Objektorientert programmering i Python

IN1000

Høst 2023 – uke 8

Siri Moe Jensen

Plan for i dag

Det man ikke husker, er vanskelig å forstå –
og det man ikke forstår, er vanskelig å huske

- Repetisjon forrige uke med flere eksempler, mer detaljer
- Referanser og objekter
- "Uforanderlige" (immutable) klasser

- Organisering av mange objekter
- Visualisering av sammensatte datastrukturer

Neste uke:

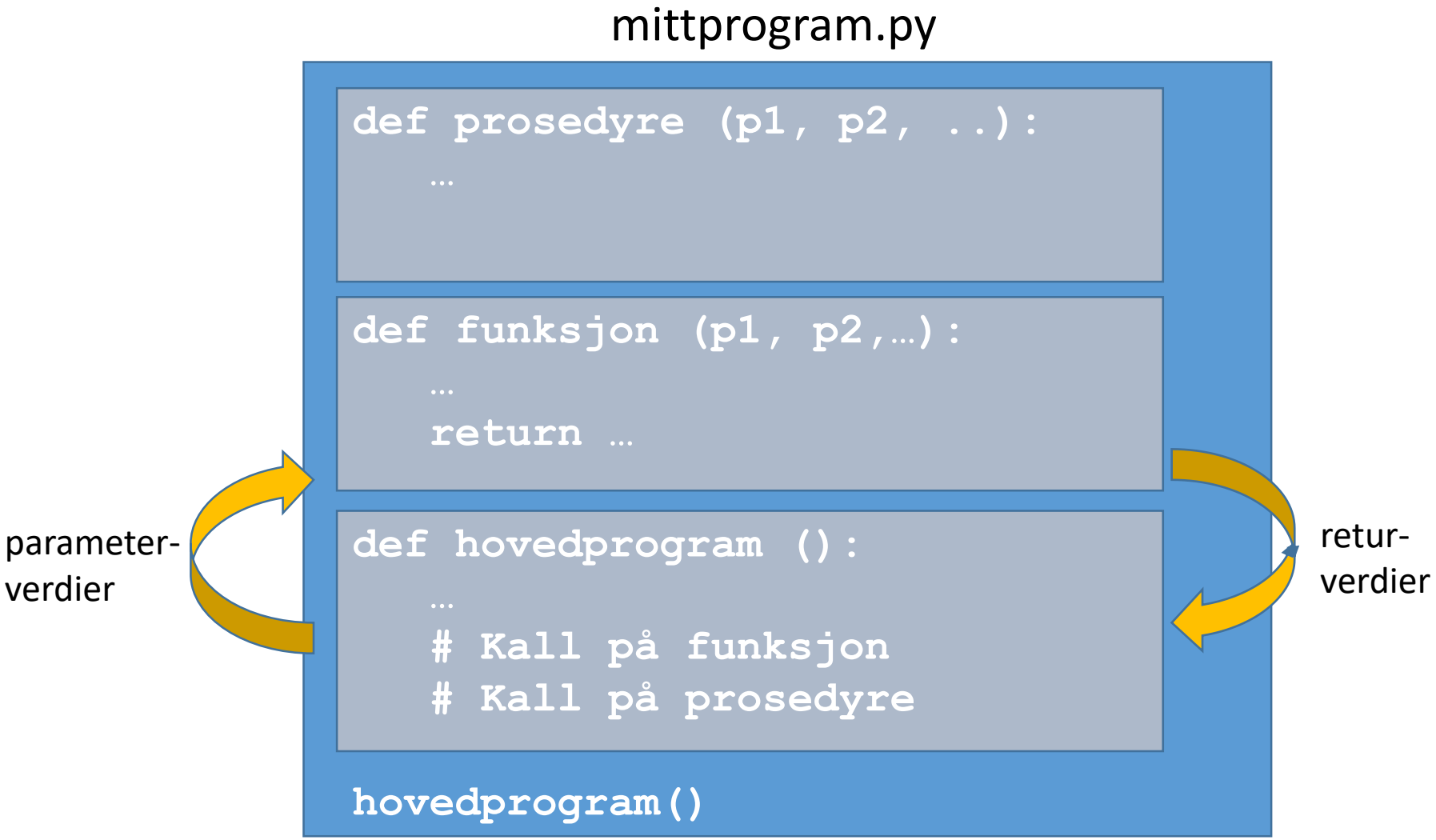
- Mer om datastrukturer med flere klasser og med objekter som refererer til andre objekter
- "Magiske metoder"

Innlevering 7: Obligatorisk

- Frist 23.10.
  Kan få nytt forsøk om det ikke er altfor store mangler ("gjort et reelt forsøk")

- Program med to klasser og flere objekter
- I innlevering 8 og eksamensoppgaven vil du skrive et større objektorientert program
- Obligatorisk oppgave 7 gir mulighet for å jobbe grundig med dette stoffet. Innlevering 6 og Trix-oppgaver gir trening i det grunnleggende.
- Gruppetimer med livekoding og felles oppgaveløsning er nødvendig supplement for de fleste

Program med prosedyrer og funksjoner



Program med klasser

klasse.py

```
class Klasse:
    def __init__(self,p1):
        self._instvar1 = p1
        self._instVar2 = 3.14

    def metode1 (self, p2,..):
        …
```

```
def hovedprogram ():
    …
    # oppretter og bruker objekter
```

```
hovedprogram()
```

Klasser kan være nyttige å gjenbruke i flere programmer

- For at det skal fungere mest mulig feilfritt og forutsigbart, må noen krav være oppfylt:

    1. Klassen må ha et stabilt og veldokumentert **grensesnitt**: Vi må vite hvilke metoder som tilbys, hva de gjør, hvilke argumenter de tar og om de returnerer en verdi
    2. **Innkapsling**: Instansvariabler i en klasse aksesseres kun fra metoder i klassen
    3. Klassedefinisjonen ligger i **en egen fil** (som kan importeres der den trengs). Unngå tester i samme fil, og fjern dem i hvert fall før du leverer inn eller bruker klassen i andre programmer!

- Selv om vi i IN1000 skriver klasser for egen bruk og gjerne i ett program, skal vi trene på å følge disse kravene (men legger mindre vekt på dokumentasjonen i 1.)

Program med klasse i egen programfil

klasse.py

```
class Klasse:
    def __init__(self,p1):
        self._instvar1 = p1
        self._instVar2 = 3.14

    def metode1 (self, p2,..):
        …
```

program.py

```
from klasse import Klasse
def hovedprogram ():
    …
    # oppretter og bruker objekter
hovedprogram()
```

Klasser skrives ofte slik at de er nyttige i mange ulike programmer og kan gjenbrukes.

Eks: strenger og lister

Visualisering

- Tegningene av hvordan vi kan strukturere programmene våre ved hjelp av funksjoner og klasser, i en eller flere filer, visualiserer kildekoden (programmet) vårt.

klasse.py

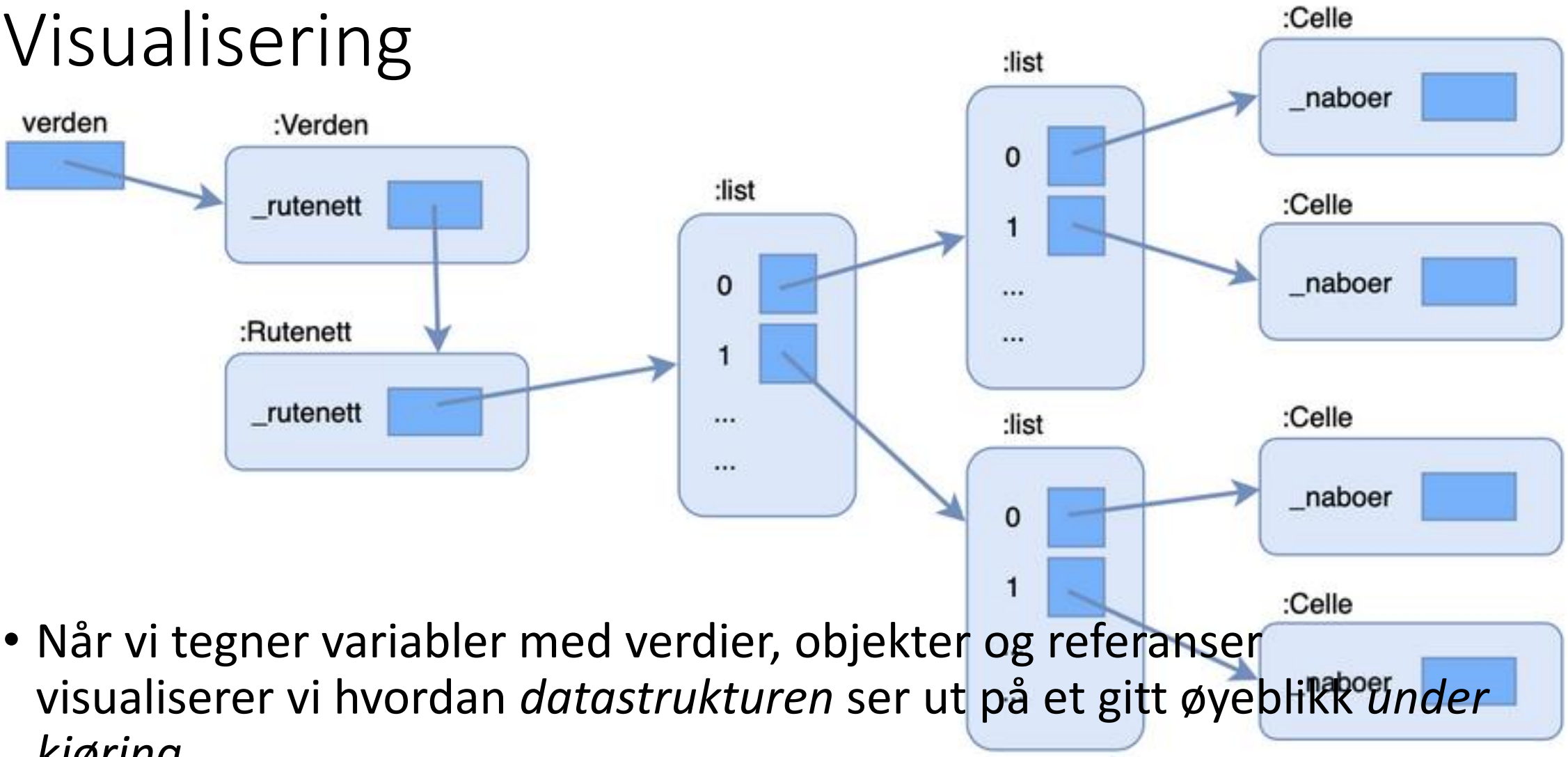
```
class Klasse:
    def __init__(self,p1):
        self._instvar1 = p1
        self._instVar2 = 3.14

    def metode1 (self, p2,..):
        …
```

program.py

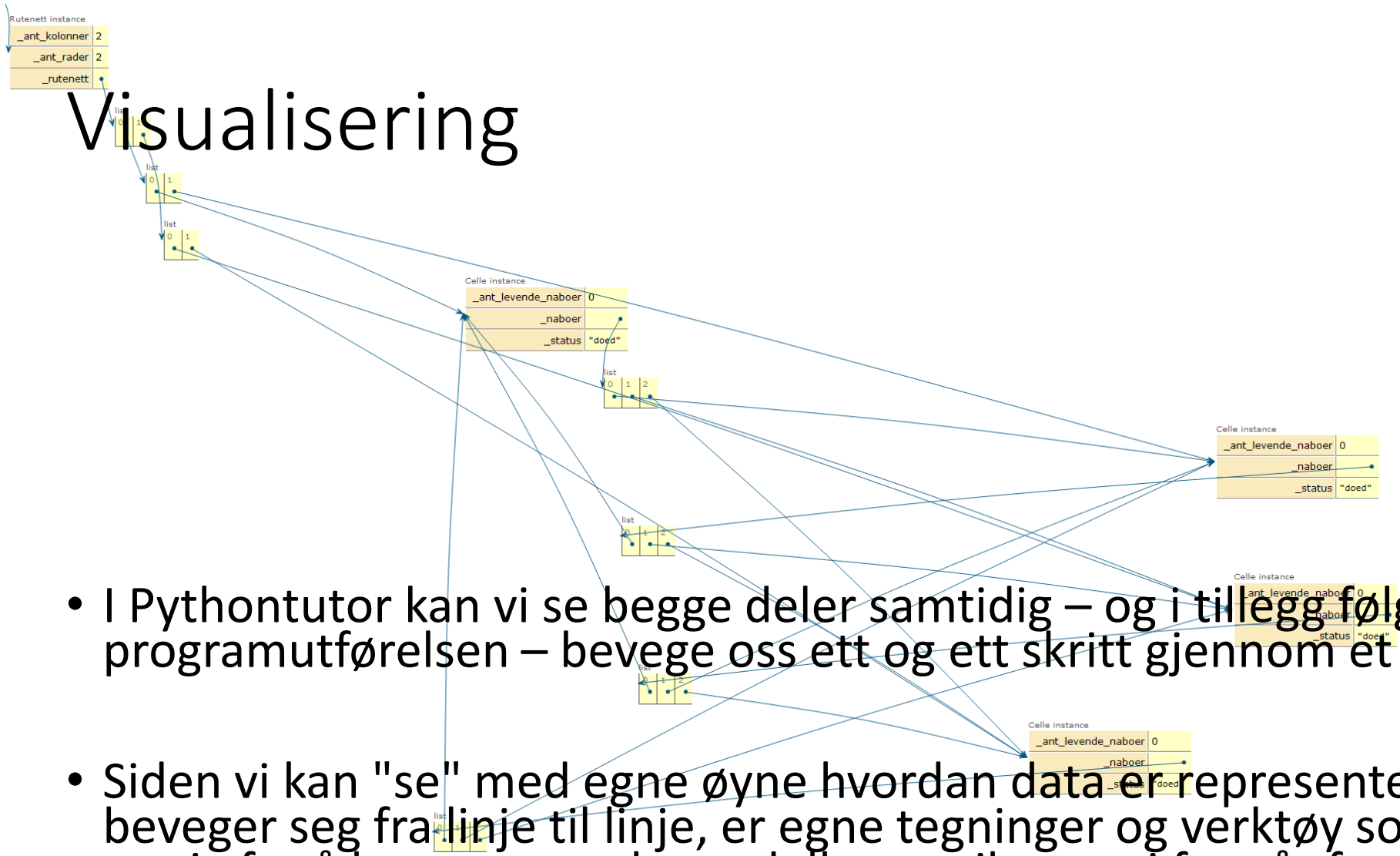
```
from klasse import Klasse
def hovedprogram ():
    …
    # oppretter og bruker objekter
hovedprogram()
```

Visualisering



- Når vi tegner variabler med verdier, objekter og referanser visualiserer vi hvordan *datastrukturen* ser ut på et gitt øyeblikk *under kjøring*

Visualisering



- I Pythontutor kan vi se begge deler samtidig – og i tillegg følge med på programutførelsen – bevege oss ett og ett skritt gjennom et program
- Siden vi kan "se" med egne øyne hvordan data er representert og utførelse beveger seg fra linje til linje, er egne tegninger og verktøy som Pythontutor nyttig for å bygge mentale modeller og sikre at vi forstår for hva som skjer når programmene våre utføres

Eksempel: Navn (startet forrige uke)

- Skriv en klasse Navn som skal huske og presentere på flere formater navn bestående av fornavn, mellomnavn og etternavn

- Uformelt grensesnitt
  - Konstruktør m/ parametere for for-, mellom- og etternavn
  - Metoder for å
    - hente på form egnet for sortering (Støre, Jonas Gahr)
    - hente naturlig (Jonas Gahr Støre)

Klassen Navn - grensesnitt

```
class Navn:
  # Konstruktør, tar argumenter til 3 navnedeler
  # instansvariabler for for-, mellom- og etternavn
  def __init__(self,fornavn,mellom,etter):

  # returnerer en string med navnet på sortert form
  def sorterbar(self):

  # returnerer en string med navnet på naturlig form
  def naturlig(self):
```

Klassen Navn

Filen navn.py

```
class Navn:
  def __init__(self,fornavn,mellom,etter):
    self._fornavn = fornavn
    self._mellom = mellom
    self._etter = etter

  def sorterbar(self):
    alfabetisk_navn = self._etter + ", " + self._fornavn + " " + self._mellom
    return alfabetisk_navn

  def naturlig(self):
    naturlig_navn = self._fornavn + " " + self._mellom + " " + self._etter
    return naturlig_navn
```

Klassen Navn: Testprogram

```
from navn import Navn

def hovedprogram():
    navn1 = Navn("Siri", "Moe", "Jensen")
    navn2 = Navn("Geir", "Kjetil", "Sandve")

    print (navn1.sorterbar())
    print (navn2.sorterbar())
    print (navn2.naturlig())

hovedprogram()
```

```
Jensen, Siri Moe
Sandve, Geir Kjetil
Geir Kjetil Sandve
```

Kan alle objekter endres?

Hva gjør disse metode-kallene på lister og strenger?

Hvordan pleier vi å bruke dem?

```
min_liste = [5, 10]
min_streng = "Hei!  "

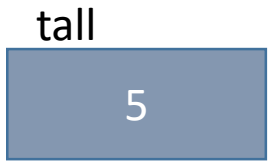
min_liste.append(5)

min_streng.lower()
min_streng = min_streng.lower()
```

Kan vi endre et objekt av klassen Navn etter å ha opprettet det?

Variabler i Python

- En variabel kan sees på som en "boks" som inneholder en verdi – et heltall, True eller False, eller et flyttall



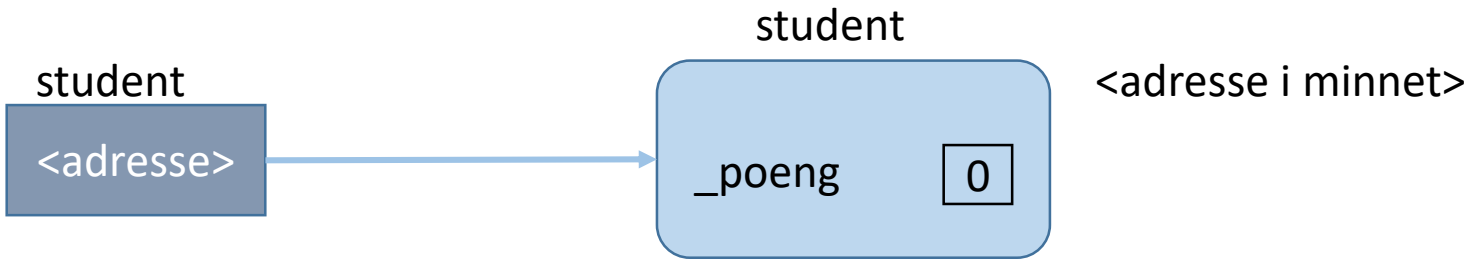
- For variabler som lagrer sammensatte verdier (objekter) trengs det et litt mer detaljert bilde:

Variablene inneholder ikke objektet selv –

men en referanse til objektet

Referanser til objekter

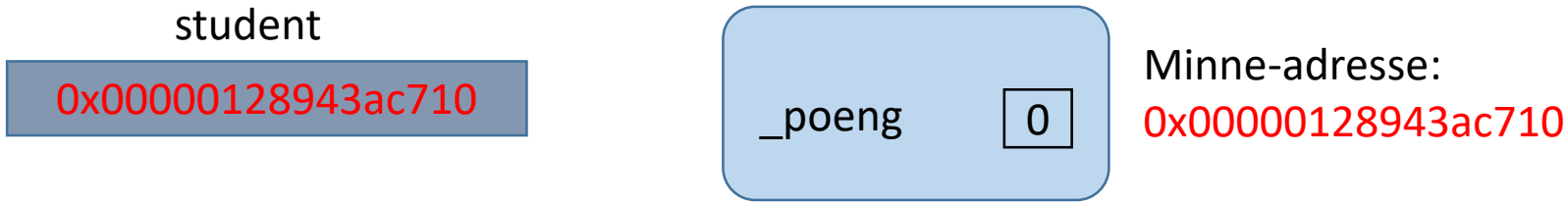
- Objekter lagres ikke direkte i variabler – variablene inneholder isteden bare referansen (adressen) til objektet.



- Tegnes ofte som en pil for å indikere at objektet ikke selv ligger i variabelen – men at variabelen kan "vise vei".
- Minneadressen – innholdet i variabelen – er en *objektreferanse* (*referanse*)

Referanser til objekter

- Litt nærmere sannheten (men veldig tungvint på tegninger...) :



=> Vi kommer til å holde oss til denne!



Referanser til objekter



- Variabler som holder rede på objekter kalles referansevariabler
- Gjør det mulig å ta vare på og bruke objekter når vi trenger dem, akkurat som heltallsvariabler husker heltall til vi trenger dem.
- Selve objektet kan lagres "hvorsomhelst" i minnet, og være stort eller lite – referansevariabelen trenger bare plass til en adresse
- Referansevariabler kan brukes for å kalle på metoder i objektet:

  ref_variabel.metode()

Forskjellen på referansen og objektet

- Når endrer vi en referansevariabel og når endrer vi objektet?

```
student = Student()          # Her endres innholdet i referansevariabelen
                             # (den refererer til det nye objektet)

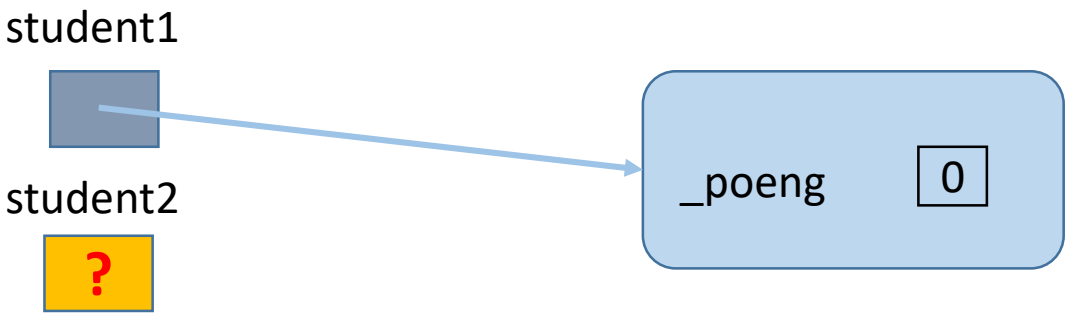
student.registrer_poeng(5) # Her endres innholdet i objektet
                             # variabelen refererer til ("peker på")
```

- Når vi snakker om *innholdet* i et objekt skjer det alltid med "dot-notasjon"
  - Enten utenfor klassen: **`student.registrer()`**
  - Eller i metodene inne i klassen: **`self._poeng += antall`**

Referanser til objekter

- Noen ganger er det spesielt viktig å ta hensyn til at referansen og objektet er to ulike ting!
- Hva skjer for eksempel om vi tilordner verdien fra en referansevariabel til en annen?

```
student1 = Student()
student2 = student1
```

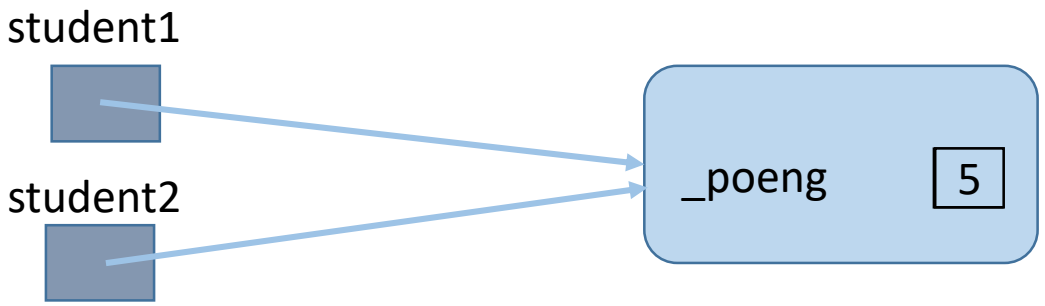


Å kopiere en referanse

- Vi kopierer *verdien* i variabelen på høyre side – akkurat som når vi kopierer en heltallsverdi
- Det er altså ikke objektet som blir kopiert – vi får bare en kopi av adressen/ referansen til objektet
- Denne verdien legges i variabelen på venstre side
- Begge referansevariablene refererer dermed til *samme objekt*

Referanser til objekter

```
student1 = Student()
student2 = student1
student2.registrer_poeng(5)
print (student1.hent_poeng())
```



Spørsmål:

Hva skrives ut i print-setningen?

Dette har du kanskje opplevd med lister?

```
liste = [1,2,3]
kopi = liste

kopi.append(5)

print(kopi)
print(liste)
```

- Utfør i Pythontutor

Hva skjer her? Fibonaccitall

```
def nyFib (fibTallene):
    nyttTall = fibTallene[-1] + fibTallene[-2]
    fibTallene.append(nyttTall)
    print("Liste med nytt tall: ", fibTallene)

start = [0,1]
print("Start-listen: ", start)

nyFib(start)

print("Start-listen: ", start)
```

- Fibonaccitallene: Hvert tall er summen av de to foregående
- Starter med 0, 1
- Hva skrives ut?

Hva skjer her? Fibonaccitall

```
def nyFib (fibTallene):
    nyttTall = fibTallene[-1] + fibTallene[-2]
    fibTallene.append(nyttTall)
    print("Liste med nytt tall: ", fibTallene)

start = [0,1]
print("Start-listen: ", start)

nyFib(start)

print("Start-listen: ", start)
```

- parameteren **fibTallene** får samme verdi som argumentet **start**
- den refererer med andre ord det samme liste-objektet
- append-metoden endrer liste-objektet den kalles på
- når nyFib returnerer, er listen som argumentet (start) refererte til, endret

pythontutor

Immutable/ "uforanderlige" objekter

- Noen klasser har ingen metoder som endrer instansvariablene, de kalles *immutable* – "uforanderlige"
- Hindrer feil som følge av at flere variabler refererer til samme objekt

```
class Navn:
  def __init__(self,fornavn,mellom,etter):
    self._fornavn = fornavn
    self._mellom = mellom
    self._etter = etter

  def sorterbar(self):
    alfabetisk_navn = self._etter + ", " + self._fornavn + " " + self._mellom
    return alfabetisk_navn

  def naturlig(self):
    naturlig_navn = self._fornavn + " " + self._mellom + " " + self._etter
    return naturlig_navn
```

Strenger endres aldri – oppretter bare en ny

- Metoder for å endre en streng endrer aldri objektet de kalles på
- I stedet er de laget som funksjonsmetoder som *returnerer en ny streng*
- Denne returverdien må vi evt ta vare på for å benytte videre
- pythontutor viser streng-variabler på samme måte som tall- og sannhetsvariabler

```
navn = "Erna Solberg"

# Strenger er immutable, returnerer en endret kopi
navn_med_små_bokstaver = navn.lower()

print(navn_med_små_bokstaver)
print(navn)
```

Kode og tegninger



- variabler tegnes som en navngitt boks (rektangel) med verdi inni
- hvis verdien er en int, boolean eller string skriver vi den vanligvis direkte inne i boksen (selv om string er et objekt vi kan kalle metoder på)

- andre objekter tegnes (vanligvis) som rektangler med runde kanter
- ofte angir vi klassen (typen) over objektet – da med : foran klassenavnet
- referanser tegnes som piler fra referansevariabel til objekt
- instansvariabler tegnes inne i objektet: Som bokser eller bare <_instansvariabel> = <verdi>



Oppgave: Klassen Rektangel

Rektangler er nyttige elementer i mange sammenhenger

=> Mulighet for gjenbruk

Skriv en klasse Rektangel som

- Har en konstruktør som tar to parametere: lengde og bredde
- En metode areal som returnerer rektangelets areal
- En metode endre som endrer størrelsen på rektangelet og har to parametere - lengde og bredde som angir hvor mye sidene i rektangelet skal endres med

Klassen Rektangel

```
class Rektangel:
  def __init__(self, lengde, bredde):
    self._lengde = lengde
    self._bredde = bredde

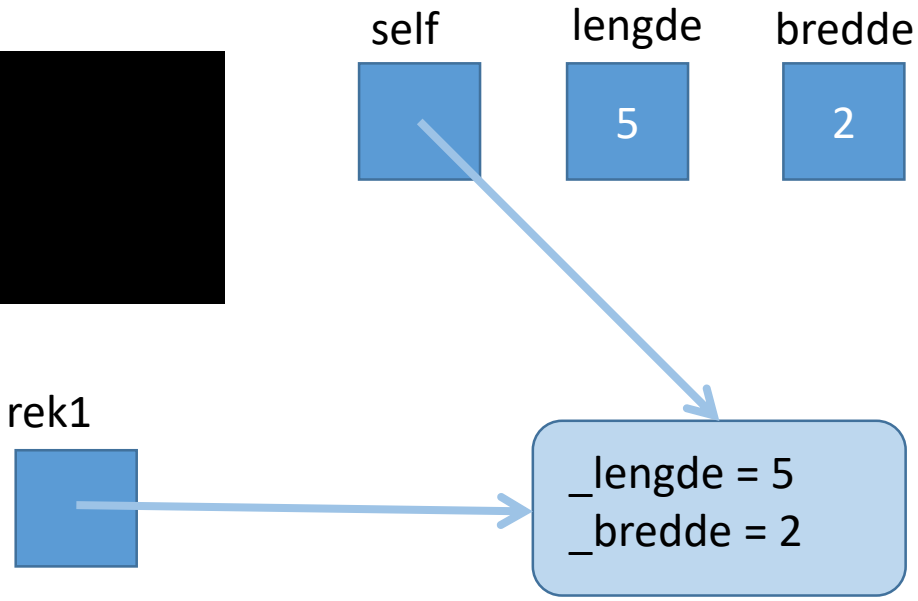
  def areal(self):
    return self._lengde*self._bredde

  def endre (self, lengde, bredde):
    self._lengde += lengde
    self._bredde += bredde
```

Hva skjer når vi oppretter objekter?

```
class Rektangel:
  def __init__(self, lengde, bredde):
    self._lengde = lengde
    self._bredde = bredde
```

```
rek1 = Rektangel(5, 2)
```

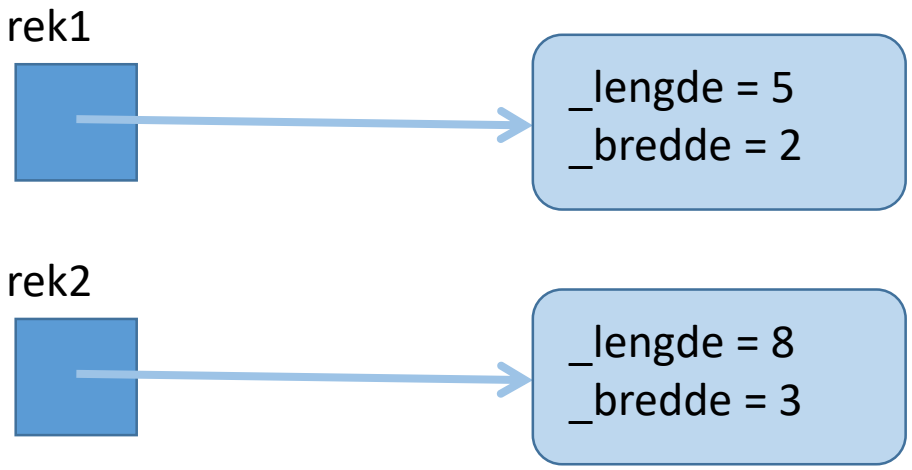


Hva skjer når vi oppretter objekter?

```
class Rektangel:
  def __init__(self, lengde, bredde):
    self._lengde = lengde
    self._bredde = bredde
```

```
rek1 = Rektangel(5,2)

rek2 = Rektangel (8,3)
```



Hva skjer når vi kaller metoder?

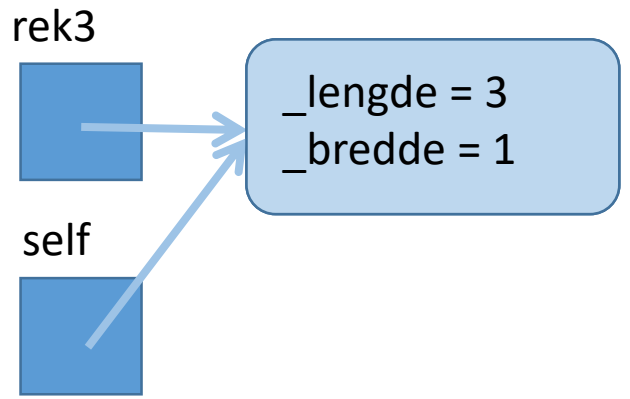
```
class Rektangel:
  def __init__(self, lengde, bredde):
    self._lengde = lengde
    self._bredde = bredde

  def areal(self):
    return self._lengde*self._bredde

  def endre(self,lengde,bredde):
     self._lengde +=lengde
     self._bredde += bredde
```

```
rek3 = Rektangel(5,2)
print (rek3.areal())

rek3.endre(-2,-1)
print (rek3.areal())
```



Kjøring

```
M:> python testrektangel.py
10
3
M:>
```

Lokale variabler i metoder

- Kan ikke aksesseres utenfor egen metode (i motsetning til instansvariablene)
- Oppstår og dør hver gang metoden kalles (som i funksjoner)

```
class Rektangel:
  def __init__(self, lengde, bredde):
    self._lengde = lengde
    self._bredde = bredde

  def areal(self):
    beregnet_areal = self._lengde*self._bredde
    return beregnet_areal
```

- Parametere oppfører seg som lokale variabler – MEN initialiseres med argumentene i kallet (som i funksjoner)

Verdien None

- Verdien None signaliserer at en referansevariabel ikke holder på noe objekt i øyeblikket.
- None kan *i noen tilfeller* være en nyttig initialverdi
- Kan være nødvendig å sjekke mot None for å unngå å kalle på en metode i et objekt som ikke finnes (gir ellers feil under kjøring):

- **if student is not None:**

  student.registrer_poeng(5)

- Mer om dette og sammenligning av referanser og objekter neste uke!

Fil: test_navn.py

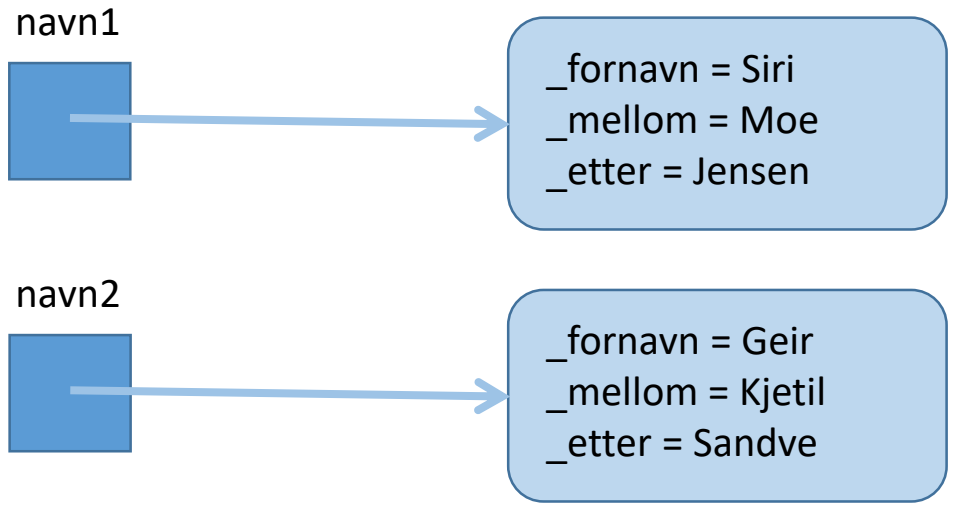
Klassen Navn: Testprogram

```
from navn import Navn

def hovedprogram():
    navn1 = Navn("Siri", "Moe", "Jensen")
    navn2 = Navn("Geir", "Kjetil", "Sandve")

    print (navn1.sorterbar())
    print (navn2.sorterbar())
    print (navn2.naturlig())

hovedprogram()
```




```
Jensen, Siri Moe
Sandve, Geir Kjetil
Geir Kjetil Sandve
```

Flere objekter av samme klasse

- En klasse er et mønster å lage objekter av
- Nyttig for å representere mange "ting" som følger samme mønster, f. eks. navn
- Skal lese inn et ukjent antall navn, opprette navneobjekter, legge dem i en liste og deretter skrive dem ut på sorterbar form (ikke sortert)

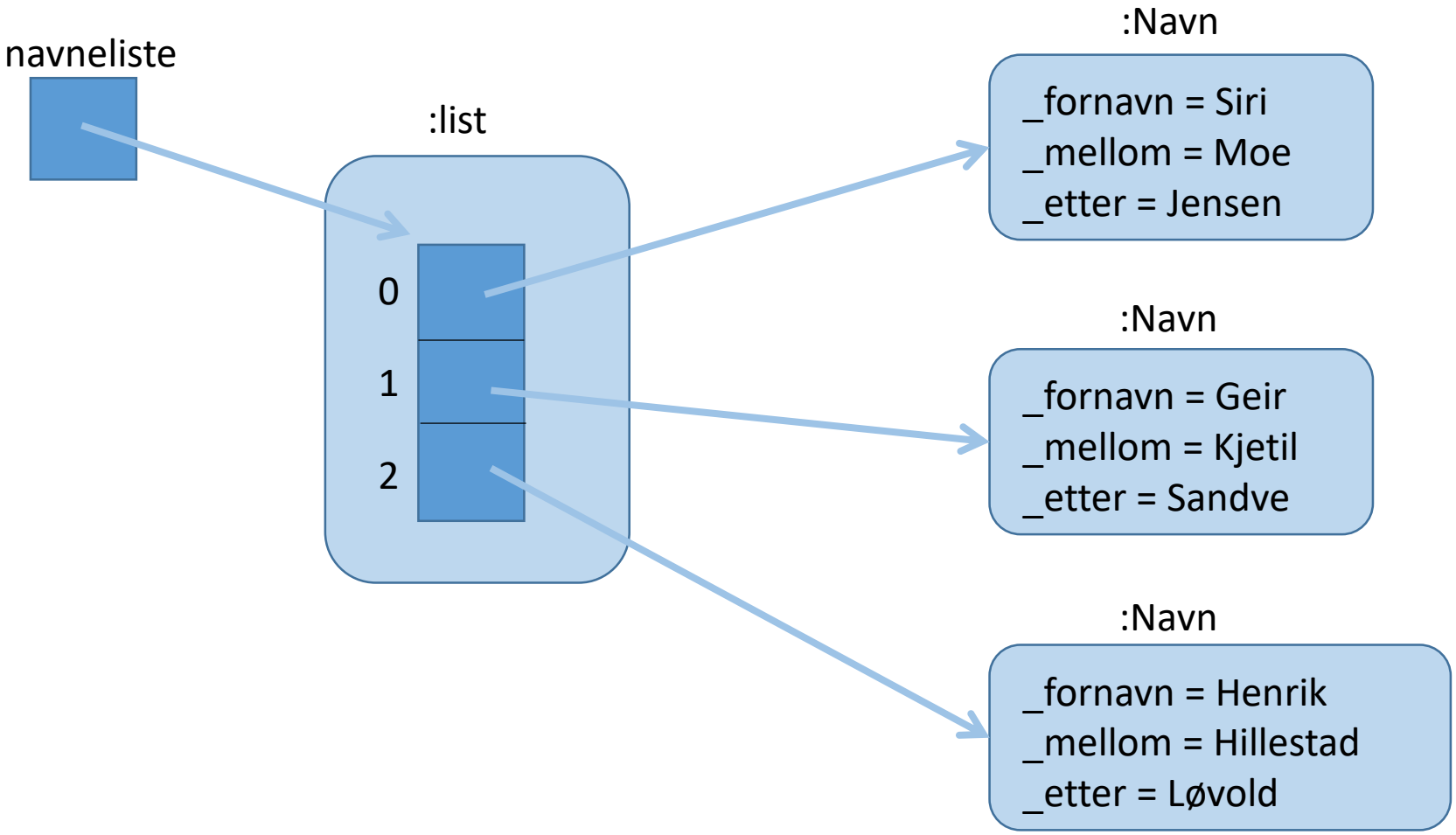
```
from navn import Navn

navneliste = []
les = input("Oppgi navn på naturlig form: ")
while les != "" :
    navnene = les.split()
    nytt = Navn(navnene[0],navnene[1],navnene[2])
    navneliste.append(nytt)
    les = input("Oppgi navn på naturlig form: ")

for et_navn in navneliste:
    print(et_navn.sorterbar())
```

Fil: test_flere_navn.py

Datastrukturen etter innlesing av tre navn



Eksempel: Klasse Person

- Skal lagre og skrive ut data om en person
  - Navn
  - Alder
  - (og mye mer etter hvert)

Implementasjon av Person

- Hvordan representere navnet til personen i klassen?
- Kan ha en instansvariabel som inneholder en tekst (string)

  ... men da vet vi ingenting om hvordan navnet er bygd opp

$\Rightarrow$ velger å bruke vår klasse Navn som en mer intelligent og fleksibel representasjon av personnavn

- Hvert Person-objekt får en instansvariabel som refererer et Navn-objekt
- Da kan et Person-objekt oppgi navnet sitt på flere former!

Implementasjon av Person

Filen person.py

```
class Person:
  def __init__(self,fulltNavn,alder):
    self._navn = fulltNavn
    self._alder = alder

  def skrivUt(self):
    print ("Navn: " + self._navn.naturlig() )
    print ("Alder: " + str(self._alder))
```

- Objekter kan ha instansvariabler som refererer til andre objekter
  - f eks en liste, eller
  - et objekt av en annen klasse, eller
  - et objekt av samme klasse som objektet selv (=> uke 10)

Klassen Person: Testprogram

Fil: test_person.py

```
from navn import Navn
from person import Person

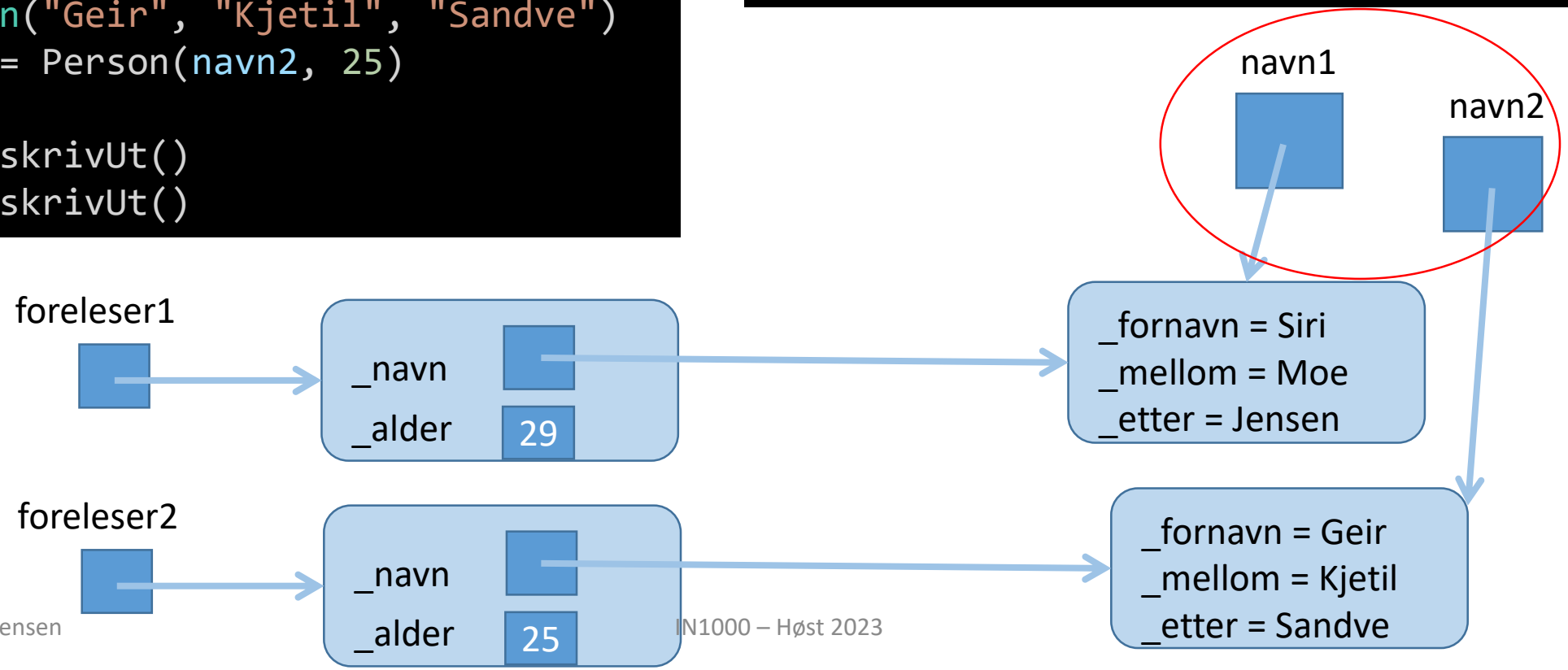
navn1 = Navn("Siri", "Moe", "Jensen")
foreleser1 = Person(navn1, 29)

navn2 = Navn("Geir", "Kjetil", "Sandve")
foreleser2 = Person(navn2, 25)

foreleser1.skrivUt()
foreleser2.skrivUt()
```

```
class Person:
  def __init__(self,fulltNavn,alder):
    self._navn = fulltNavn
    self._alder = alder

  def skrivUt(self):
    print ("Navn: " + self._navn.naturlig() )
    print ("Alder: " + str(self._alder))
```



Testprogram: Flere personer

fil: test_flere_personer.py

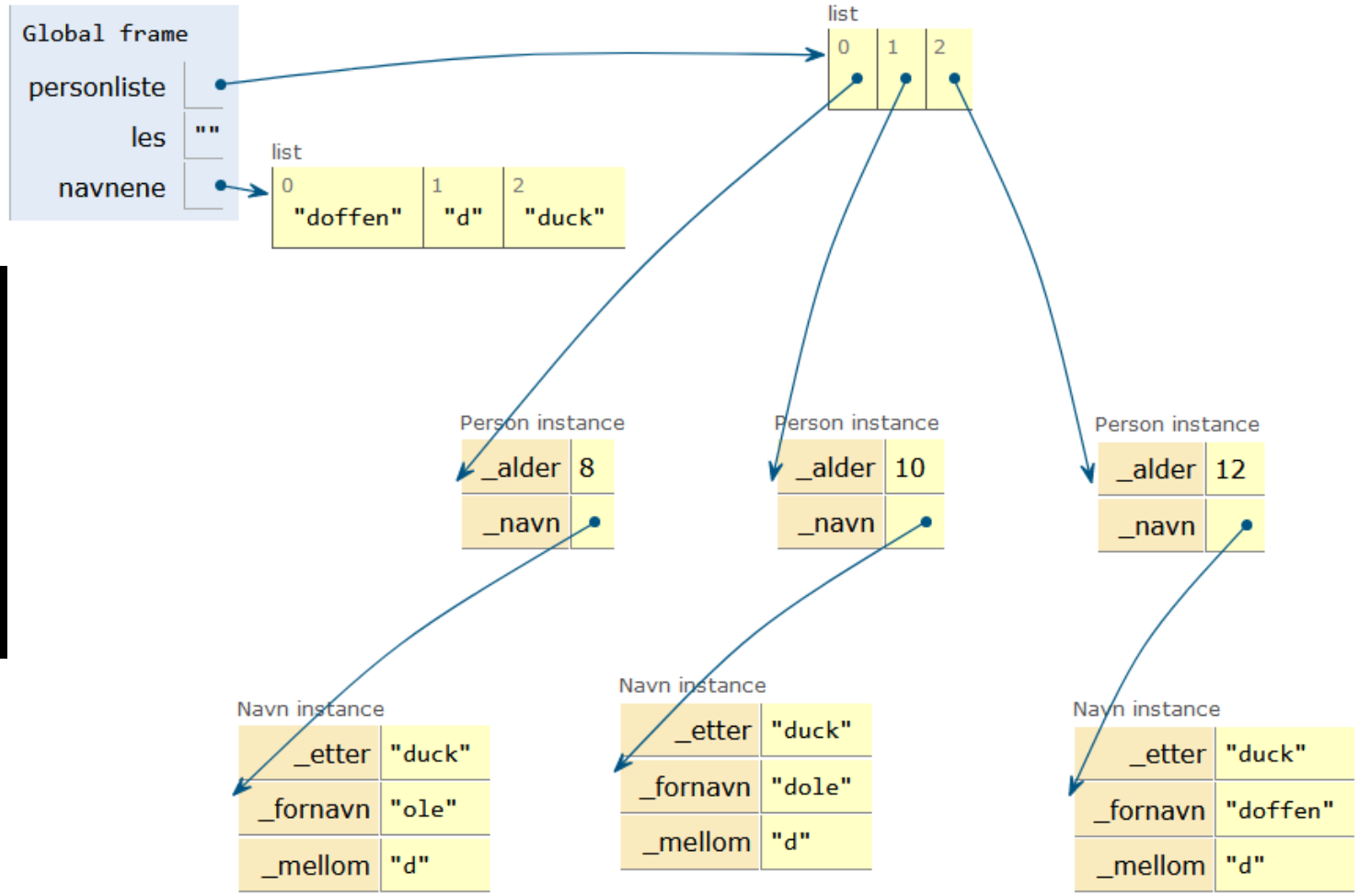
```
from navn import Navn
from person import Person

personliste = []
les = input("Oppgi navn på naturlig form: ")
while les != "" :
    navnene = les.split()
    nytt = Navn(navnene[0],navnene[1],navnene[2])
    alder = int(input("Oppgi alder: "))
    nyPerson = Person(nytt, alder)
    personliste.append(nyPerson)
    les = input("Oppgi navn på naturlig form: ")

for person in personliste :
    person.skrivUt()
    print("\n")
```

Visualisering i Pythontutor (forenklet)

```
Oppgi navn på naturlig form: ole d duck
Oppgi alder: 8
Oppgi navn på naturlig form: dole d duck
Oppgi alder: 10
Oppgi navn på naturlig form: doffen d duck
Oppgi alder: 12
Oppgi navn på naturlig form:
```



Læringsmål uke 8

- Forstå (mer av) hva som skjer bak kulissene når vi oppretter og bruker objekter
- Kjenne til forskjellen på å endre en referansevariabel og å endre objektet den refererer til
- Kunne skrive programmer med samlinger av (referanser til) objekter
- (Kunne sette seg inn i og skrive enkle programmer med flere klasser, og med objekter som refererer til andre objekter)

Neste uke

- "Magiske (spesielle) metoder" for egendefinerte klasser
  - Hvordan representere objekter med innhold som en streng?
  - Hvordan sjekke om to objekter er "like"?

- Objekter i ulike strukturer
  - Samlinger av objekter
  - Referanser mellom objekter av ulike klasser