kolumnen. Fortsätt enligt samma princip så att den översta raden är fylld.
   Fortsätt därefter med nästa rad, men förutom att testa att unika nummer fås i den raden, måste även numret vara unikt med avseende på kolumnen och aktuell kvadrat.
   Fortsätt med nästa rad…

   Notera: Detta program kommer att ta lång tid, kommer kanske aldrig att sluta, men gör lämpliga utskrifter så att du ser hur programmet löpande fyller sudoku't.

Ex. Pascals triangel, $(a+b)^n$ binomialsatsen

Koefficienterna i Pascals triangel (de sex första raderna)



1					
1	1				
1	2	1			
1	3	3	1		
1	4	6	4	1	
1	5	10	10	5	1

```
p = [ [1], [1,1], [1,2,1], [1,3,3,1],
                    [1,4,6,4,1], [1,5,10,10,5,1] ]
```

`len(p)` har värdet 6
`len(p[0])` har värdet 1
Utskrift av `p`:

```
for el in p:
    print(el)
```

```
[1]
[1, 1]
[1, 2, 1]
[1, 3, 3, 1]
[1, 4, 6, 4, 1]
[1, 5, 10, 10, 5, 1]
```

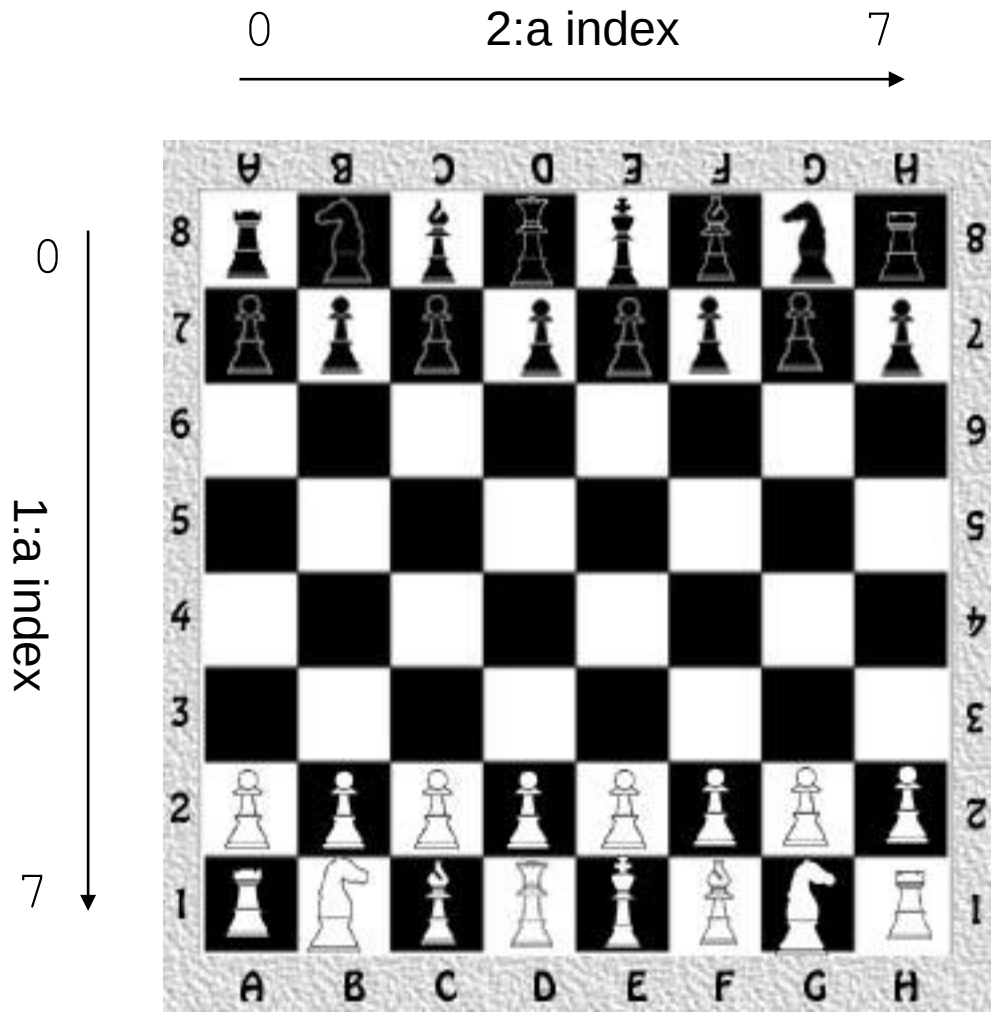
```
p = [ [1], [1,1], [1,2,1], [1,3,3,1],
                  [1,4,6,4,1], [1,5,10,10,5,1] ]
```

Skapa nästa rad i `p`, dvs elementet `[1, 6, 15, 20, 15, 6, 1]`

```
n = len(p)
# Nästa rad skall innehålla n+1 st element
el = [1]   # Nästa rad's första element är 1
for i in range(0,n-1):  # Loopa i=0..(n-1)
    el.append(p[n-1][i]+p[n-1][i+1]) # Mellanliggande element
el.append(1)  # Sista elementet är 1

p.append(el) # Lägg till det nya elementet (raden)
```

Ex. schackspel



Schack fortsätter→

Listor: (Torsten Andersson, IT-inst, Uppsala Universitet)

Ex, hur ett schackbräde kan representeras

Skapar en tvådimensionell lista, där varje element är en sträng bestående av ett tecken, dvs 8 ggr 8 element, 8 rader, 8 kolumner. Varje tecken är första bokstaven i pjäsens namn:
Torn, Springare, Löpare, Dam, Kung, Bonde
Versaler är svarta pjäser. Gemener är vita pjäser.
Låt tecknet `'_'` (understrykningstecknet) betyda tom plats.

```
board =  [
           ['T','S','L','D','K','L','S','T'],
           ['B','B','B','B','B','B','B','B'],
           ['_','_','_','_','_','_','_','_'],
           ['_','_','_','_','_','_','_','_'],
           ['_','_','_','_','_','_','_','_'],
           ['_','_','_','_','_','_','_','_'],
           ['b','b','b','b','b','b','b','b'],
           ['t','s','l','d','k','l','s','t']
         ]
```

Schack fortsätter→

Vi skriver ut startbrädet

```
for s in board:
    print(s)
}
```

Ger följande resultat:

```
['T', 'S', 'L', 'D', 'K', 'L', 'S', 'T']
['B', 'B', 'B', 'B', 'B', 'B', 'B', 'B']
['_', '_', '_', '_', '_', '_', '_', '_']
['_', '_', '_', '_', '_', '_', '_', '_']
['_', '_', '_', '_', '_', '_', '_', '_']
['_', '_', '_', '_', '_', '_', '_', '_']
['b', 'b', 'b', 'b', 'b', 'b', 'b', 'b']
['t', 's', 'l', 'd', 'k', 'l', 's', 't']
```

Versaler = svarta pjäser
Gemener = vita pjäser

Schack fortsätter→

Listor med trippla index

Exempel på tredimensionella data:

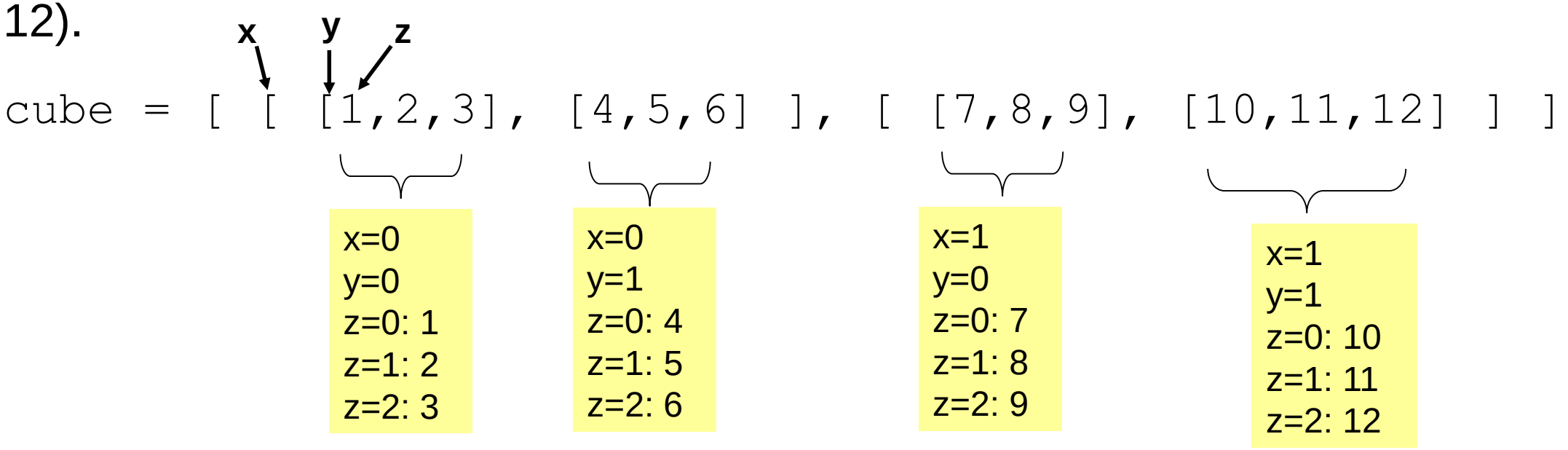
- Bilder (medicinska bilder med *voxlar,* jmfr *pixlar*)
- Spel (ex. 3D luffarschack, 3D sudoku)
- 2D-data med flera lager

  t.ex. tidslager med sudoku, schack

Lista med 3 dimensioner

En variabel med tre index kan betraktas som en variabel som representerar data med tre dimensioner, t.ex. en volym med x,y och z.

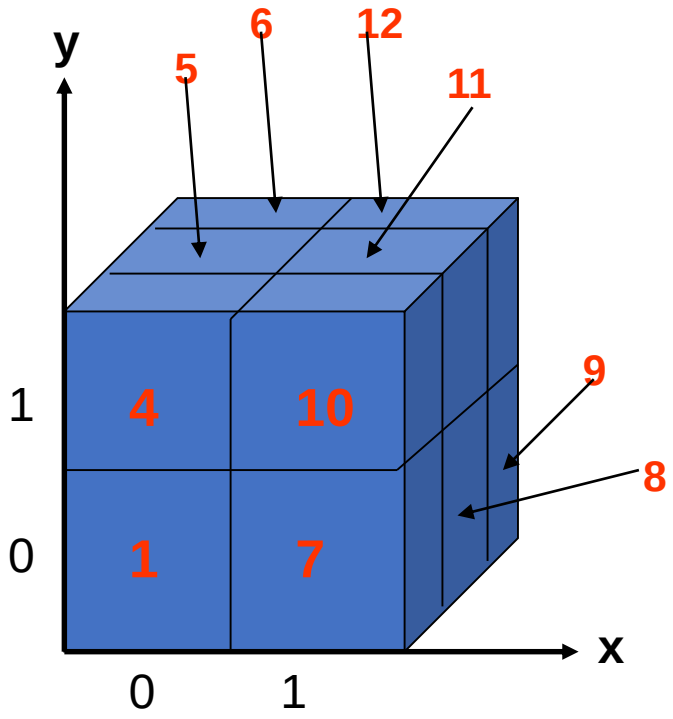
Exempel: En variabel som representerar ett rätblock med bredden (x=0-1), höjden (y=0-1) och djupet (z=0-2), dvs 2x2x3 diskreta element (värdena är 1-12).



Vad är `cube[0]` och `cube[1]`?

Forts. →

Så här kan variabeln `cube` visualiseras



Elementen `cube[0][0][1]=2` och `cube [0][0][2]=3` syns ej i bilden ovan, eftersom dessa element är skymda (i den valda projektionen)