Pythontutor lenke: Gruppe-Person-Navn

https://pythontutor.com/visualize.html#code=class%20Navn%3A%0A%20%20%20%20def%20__init__%28self,%20fornavn,%20mellomnavn,%20etternavn%29%3A%0A%20%20%20%20%20%20%20%20self._for%20%3D%20fornavn%0A%20%20%20%20%20%20%20%20self._mellom%20%3D%20mellomnavn%0A%20%20%20%20%20%20%20%20self._etter%20%3D%20etternavn%20%20%20%20%20%20%20%20%0A%20%20%20%20def%20naturlig%28self%29%3A%0A%20%20%20%20%20%20%20%20return%20%28self._for%20%2B%20%22%20%22%20%2B%20self._mellom%20%2B%20%22%20%22%20%2B%20self._etter%29%20%20%20%20%20%20%20%20%0A%20%20%20%20def%20sorterbar%28self%29%3A%0A%20%20%20%20%20%20%20%20return%20self._etter%2B%22,%20%22%2Bself._for%2B%22%20%22%2Bself._mellom%20%20%20%0A%20%20%20%20def%20__str__%28self%29%3A%0A%20%20%20%20%20%20%20%20return%20self.naturlig%28%29%0A%0Aclass%20Person%3A%0A%20%20%20%20def%20__init__%28self,fullt_navn,alder%29%3A%0A%20%20%20%20%20%20%20%20self._navn%20%3D%20fullt_navn%0A%20%20%20%20%20%20%20%20self._alder%20%3D%20alder%0A%20%20%20%20def%20skrivUt%28self%29%3A%0A%20%20%20%20%20%20%20%20print%20%28%22Navn%3A%20%22%20%2B%20self._navn.naturlig%28%29%20%29%0A%20%20%20%20%20%20%20%20print%20%28%22Alder%3A%20%22%20%2B%20str%28self._alder%29%29%0A%0Aclass%20Gruppe%3A%0A%20%20%20%20def%20__init__%28self,%20nr%29%3A%0A%20%20%20%20%20%20%20%20self._gruppe_nr%20%3D%20nr%0A%20%20%20%20%20%20%20%20self._medlemmer%20%3D%20%5B%5D%0A%20%20%20%20def%20nytt_medlem%28self,%20ny%29%3A%0A%20%20%20%20%20%20%20%20self._medlemmer.append%28ny%29%0A%20%20%20%20def%20hent_alle_medlemmer%28self%29%3A%0A%20%20%20%20%20%20%20%20return%20self._medlemmer%0A%20%0Adef%20hovedprogram%28%29%3A%0A%20%20%20%20gruppe_1%20%3D%20Gruppe%281%29%0A%20%20%20%20ny_person%20%3D%20Person%28Navn%28%22Gro%22,%20%22H.%22,%20%22Brundtland%22%29,%2050%29%0A%20%20%20%20gruppe_1.nytt_medlem%28ny_person%29%0A%20%20%20%20ny_person%20%3D%20Person%28Navn%28%22Jonas%22,%20%22G.%22,%20%22St%C3%B8re%22%29,%2040%29%0A%20%20%20%20gruppe_1.nytt_medlem%28ny_person%29%0A%20%20%20%20medlemmer%20%3D%20gruppe_1.hent_alle_medlemmer%28%29%0A%20%20%20%20for%20medlem%20in%20medlemmer%3A%0A%20%20%20%20%20%20%20%20print%28medlem%29%0A%0Ahovedprogram%28%29&curInstr=48&mode=display&origin=opt-frontend.js&py=3

Objektorientert programmering i Python

IN1000

Høst 2026 – uke 10

Siri Moe Jensen og Geir Kjetil Sandve

Semesteret videre

Uke	Forelesninger	Gruppetimer	Større egne oppgaver
10	Sentrale datastrukturer for programmering	Fra ukens forelesning, oblig 8-trening	Oblig 8
11	Live-koding av et større, objektorientert program	Større programmer Repetisjon Eksamensoppgaver	
12	Pensum-gjennomgang og introduksjon til Inspera		Prøveeksamen i Inspera (rettes ikke)
13	Gjennomgang prøveeksamen, eksamenstips		Tidligere eksamensoppgaver
14-15	Ingen forelesninger eller ordinære gruppetimer	Fredagspython Pragmatisk programmering	
16	Eksamen (4.12)		

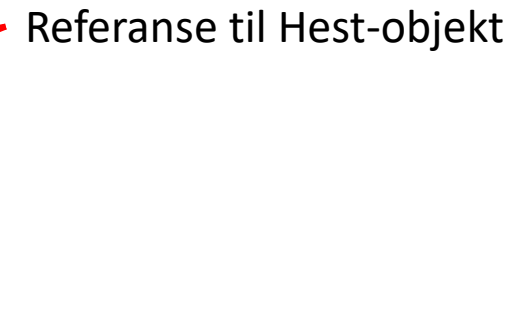
- Fredagspython fortsetter til og med uken før eksamen

Instansvariabler i en klasse

- Kan for eksempel være
  - Enkle (primitive) typer som int, boolean, float
  - Referanser til objekter av innebygde typer som string, list og dict
  - Referanser til objekter av klasser vi skriver selv, som Navn

    … inkludert objekter av klassen selv!

```
class Hest:
    def __init__(self, navn, født):
        self._navn = navn
        self._født = født
        self._avkom = None
```



Et føll blir født!

```
class Hest:
    def __init__(self, navn, født):
        self._navn = navn
        self._født = født
        self._avkom = None

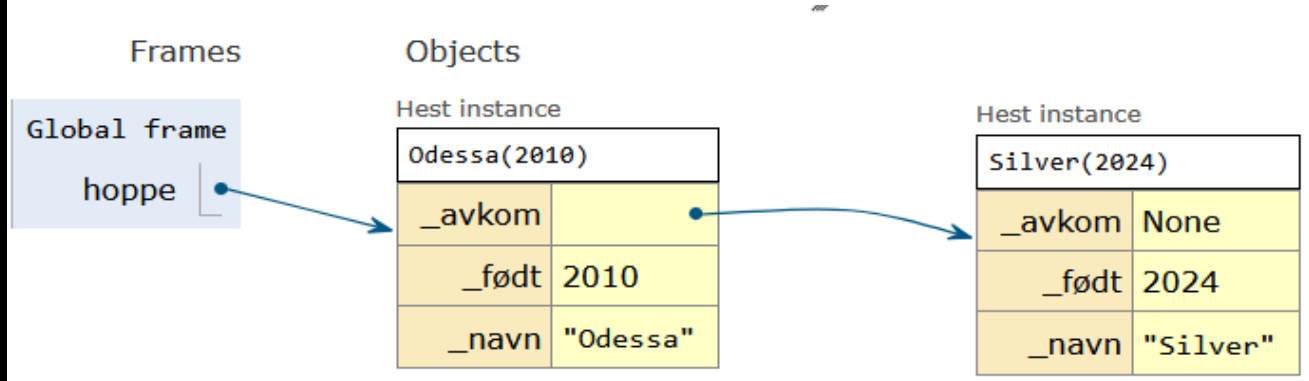
    def fødsel(self, føll):
        self._avkom = føll

    def hent_avkom(self):
        return self._avkom

    def __str__(self):
        return self._navn + "(" + str(self._født) + ")"

hoppe = Hest("Odessa", 2010)
hoppe.fødsel(Hest("Silver", 2024))

avkom = hoppe.hent_avkom()
print(str(hoppe) + " har følget: " + str(avkom))
```



Flere føll?

```
class Hest:
    def __init__(self, navn, født):
        self._navn = navn
        self._født = født
        self._avkom = []

    def fødsel(self, føll):
        self._avkom.append(føll)

    def hent_avkom(self):
        return self._avkom

    def __str__(self):
        return self._navn + "(" + str(self._født) + ")"
```

```
hoppe = Hest("Odessa", 2010)
hoppe.fødsel(Hest("Silver", 2024))
hoppe.fødsel(Hest("Tora", 2026))

avkom = hoppe.hent_avkom()
print(str(hoppe) + " har følgende avkom: ")"
for hest in avkom:
    print (hest)
```

```python
class Hest:
    def __init__(self, navn, født):
        self._navn = navn
        self._født = født
        self._avkom = []

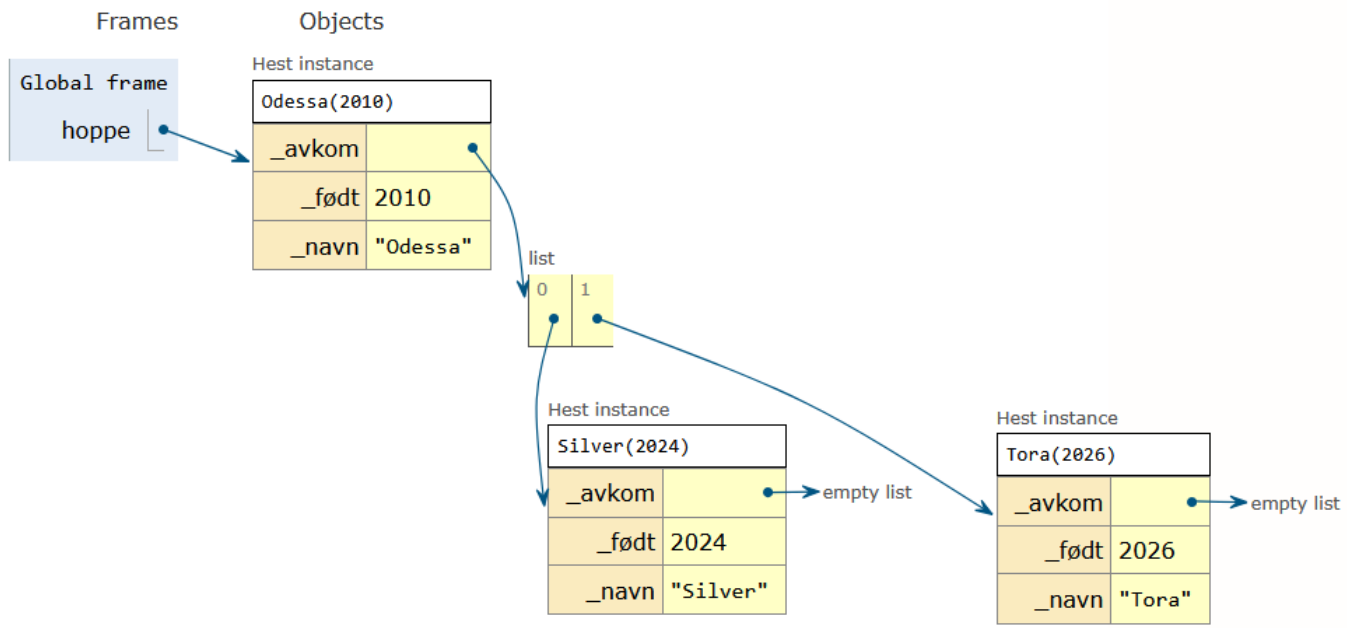
    def fødsel(self, føll):
        self._avkom.append(føll)

    def hent_avkom(self):
        return self._avkom

    def __str__(self):
        return self._navn + "(" + str(self._født) + ")"

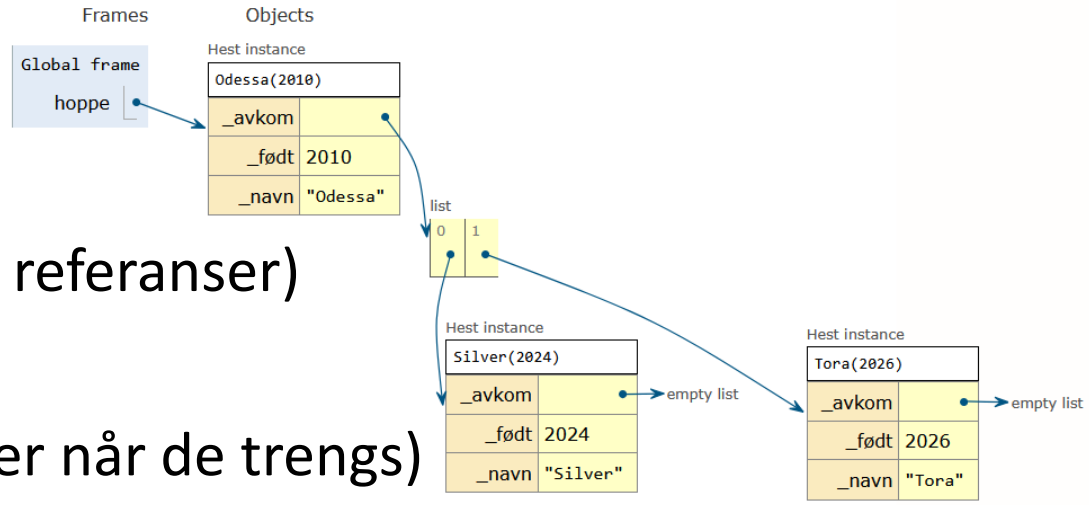
hoppe = Hest("Odessa", 2010)
hoppe.fødsel(Hest("Silver", 2024))
hoppe.fødsel(Hest("Tora", 2026))

avkom = hoppe.hent_avkom()
print(str(hoppe) + " har følgende avkom: ")"
for hest in avkom:
    print (hest)
```



Magien i objekter

- Kan opprette så mange vi vil, når som helst
- Kan skreddersy objekter (med egne klasser)
- Kan koble dem sammen akkurat som vi vil (med referanser)

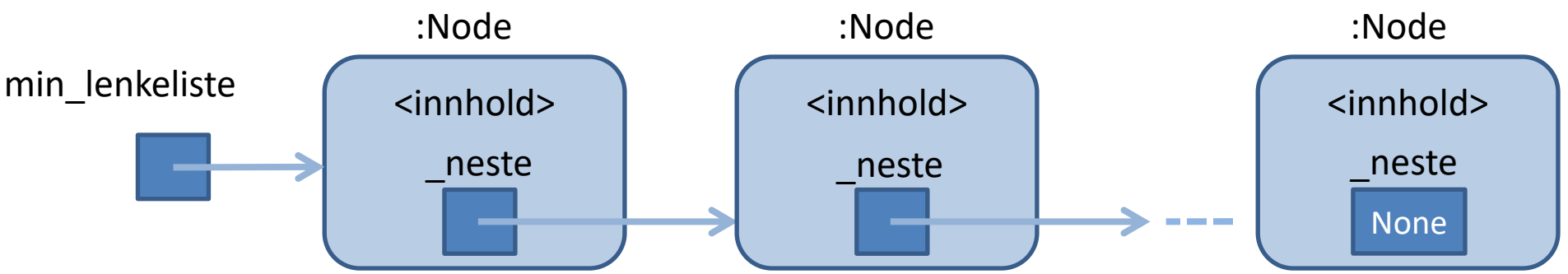


⇒ Lego!! (der vi designer og lager våre egne klosser når de trengs)

- Variablene våre må defineres når vi skriver programmet
- Objektene kan opprettes når som helst
- Men vi trenger en måte å få tak i dem på (referanser)

Lenkede lister

- Hvordan kunne vi lagre et ukjent antall objekter om vi ikke hadde Python lister?
- En lenket liste: Hvert objekt har en referansevariabel som kan referere til et annet, nytt objekt
- Vi får en lenket liste av objekter – og trenger bare én variabel som refererer til det første objektet

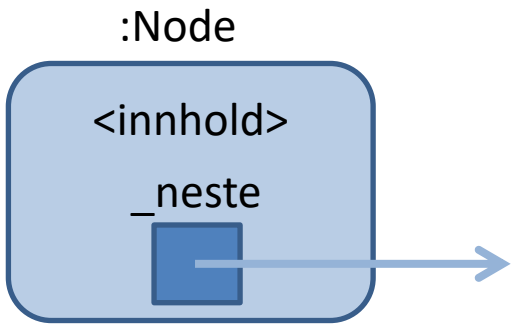


Lenket liste

Vi lager en generell klasse for objektene i lenkelisten

- Klassen Node

- **Grensesnitt** for klassen Node:
  - lese, endre, skrive ut, ... innhold
  - konstruktør med parameter innhold
  - legg til ny etterfølger
  - hent etterfølger
  - (fjern etterfølger)



Klassen Node: Implementasjon

- Datastruktur Node-klassen
  - innholdet objektene skal representere (her: en streng)
  - en referansevariabel så vi får tak i neste objekt

```
class Node :
    def __init__(self, nytt) :
        self._innhold = nytt
        self._neste = None

    def ny_etterfolger (self, ny) :
        self._neste = ny

    def hent_neste (self) :
        return self._neste

     # + metoder for manipulering av innholdet
```

Skriv et program som oppretter en lenket liste med 5 noder, og deretter skriver ut innholdet i alle nodene.

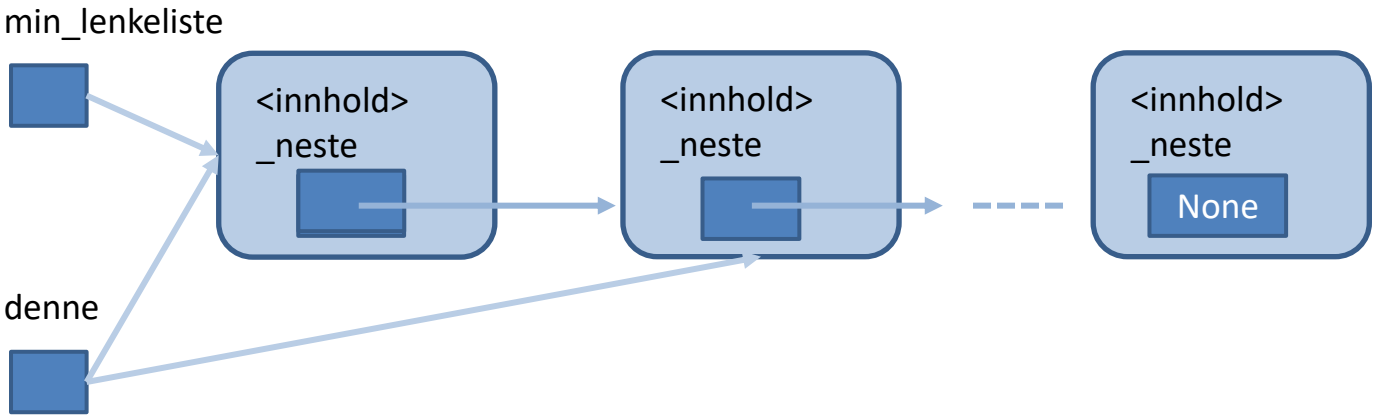
Bygge opp en lenkeliste

```
class Node :
    def __init__(self, nytt) :
        self._innhold = nytt
        self._neste = None

    def ny_etterfolger (self, ny) :
        self._neste = ny

    def hent_neste (self) :
        return self._neste

    def hent_innhold(self):
        return self._innhold
```

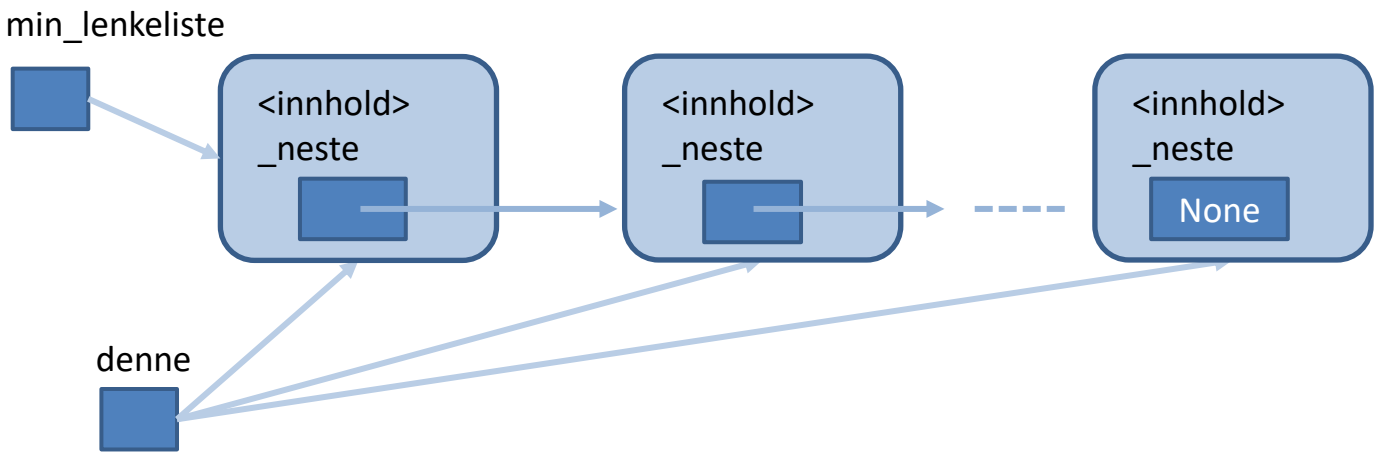



```
min_lenkeliste = Node("1")
denne = min_lenkeliste
for teller in range(2,6):
    denne.ny_etterfolger(Node(str(teller)))
    denne = denne.hent_neste()
```

Flytte seg gjennom nodene i en liste

```
min_lenkeliste = Node("1")
denne = min_liste
for i in range(2,6):
    ny = Node(str(i))
    denne.ny_etterfolger(ny)
    denne = denne.hent_neste()

denne = min_lenkeliste
while denne is not None:
    print(denne.hent_innhold())
    denne = denne.hent_neste()
```



Live eksempel I – trikk

Et eksempel der lenkene (referansene) representerer viktig informasjon om objektene våre (ikke bare hjelper oss å lagre et ukjent antall objekter).

Her er nodene *stasjoner på en trikkerute*, dvs. den lenkede listen består av objekter av klassen Stasjon som inneholder stasjonsnavn og referanse til neste stasjon.

<se egen presentasjon og kildekode>

Live eksempel II – kollektivnettet (graf)

Innlevering 8

1. Denne har tidligere vært obligatorisk – og pensum og krav er de samme som før

2. Samme oppgave blir første obligatoriske oppgave i IN1010 (men i Java)

- Conway's *Game of Life* - se f eks Wikipedia
- Husk:
  - Test for hver metode
  - Hvilken type har variablene du bruker til enhver tid?
  - Grensesnitt og innkapsling:
    - Celle håndterer sine data og operasjoner på disse
    - Rutenett håndterer rutenettet og operasjoner på dette

- Et mindre program med et rutenett gjennomgås på gruppene denne uken (Bondesjakk/ Tre på rad)

Kan tenke oss innholdet i en tabell

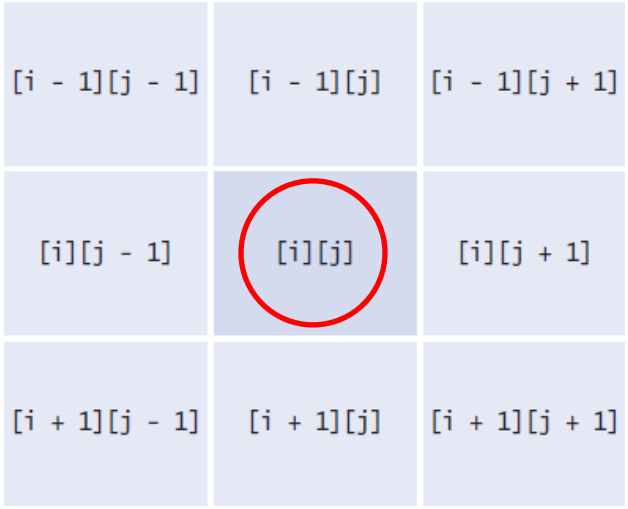
	0	1	2	3	4	5	..	..
0		[0][1]						
1					[1][4]			
2								
3				[3][3]				
4		[4][1]						
5	[5][0]							
..								
..								

- Her er elementene nummerert som om vi leser en bokside (med latinsk alfabet!):
  Vi leser en rad av gangen fra øverst til nederst, tegn for tegn på raden fra venstre mot høyre.

- Dvs første indeks angir rad, andre indeks angir kolonne.

Indeksering

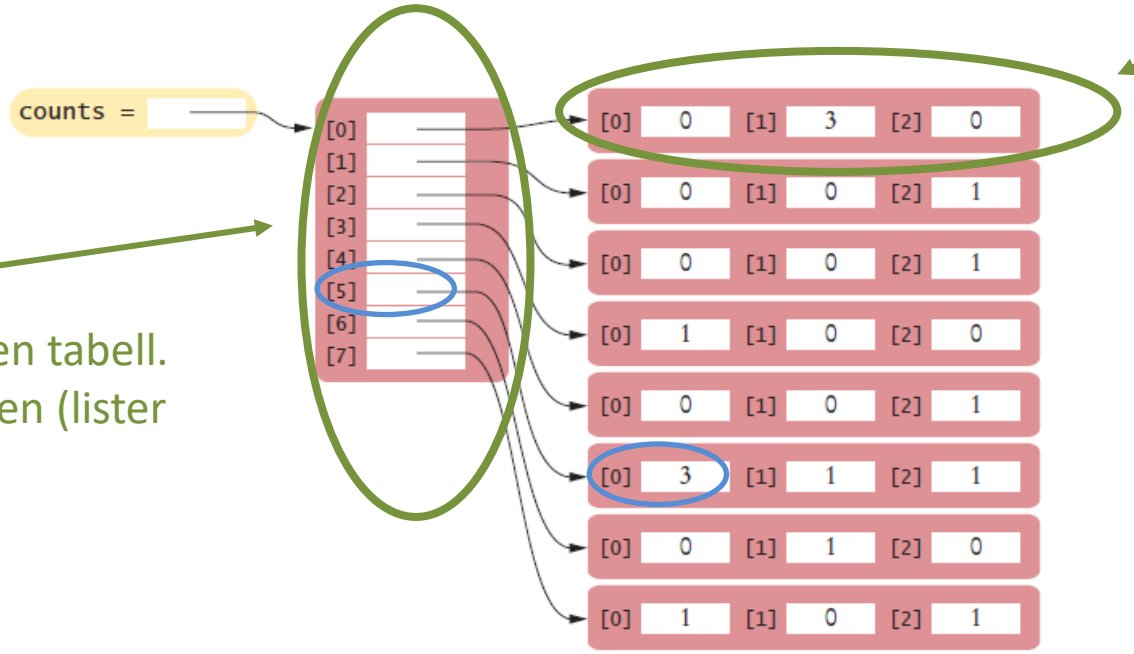
- En tabell med generiske indekser:



Liste av lister for en tabell

Fra læreboken:

- **Her bruker vi en liste av lister til å representere en tabell**
- **Har valgt å la den ytterste liste representere rader**
- **Da blir verdiene i de innerste listene det som er i en gitt kolonne på hver rad**
- innholdet i disse listene hentes ut med f eks counts[5][0]



Denne listen er én rad. Elementene er verdien i hver kolonne for denne raden

Denne listen representerer en tabell. Elementene er rader i tabellen (lister med verdiene i en rad)

Liste av lister

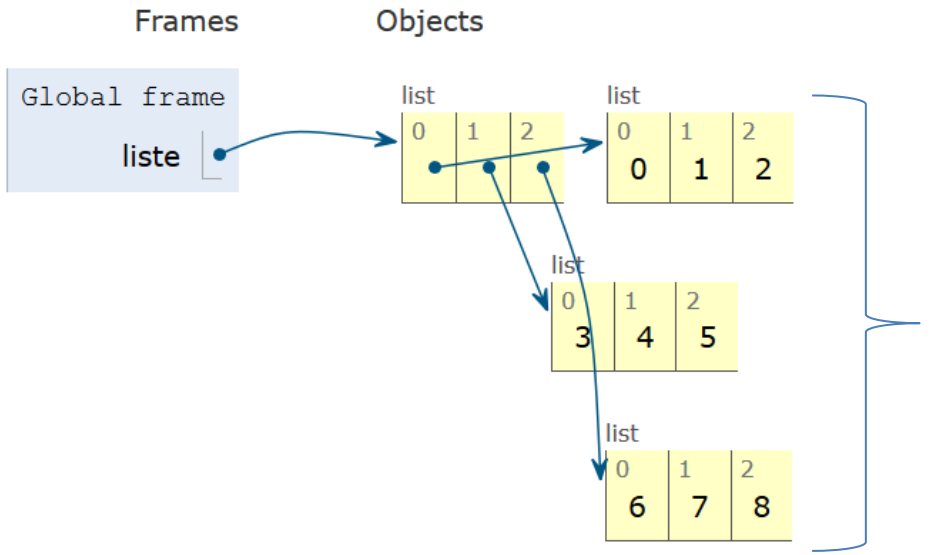
- En liste inneholder referanser til elementer
- Elementene i (den ytre) listen kan være andre (indre) lister, som igjen kan ha lister som elementer, etc

- Python notasjon for å aksessere innholdet i lister:
  - liste[indeks]
  - For nøstede lister: liste[indeks1][indeks2][indeks3]

- Men hvordan *lage* (store) nøstede lister?
  Hvorfor blir dette (lenke til Python tutor) feil?
  Hvordan fikse koden?

liste = [[0,1,2],[3,4,5],[6,7,8]]

Hva skrives ut her?
print(liste[0][1])



fra pythontutor.com