

Håndtere mange verdier

Lister, mengder og ordbøker. Samt et lite frempek om objekter og tjenester.

IN1000, uke3
Geir Kjetil Sandve

Hva vi har lært så langt

- Variabler
- Hvordan uttrykk evaluerer til verdier
- Kontrollflyt
- Beslutninger
- Prosedyrer

Plan for dagen

- Objekter og tjenester
- Lister
- Mengder
- Ordbøker
- Nøstede samlinger

Plan for dagen

- Objekter og tjenester
- Lister
- Mengder
- Ordbøker
- Nøstede samlinger

Objekter

- IN1000 er et kurs i objekt-orientert programmering
 - Det å lage, bruke og forstå objekter vil være sentralt senere i faget
 - For nå nøyer vi oss med hvordan vi kan bruke tjenester som objekter tilbyr
- Verdier er objekter og tilbyr ofte nyttige tjenester
 - Noen objekter er konkrete og rett frem, slik som en tekstverdi
 - Andre objekter er mer abstrakte og sammensatte, slik som et programvindu med menyer osv.
- Punktum benyttes for å aksessere tjenestene (metodene) til et objekt

Eksempel på tjenester som tekstverdier tilbyr

- **upper** gjør om til store bokstaver
 - `setning = "hallo"`
`print(setning.upper())` #HALLO
- **count** teller hvor mange ganger man finner et bestemt tegn
 - `setning = "kykkeli ky"`
`print(setning.count("k"))` #4

Mer abstrakte objekter: tegne på skjermen

- Mer abstrakte objekter: vindu og lerret
 - Med *print* kommer utskriften rett til terminalen
 - For å tegne figurer må man lage et eget vindu
 - Man lager først et vindu-objekt som representerer et område på skjermen
 - Deretter får man fra vinduet et lerret-objekt som er det som holder selve tegningene (en av tjenestene et vindu tilbyr er å lage lerret)
- Hvordan dette ser ut i kode:
 - `vindu = GraphicsWindow()`
 - `lerret = vindu.canvas()`

Tegne på skjermen

- For å kunne tegne i praksis må vi hente en kodefild:
 - Muligheten for å tegne på enkel måte følger ikke med Python
 - Vi trenger derfor en ekstra pakke for dette formålet - læreboka (og vi) bruker ezgraphics
- Hvordan få på plass pakken vi trenger:
 - Legg (last ned) "*ezgraphics.py*" i samme mappe som python-programmet som skal tegne noe
 - I programmet: `from ezgraphics import GraphicsWindow`
- Deretter lager man objekter og begynner å tegne:
 - Lag vindu og lerret (canvas)
 - Tegn rektangler o.l. på lerretet
- {grafikk.py}
- *(merk forøvrig at både utskrift og tegning ikke er blant det sentrale i faget - ikke fokuser mye på det)*

Plan for dagen

- Objekter og tjenester
- **Lister**
- Mengder
- Ordbøker
- Nøstede samlinger

Sjonglere med flere verdier

- {hoyde1.py}

Finne verdien vi trenger direkte

```
hoydeAar0 = 50  
hoydeAar1 = 76  
hoydeAar2 = 87  
hoydeAar3 = 96
```

```
alder = int(input("Hvilken alder vil du vite hoyden  
for (0,1,2 eller 3 aar)? "))
```

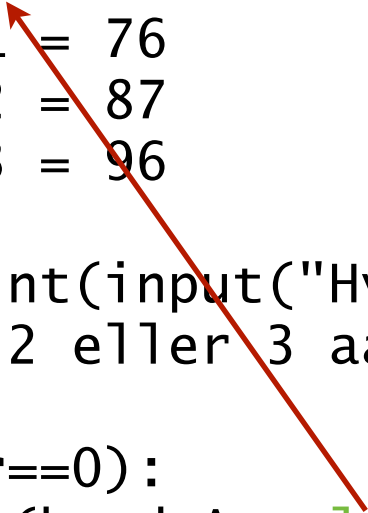
```
if (alder==0):  
    print(hoydeAar0)  
elif (alder==1):  
    print(hoydeAar1)  
elif (alder==2):  
    print(hoydeAar2)  
elif (alder==3):  
    print(hoydeAar3)
```

Finne verdien vi trenger direkte

```
hoydeAar0 = 50  
hoydeAar1 = 76  
hoydeAar2 = 87  
hoydeAar3 = 96
```

```
alder = int(input("Hvilken alder vil du vite hoyden  
for (0,1,2 eller 3 aar)? "))
```

```
if (alder==0):  
    print(hoydeAaralder)  
elif (alder==1):  
    print(hoydeAar1)  
elif (alder==2):  
    print(hoydeAar2)  
elif (alder==3):  
    print(hoydeAar3)
```



Vi kan slå opp verdien vi trenger direkte!

- Det vi ønsket:
 - `hoydeAar`**alder**
- Syntaks i Python:
 - `hoydeAar[alder]`
- Og før dette må vi definere `hoydeAar` som en liste:
 - `hoydeAar = [50, 76, 87, 96]`

Håndtere høydene i en liste

- {hoyde2.py}

Liste

- Definere en liste:

-

Liste

- Definere en liste:
 - hoydeAar = [50, 76, 87, 96]

0	50
1	76
2	87
3	96

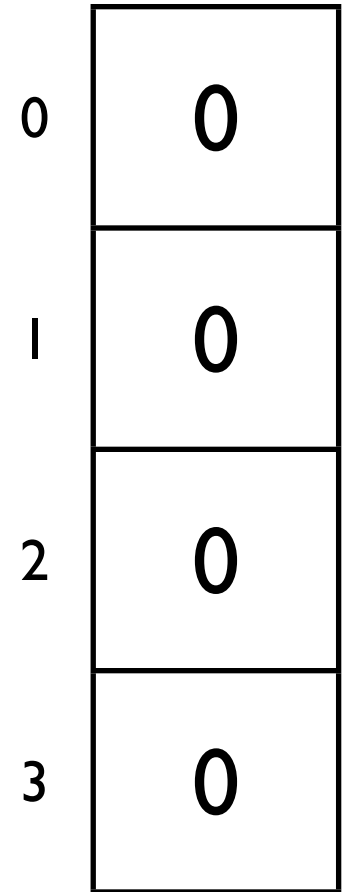
Liste



- Definere en liste:
 - `hoydeAar = [50, 76, 87, 96]`
 - `hoydeAar = []`

Liste

- Definere en liste:
 - `hoydeAar = [50, 76, 87, 96]`
 - `hoydeAar = []`
 - `hoydeAar = [0] * 4`



Liste

- Definere en liste:
 - `hoydeAar = [50, 76, 87, 96]`
 - `hoydeAar = []`
 - `hoydeAar = [0] * 4`
- Sette en enkeltverdi:
 - `hoydeAar[0] = 5`

0	5
1	0
2	0
3	0

Liste

- Definere en liste:
 - `hoydeAar = [50, 76, 87, 96]`
 - `hoydeAar = []`
 - `hoydeAar = [0] * 4`
- Sette en enkeltverdi:
 - `hoydeAar[0] = 5`
 - `hoydeAar[2] = 8`

0	5
1	0
2	8
3	0

Liste

- Definere en liste:
 - `hoydeAar = [50, 76, 87, 96]`
 - `hoydeAar = []`
 - `hoydeAar = [0] * 4`
- Sette en enkeltverdi:
 - `hoydeAar[0] = 5`
 - `hoydeAar[2] = 8`
- Bruke enkeltverdi
 - `print(hoydeAar[2])`

0	5
1	0
2	8
3	0

Liste

- Definere en liste:
 - `hoydeAar = [50, 76, 87, 96]`
 - `hoydeAar = []`
 - `hoydeAar = [0] * 4`
- Sette en enkeltverdi:
 - `hoydeAar[0] = 5`
 - `hoydeAar[2] = 8`
- Bruke enkeltverdi
 - `print(hoydeAar[2])`

0	5
1	0
2	8
3	0

Utvide en liste

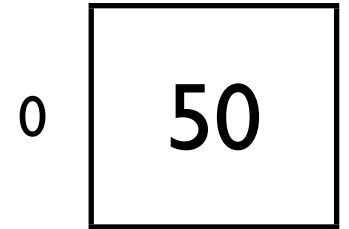
Utvide en liste



- Først definere en tom liste
 - `hoydeAar = []`

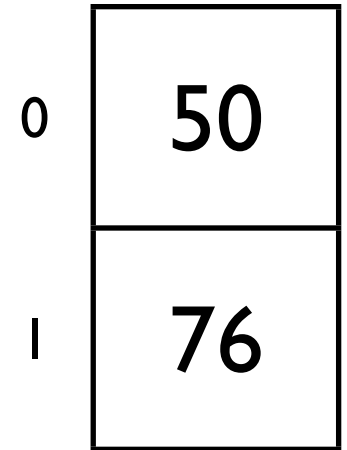
Utvide en liste

- Først definere en tom liste
 - `hoydeAar = []`
- Utvide med en enkeltverdi:
 - `hoydeAar.append(50)`



Utvide en liste

- Først definere en tom liste
 - `hoydeAar = []`
- Utvide med en enkeltverdi:
 - `hoydeAar.append(50)`
 - `hoydeAar.append(76)`



Utvide en liste

- Først definere en tom liste
 - `hoydeAar = []`
- Utvide med en enkeltverdi:
 - `hoydeAar.append(50)`
 - `hoydeAar.append(76)`
- Konkatenerere lister
 - `print([50,76] + [87,96])`

0	50
1	76
2	87
3	96

Utvide en liste

- Først definere en tom liste
 - `hoydeAar = []`
- Utvide med en enkeltverdi:
 - `hoydeAar.append(50)`
 - `hoydeAar.append(76)`
- Konkatenerere lister
 - `print([50,76] + [87,96])`
 - `hoydeAar = hoydeAar + [87,96]`

0	50
1	76
2	87
3	96

Liste - funksjonalitet

- Kan inneholde alle typer verdier
 - `min_liste = [1.5, 2.9, 1.0]`
 - `min_liste = ["Oslo", "Bergen"]`
- Lengde av liste:

Liste - funksjonalitet

- Kan inneholde alle typer verdier
 - `min_liste = [1.5, 2.9, 1.0]`
 - `min_liste = ["Oslo", "Bergen"]`
- Lengde av liste:
 - `len(min_liste)` **2**

Liste - funksjonalitet

- Kan inneholde alle typer verdier
 - `min_liste = [1.5, 2.9, 1.0]`
 - `min_liste = ["Oslo", "Bergen"]`
- Lengde av liste:
 - `len(min_liste)` **2**
- Sjekke om en verdi finnes:

Liste - funksjonalitet

- Kan inneholde alle typer verdier
 - `min_liste = [1.5, 2.9, 1.0]`
 - `min_liste = ["Oslo", "Bergen"]`
- Lengde av liste:
 - `len(min_liste)` **2**
- Sjekke om en verdi finnes:
 - `"Bergen" in min_liste` **True**

Liste - funksjonalitet

- Kan inneholde alle typer verdier
 - `min_liste = [1.5, 2.9, 1.0]`
 - `min_liste = ["Oslo", "Bergen"]`
- Lengde av liste:
 - `len(min_liste)` **2**
- Sjekke om en verdi finnes:
 - `"Bergen" in min_liste` **True**
 - `"Trondheim" in min_liste` **False**

Noen tjenester som lister tilbyr

- Telle

- `liste = [1945, 1814, 1905, 1945]`
`print(liste.count(1945))` # 2

- Sortere

- `liste.sort()`
`print(liste)` # [1814, 1905, 1945, 1945]

En streng er en spesiell type liste

- `tekst = "kamel"`
- `sorted(tekst) # "aeklm"`
- `tekst.count("m") # 1`
- `tekst.append("a") #streng er immutable - kan ikke endres`
- `print(list(tekst)) # ["k","a","m","e","l"] (vanlig liste av bokstaver)`

Hva skrives ut?

```
vest = ["Hallo", "Bergen"]  
midt = ["Trondheim"]  
print( vest + midt) ['Hallo', 'Bergen', 'Trondheim']
```

```
nord = ["Alta", "Kautokeino"]  
vest = nord + vest  
print(vest) ['Alta', 'Kautokeino', 'Hallo', 'Bergen']
```

```
nord.append("Narvik")  
print(nord) ['Alta', 'Kautokeino', 'Narvik']
```

```
lengde = len(vest+nord)  
print(lengde) 7
```

En liten oppgave

A. Lag en liste med fem terningkast (tall fra 1 til 6) som du leser inn fra tastaturet (input)

- Prøv selv med blyant og papir!
- Etterpå diskuter med nabo

En liten oppgave

- A. Lag en liste med fem terningkast (tall fra 1 til 6) som du leser inn fra tastaturet (input)
 - B. Brukeren spiller yatsy og vil bruke sitt kast som firere - hvor mange poeng får brukeren (hun får fire poeng for hver firer hun har)
- Prøv selv med blyant og papir!
 - Etterpå diskuter med nabo

Løsning på A

(lage liste med fem terningkast)

- {firere1.py}

Løsning på B

(telle poeng for firere)

- {firere2.py}

Plan for dagen

- Objekter og tjenester
- Lister
- Mengder
- Ordbøker
- Nøstede samlinger

Mengder

- Fagene man tar et semester
 - Har ingen spesifikk rekkefølge (hva er fag 1, 2 og 3..)
 - Alle fag må være ulike (kan ikke ta IN1000 og IN1000)
- Kan representeres med en liste, men:
 - Får uansett en rekkefølge (som dog kan ignoreres):
["IN1000","IN1020","IN1050"]
 - Ingenting hindrer å ha samme fag to ganger:
["IN1000","IN1000","IN1000"]

Mengder

- Mengde:
 - En samling av ulike verdier
 - Det vil si: a) uten ordning, b) kun ulike verdier
- Mengde i Python:
 - `min_mengde = set([1,5,1,1])`

Mengder

- Mengde:
 - En samling av ulike verdier
 - Det vil si: a) uten ordning, b) kun ulike verdier
- Mengde i Python:
 - `min_mengde = set([1,5,1,1])`
 - `len(min_mengde)` **2**

Mengder

- Mengde:
 - En samling av ulike verdier
 - Det vil si: a) uten ordning, b) kun ulike verdier
- Mengde i Python:
 - `min_mengde = set([1,5,1,1])`
 - `len(min_mengde)` **2**
 - `print(min_mengde)` **set([1, 5])**

Mengder

- Mengde:
 - En samling av ulike verdier
 - Det vil si: a) uten ordning, b) kun ulike verdier
- Mengde i Python:
 - `min_mengde = set([1,5,1,1])`
 - `len(min_mengde)` **2**
 - `print(min_mengde)` **set([1, 5])**
 - `5 in min_mengde` **True**

Mengder

- Mengde:
 - En samling av ulike verdier
 - Det vil si: a) uten ordning, b) kun ulike verdier
- Mengde i Python:
 - `min_mengde = set([1,5,1,1])`
 - `len(min_mengde)` **2**
 - `print(min_mengde)` **set([1, 5])**
 - `5 in min_mengde` **True**
 - ~~`min_mengde[1]`~~

En liten oppgave

- Oppgave:
 - Gitt en liste med fem terningkast (fem tall fra 1 til 6), fortell brukeren om hun har fått yatsy (holder å skrive *True/False*)

*(skriv kun kode for å finne ut om yatsy
- anta at du allerede har en variabel med liste av terningkast, f.eks:
terninger = [3,5,3,3,3]*
 - Prøv selv med blyant og papir! (3 minutt)
*(NB! Bruk tiden på å tenke, ikke skrive
- løsningen kan være veldig kort)*

Løsning

- {yatzy.py}

En liten nøtt

- Oppgave:
 - Gitt en liste med fem terningkast (fem tall fra 1 til 6), fortell brukeren om hun har fått hus (tre av ett terningkast, og to av et annet)
 - Løs det gjerne ved å kombinere teknikkene fra firere og yatzy (count og mengder)
- Løs gjerne først følgende oppgave:
 - Brukeren mangler kun fire like og hus. Fortell brukeren om hun har fått noe hun kan bruke (*uten å skille mellom hvilken av de to*)
 - *Hint: Hvordan kan det ha seg at denne oppgaven er enklere!?*
- *Prøv selv med blyant og papir!*
- *Etterpå diskuter med nabo!*

Løsning på hus eller fire like

- {hus_eller_4like.py}

Løsning på hus

- {hus.py}

Plan for dagen

- Objekter og tjenester
- Lister
- Mengder
- Ordbøker
- Nøstede samlinger

Ordbøker

(dictionaries)

- Tilbake til listen hoydeAar: [50, 76, 87, 96]
 - En samling av fire ulike høyder
 - En slags ordbok fra alder til høyde:
0->50, 1->76, 2->87, 3->96
- Hva om man vil legge til én ekstra sammenheng?
 - 18->179
 - Hva gjør man med indeksene mellom 3 og 18?
 - [50, 76, 87, 96, 0, 0, 0,0,0,0,0,0,0,0,0,0, 0, 0, 179] ??
- Man bruker heller en ordbok (dict)
 - {0:50, 1:76, 2:87, 3:96, 18:179}

Ordbøker

- En ordbok (dict) er en samling mappinger (transformasjoner) fra én verdi til én annen
 - Det man mapper fra kalles nøkkelverdi (key)
 - Det man mapper til kalles en innholdsverdi (value)
- Både nøkler og innholdsverdier kan være av ulike typer
 - `by = {"Norge": "Oslo", "Tyskland": "Berlin", "Italia": "Roma"}`
 - `tlf = {"Norge": 47, "Tyskland": 49, "Italia": 39}`
- Slår opp som i en liste:
 - `by["Norge"]` **Oslo**

Ordbok - funksjonalitet

- Lage en liste
 - `min_ordbok = {}`
 - `min_ordbok = {"Norge": 47, "Tyskland": 49}`
- Legge til ekstra element
 - `min_ordbok["Italia"] = 39`
- Slå opp i ordlisten

Ordbok - funksjonalitet

- Lage en liste
 - `min_ordbok = {}`
 - `min_ordbok = {"Norge": 47, "Tyskland": 49}`
- Legge til ekstra element
 - `min_ordbok["Italia"] = 39`
- Slå opp i ordlisten
 - `print(min_ordbok["Tyskland"] - min_ordbok["Norge"])` **2**

Ordbok - funksjonalitet

- Lage en liste
 - `min_ordbok = {}`
 - `min_ordbok = {"Norge": 47, "Tyskland": 49}`
- Legge til ekstra element
 - `min_ordbok["Italia"] = 39`
- Slå opp i ordlisten
 - `print(min_ordbok["Tyskland"] - min_ordbok["Norge"])` **2**
- Sjekke hvilke nøkkelverdier den inneholder

Ordbok - funksjonalitet

- Lage en liste
 - `min_ordbok = {}`
 - `min_ordbok = {"Norge": 47, "Tyskland": 49}`
- Legge til ekstra element
 - `min_ordbok["Italia"] = 39`
- Slå opp i ordlisten
 - `print(min_ordbok["Tyskland"] - min_ordbok["Norge"])` **2**
- Sjekke hvilke nøkkelverdier den inneholder
 - `"Norge" in min_ordbok` **True**

Ordbok - funksjonalitet

- Lage en liste
 - `min_ordbok = {}`
 - `min_ordbok = {"Norge": 47, "Tyskland": 49}`
- Legge til ekstra element
 - `min_ordbok["Italia"] = 39`
- Slå opp i ordlisten
 - `print(min_ordbok["Tyskland"] - min_ordbok["Norge"])` **2**
- Sjekke hvilke nøkkelverdier den inneholder
 - `"Norge" in min_ordbok` **True**
 - `47 in min_ordbok` **False** (*sjekker kun nøklene!*)

Oppgave: Skandinavia-orakel

- Endre hovedstads-programmet fra uke1 til å basere seg på en ordbok i stedet for if:
 - ```
land = input("Velg land i nordre Skandinavia: ")
if land == "Norge":
 print("Oslo")
if land == "Sverige":
 print("Stockholm")
```
- *Prøv selv med blyant og papir! (3 minutt)*

# Løsning

- {skandinavia.py}

# En liten nøtt

```
person = input("Konge: ")
etterkommere = {"Oscar":"Haakon", "Haakon":"Olav",
"Olav:Harald"}
```

```
#kode som setter variabel barnebarn
#med riktig verdi her....
```

```
print("Barnebarn: " + barnebarn)
```

# Løsning

- {barnebarn.py}



# Plan for dagen

- Objekter og tjenester
- Lister
- Mengder
- Ordbøker
- Nøstede samlinger

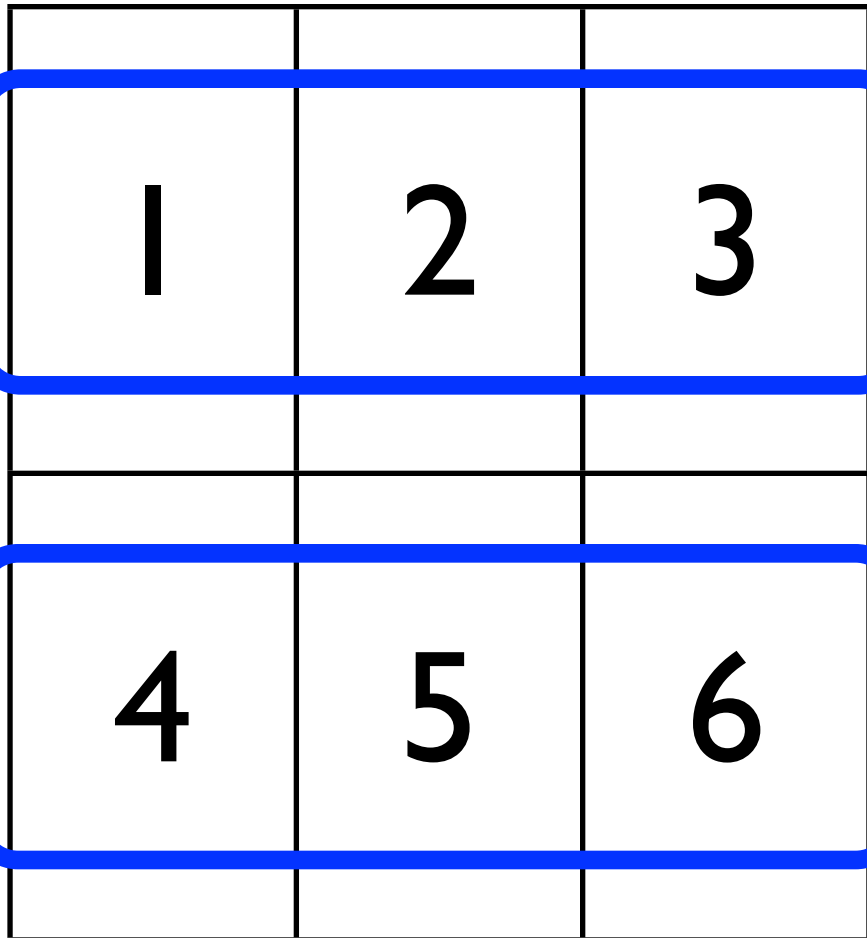
# Nøstede lister

- En liste er en samling av verdier
  - Hver verdi kan igjen være en liste
- Man får da en liste av lister
  - frokoster = [ ["egg", "bacon"], ["ost", "agurk", "tomat"] ]
  - Merk doble hakeparanteser og komma mellom lister
- Rekkefølge/indeks kan være av betydning

*En spesifikk type nøstet liste:*  
Matrise (tabell)

|   |   |   |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |

# *En spesifikk type nøstet liste:* Matrise (tabell)



|   |   |   |
|---|---|---|
|   |   |   |
| 1 | 2 | 3 |
|   |   |   |
| 4 | 5 | 6 |
|   |   |   |

- En liste av rader
- Hver rad er en liste

# *En spesifikk type nøstet liste:* Matrise (tabell)

|   |   |   |
|---|---|---|
|   |   |   |
| 1 | 2 | 3 |
| 4 | 5 | 6 |
|   |   |   |

- En liste av rader
  - Hver rad er en liste
- `t = [ [1,2,3], [4,5,6] ]`  
`assert t[1][2]==6`

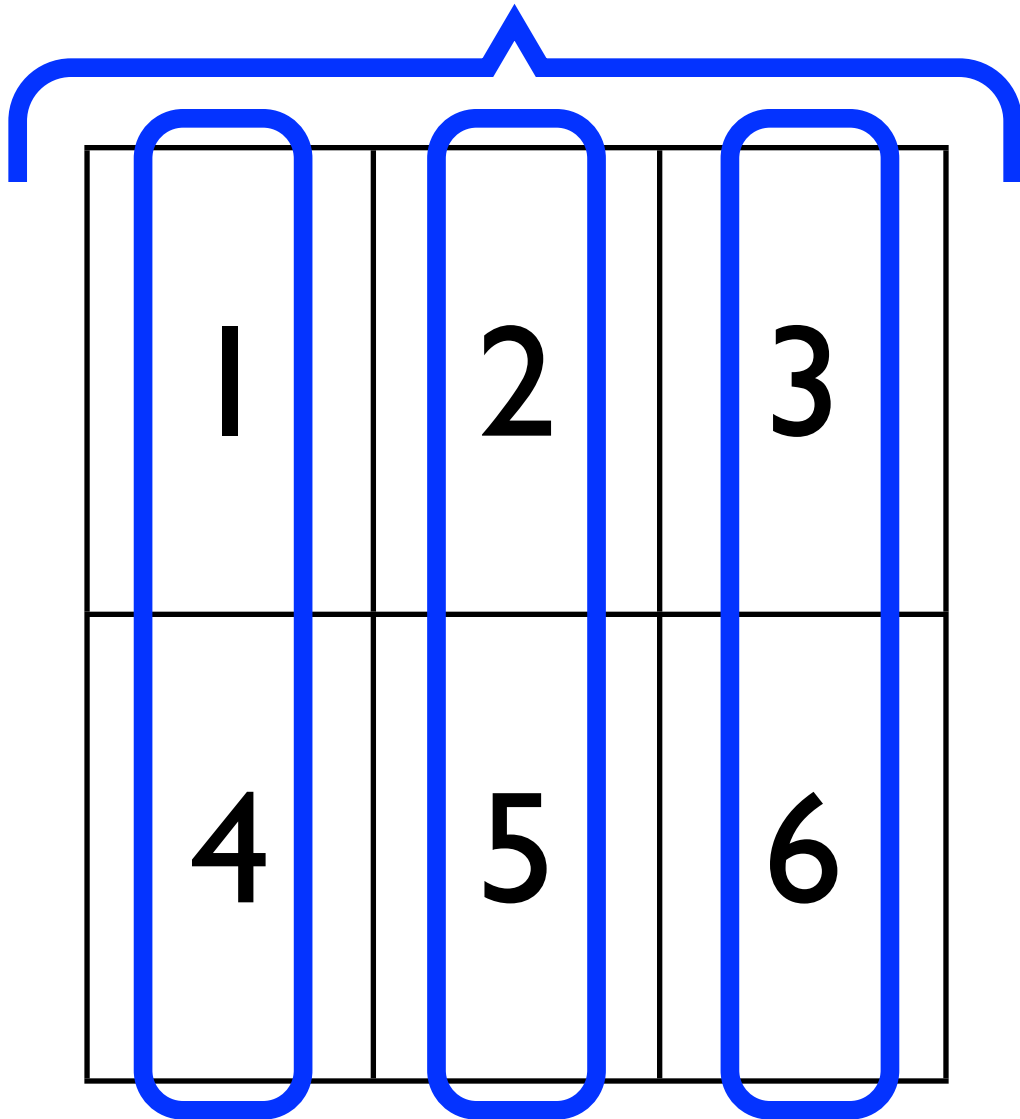
# *En spesifikk type nøstet liste:* Matrise (tabell)

|                |                |                |
|----------------|----------------|----------------|
| $t[0][0]$<br>1 | $t[0][1]$<br>2 | $t[0][2]$<br>3 |
| $t[1][0]$<br>4 | $t[1][1]$<br>5 | $t[1][2]$<br>6 |

- En liste av rader
- Hver rad er en liste
- `t = [ [1,2,3], [4,5,6] ]`

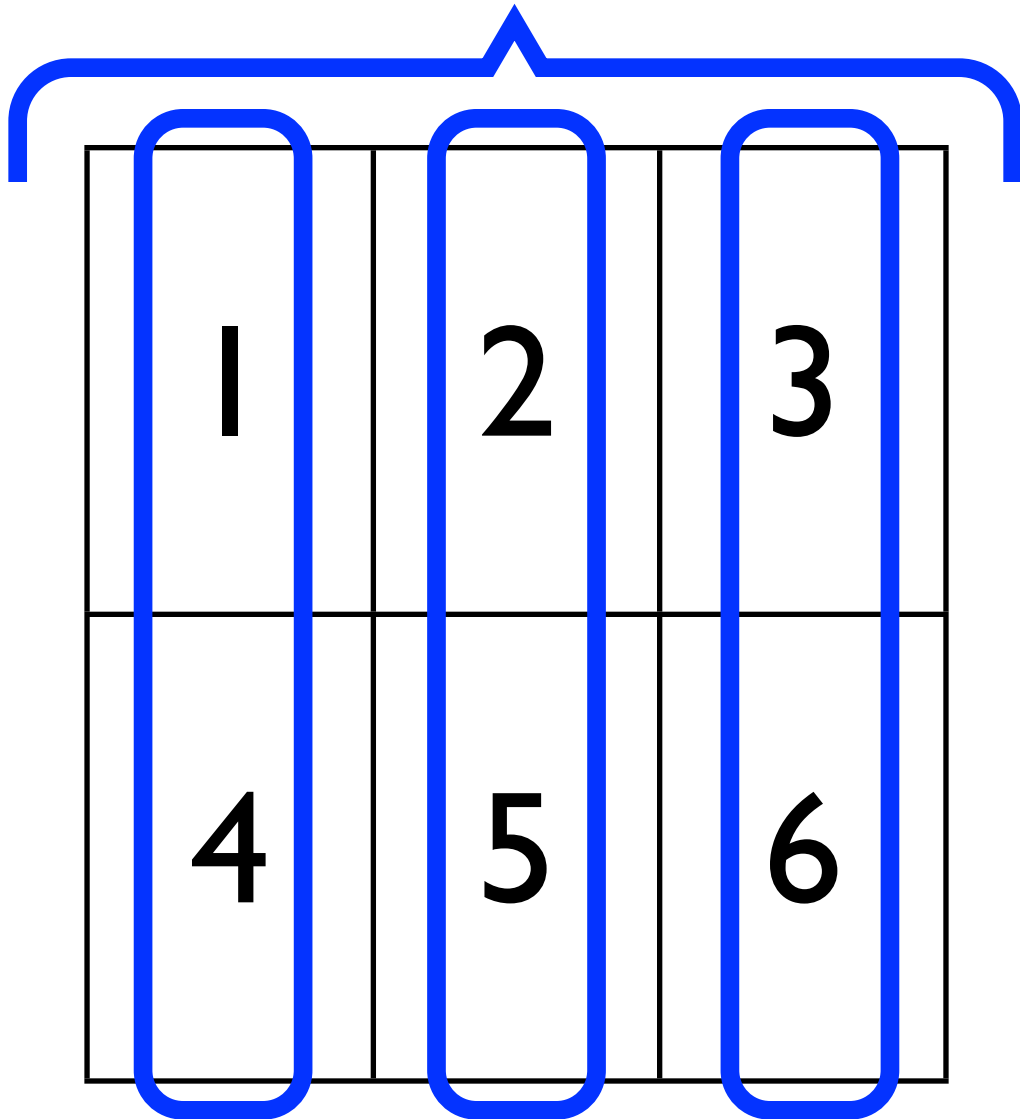
`assert t[1][2]==6`

# *En spesifikk type nøstet liste:* Matrise (tabell)



- En liste av kolonner
  - Hver kolonne er en liste

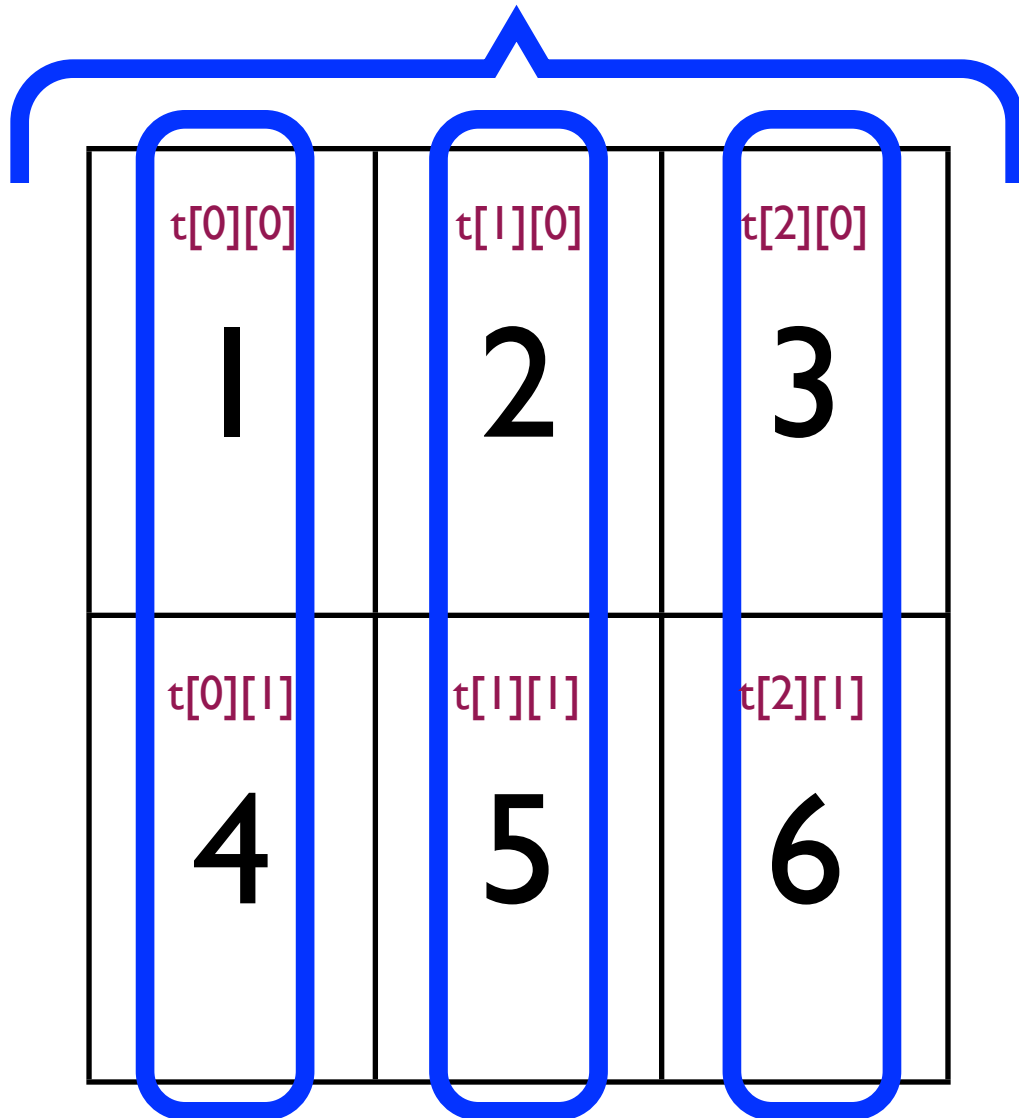
# *En spesifikk type nøstet liste:* Matrise (tabell)



- En liste av kolonner
  - Hver kolonne er en liste
- `t = [ [1,4], [2,5], [3,6] ]`  
`assert t[2][1]==6`



# *En spesifikk type nøstet liste:* Matrise (tabell)



- En liste av kolonner
  - Hver kolonne er en liste
- $t = [ [1,4], [2,5], [3,6] ]$   
assert  $t[2][1]==6$

# Nøstede samlinger

- Man kan nøste i flere enn 2 nivåer
  - `liste = [ [ [1,2], [3,4,5] ], [ [11,12], [13,14,15] ] ]`
  - `assert liste[1][0][1] == 12`
- Man kan nøste andre typer samlinger:
  - `landinfo = { "byer": {"Norge": "Oslo", "Tyskland": "Berlin"}, "tlf": {"Norge": 47, "Tyskland": 49} }`
  - `assert landinfo["tlf"]["Norge"] == 47`
- Man kan blande ulike typer samlinger:
  - `frokoster = { "tung": ["egg", "bacon"], "lett": ["ost", "agurk", "tomat"] }`
  - `assert frokoster["lett"][1] == "agurk"`

# Oppgave

- Gitt en 2x2-tabell som nøstet liste, finn ut om all radsummer og kolonnesummer er like.

|   |   |   |
|---|---|---|
| 5 | 3 | 8 |
| 3 | 5 | 8 |
| 8 | 8 |   |

[ [5,3], [3,5] ]

- ```
tab = [ [5,3], [3,5] ]  
#Skriv kode her som setter variabelen alle_like  
print(alle_like)
```
- *Prøv selv med blyant og papir!*

Løsning

- {tabell.py}

Oppsummering:

Egenskaper ved de tre typene samlinger

- Liste (list)
 - En samling verdier i bestemt rekkefølge med **indeks** fra 0 og oppover
 - Typisk: hva finnes på en bestemt lokasjon i lista?
- Mengde (set)
 - En samling **ulike** verdier uten rekkefølge
 - Typisk: finnes en bestemt verdi i mengden?
- Ordbok (dict)
 - En samling av koblinger fra én verdi (nøkkelverdi) til en annen
 - Typisk: hvilken verdi er koblet til en bestemt nøkkel?

Lage de nye typene objekter

- Lage en tom samling
 - Liste: []
 - Mengde: set()
 - Ordbok: {}
- Legge til i samlingen:
 - `min_liste.append(ny_verdi)`
 - `min_mengde.add(ny_verdi)`
 - `min_ordbok[ny_nokkel] = ny_koblet_verdi`

Tjenester for de nye typene objekter

- Lister, mengder og ordbøker tilbyr litt ulike tjenester
- Eksempel: om noe finnes i samlingen
 - Liste: om noe finnes eller hvor mange ganger
 - Mengde: om noe finnes (aldri mer enn en gang..)
 - Ordbok: om noe finnes som nøkkelverdier

Begrensede måter å bruke liste på

- Array
 - En klassisk type liste som er begrenset til å ha en fast størrelse, og elementer av en bestemt type
 - Dette kan tillate effektiv representasjon i en datamaskin
- Stakk (først-inn-sist-ut)
 - En type samling hvor man kun kan legge til og fjerne elementer fra slutten av samlingen (append og pop)
- Kø (først-inn-først-ut)
 - En type samling hvor man kun kan legge til elementer i slutten og kun fjerne fra starten av samlingen

Oppsummering

- Lister gjør det mulig å jobbe med mange verdier
 - Kan slå opp direkte på verdien i en bestemt posisjon
- Man har flere typer samlinger til disposisjon:
 - Lister, mengder, ordbøker

Hva som er viktig fra dagens forelesing

- At ulike verdier (tekster, lister osv) er objekter som tilbyr tjenester
 - Men ikke viktig å huske akkurat hvilke tjenester de ulike typene objekter tilbyr
- Hva lister, mengder og ordbøker kan brukes til, samt hva som overordnet skiller de tre
 - Men den eksakte syntaksen er ikke så kritisk for nå