

Åpen chat for spørsmål, leses i pausen.

Ubesvarte spørsmål: Bruk Mattermost felleskanal eller pm til meg – evt epost.

Underveis-evaluering og status: Mentimeter i slutten av forelesning

# Objektorientert programmering i Python

IN1000

Høst 2020 – uke 9

Siri Moe Jensen

# Innhold uke 9

## Mer komplekse strukturer

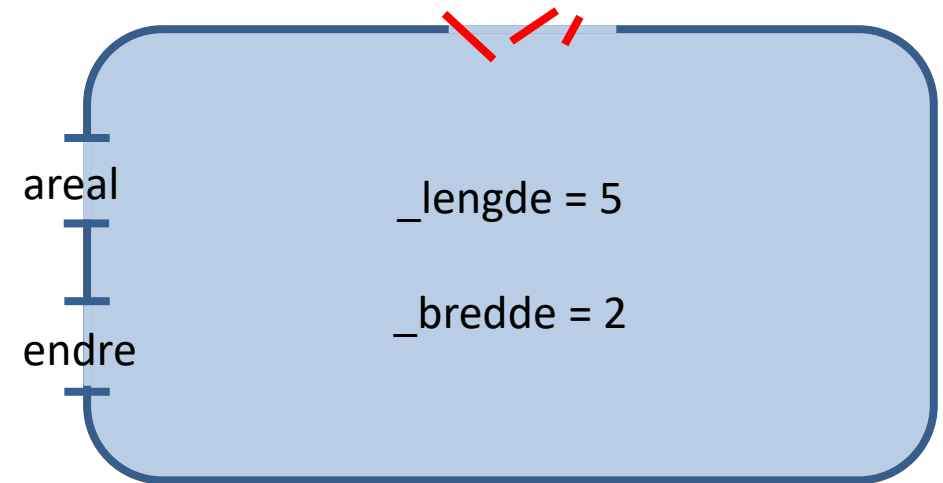
- Innkapsling og grensesnitt (repetisjon)
- Spesielle metoder i egendefinerte klasser
  - sammenligning av objekter
  - utskrift
- Samlinger av objekter i beholdere (containers) som liste, mengde og ordbok.
- Underveis-evaluering (Mentimeter)

# Klasse og grensesnitt

- Klassen er mønsteret vi lager objektene etter
- Klassedefinisjonen bestemmer hva objektene kan lagre av data (instansvariabler) og hva de kan utføre (instansmetoder)
- Grensesnittet er de metodene vi tilbyr programmereren som skal bruke klassen
- Alt annet i klassen navngir vi med '\_' først i navnet – de er "non-public" (private)
- Instansmetoder kan også være private! Brukes da kun av andre metoder inne i klassen

# Innkapsling :Tenk deg et hus...

- Bygget etter tegninger, med vegger og tak
- Planlagte åpninger - dører (og vinduer)
- Man *kan* også slå hull i tak eller vegger
- Men har da åpnet huset for regn, mus og rotter!
  
- Grensesnittet til en klasse definerer hvilke dører/ vinduer objektene skal ha.
- Når vi la være å slå hull i vegger og tak (dvs aksessere private metoder og variabler), er objektene tryggere og lettere å sikre.



objekt av klassen rektangel, som tilbyr to metoder i sitt grensesnitt:

- `areal`
- `endre`

# Hva har vi tilgang til i instansmetodene?

- Data:
  - instansvariablene (ved hjelp av `.self`). Når en metode avslutter, finnes fortsatt instansvariablene med sine verdier (så lenge det finnes en eller flere referanser til objektet)

Dessuten (akurat som vanlige funksjoner og prosedyrer)

- eventuelle parametere til metoden (eks: `lengde`, `bredde`). Disse oppstår når metoden blir kalt, og får verdi fra argumentene som sendes med. Når metoden er utført forsvinner de.
- eventuelle lokale variabler metoden definerer (eks: `areal`). Som parametere, men må gis verdi inne i metoden.

# Hva har vi tilgang til i instansmetodene?

- En metode kan bruke (kalle på) det samme som ellers i programmet:
  - prosedyrer som `print("tekst")`
  - funksjoner som `input("tekst")`
  - (andre) metoder: `referanse.metode(argumenter)`
- `referanse` kan være
  - til det objektet "denne" metoden ble kalt på: `self._bredde` i klassen `Rektangel`
- eller til et annet objekt! Referansen kan da hentes fra
  - en instansvariabel i objektet vi "er i": `self._navn.naturlig()` i klassen `Person`
  - en parameter eller lokal variabel til metoden vi er (som i funksjoner/ prosedyrer ellers)

# Referanser til objekter

- Ofte trenger vi ikke tenke på at selve objektet ikke ligger i variabelen
- Men noen ganger må vi huske forskjellen på referansen og objektet
  - Hvis vi tilordner verdien fra en referansevariabel til en annen (f eks ved parameter-overføring)
    - objektet kopieres ikke
    - variablene refererer til samme objekt!
    - endringer når objektet endres synes det fra begge referansevariabler
  - om vi sammenligner to referanser
    - Om *referansene* er like (inneholder samme adresse), er det samme objekt
    - Å sammenligne to *objekter* er litt mer komplisert

# Sammenligning av referansevariable

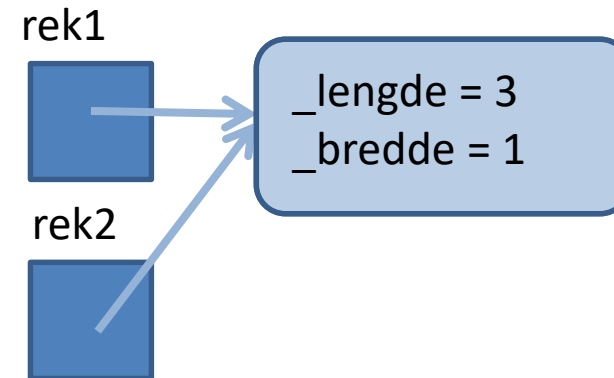
Kan sammenligne enten:

- Referansene, er det **samme** objekt?

- Bruk **is** for å sjekke

`rek1 is rek2`

`True`



- Objektene, er det **like** objekter (likt innhold)?

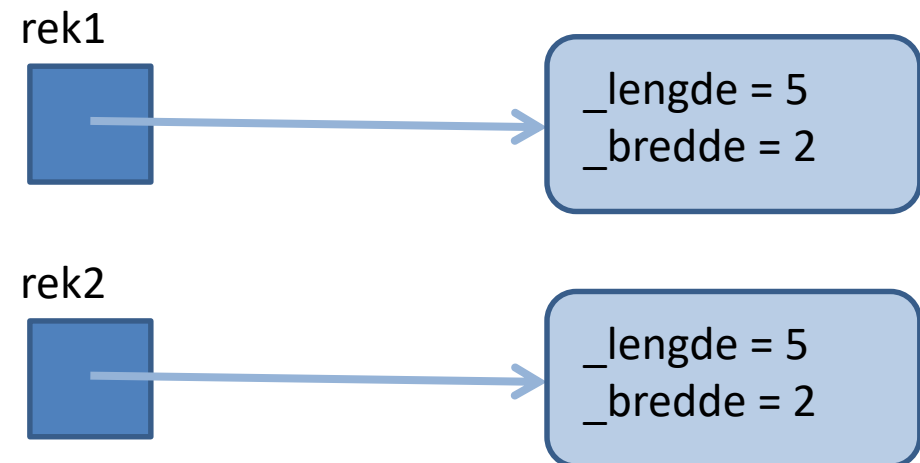
- Må sjekke instansvariablene i begge objekter!

- `rek1 is rek2`

`False`

- `rek1 == rek2`

`????`





# Når er *objekter* like?

- Den som har skrevet klassen vet (bestemmer) hva som gjør to objekter "like".
- Vi kan lage en egen metode i klassen som gjør dette. Hvis metoden får navnet `__eq__` vil operatoren `==` sammenligne objekter av klassen slik vi bestemmer.
- Dette er gjort i Pythons List-klasse, slik at  
`liste1 == liste2`  
sammenligner alle elementene i en liste for oss

# eksempel på `__eq__` metode

- Vil gjøre det mulig å sammenligne to rektangler for likhet
- Bestemmer at to Rektangel-objekter er like hvis og bare hvis lengden er den samme i begge, og bredden er den samme i begge
- Trenger en `__eq__` metode i klassen som skal sammenligne instansvariablene for lengde og instansvariablene for bredde i self-objektet og et objekt som refereres til av en parameter
- `__eq__` metoden tar som parameter (i tillegg til self) en referanse til et annet objekt (som vi skal sammenligne med) og returnerer True eller False

# eksempel på `__eq__` metode

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

    def __eq__(self, annen):
        return (self._lengde == annen._lengde and
                self._bredde == annen._bredde)

r1 = Rektangel(8,6)
r2 = Rektangel(8,6)
print (r1 == r2)

r3 = Rektangel(6,4)
print (r2 == r3)
```

Spørsmål: Er dette et brudd på innkapslingen i klassen Rektangel?

```
M:> rekt2.py
True
False
M:>
```

# Sammenligning av objekter

```
print (o1 == o2)
```

Hvis referansene er like (refererer samme objekt) => **True**

Hvis de refererer hvert sitt objekt av samme klasse, som ikke har `__eq__` metode:  
=> **False**

Hvis det finnes en `__eq__` metode i klassen:

- `__eq__` metoden utføres for `o1`, med `o2` som argument => tilsvarer "`o1.__eq__(o2)`"
- Avhengig av hvordan `__eq__ metode` sammenligner objektene vil den returnere **True** eller **False**

=> Uttrykket `(o1 == o2)` evaluerer til returverdien fra `__eq__` metoden i klassen til `o1` og `o2`

Overloading/ overlasting av operatoren ==

# Sammenligning av objekter II

(Objekter av ulike klasser: Et sært Python-tilfelle for spesielt interesserte)

```
print (o1 == o2)
```

Hvis o1 og o2 refererer objekter av ulike klasser (Anbefaling for IN1000: ikke gjør det)

- for klasser uten `__eq__` metode: => **False**
- for innebygde Python-typer og klasser med en `__eq__` metode som sjekker typen til hvert objekt: => kommer an på `__eq__` implementasjonen

=> f eks sammenligning av en tekst og en liste => false

# Spesielle metoder

- `__eq__` er en av mange "magiske" metoder som har en spesiell betydning i Python
- `__init__` har dere allerede brukt
- felles er
  - innledende og avsluttende dobbel underscore (`__`) i metodenavnet
  - kalles på andre måter enn ved metodenavnet
  - eks:
    - `__eq__` for `==` mellom to referanser
    - `__init__` ved opprettelse av nytt objekt

# Flere spesielle metoder

- Tabell 9.1 i boka viser en rekke andre spesielle metoder som kan implementere logiske (eks `==`, `!=`, `<`) og aritmetiske (eks `+`, `*`) operatører for en klasse som trenger det
- `__init__` kalles ved opprettelse av nytt objekt, oppretter og initierer (gir startverdi til) instansvariablene
- `__str__` og `__repr__` gjør om objekter til strenger

# Objekter som strenger

For å vise frem objekter (som strenger):

- `__str__`
  - kalles når vi bruker `print(referansevar)` og `str(referansevar)`
  - lager *brukervennlig* utskrift, lag den slik du ønsker
  - hvis den ikke finnes i klassen kalles `__repr__`
- `__repr__`
  - leverer en *komplett og entydig* representasjon av objektet
  - default: Modul , klassenavn og minneadresse for objektet
  - kalles ved utskrift av elementene i en liste
  - eller når `__str__` ikke finnes i klassen

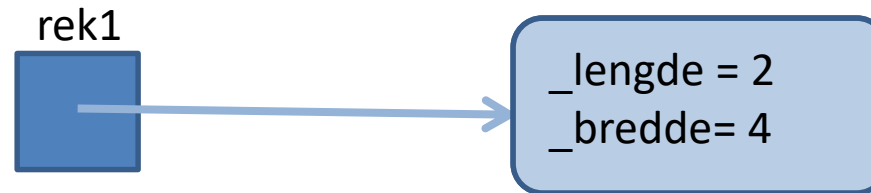
Lag gjerne denne

Finnes fra før



# "Skrive ut" objekter

Printer vi en referanse, får vi en (ofte litt kryptisk) tekst ut:



```
print(rek1)
```

```
C:\Programmering>
C:\Programmering>python rektangel.py
<__main__.Rektangel object at 0x0000025FC328E6A0>
C:\Programmering>
C:\Programmering>
```

*resultat av `__repr__`*

# eksempel på `__str__` metode

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

    def __str__(self) :
        penStreng = "Lengde: " + self._lengde + \
            ", bredde: " + self._bredde)
        return penStreng

r1 = Rektangel(8,6)
print (r1)
```

```
>python rek.py
Lengde: 8, bredde: 6
```


⇒ Husk at en `__str__` metode skal returnere, ikke printe ut en streng!  
(Du vet ikke hva de som bruker klassen ønsker å gjøre med strengen)

# En referanse trenger ikke ligge i en variabel

- En referanse kan ligge i en variabel (som rek1) – eller være en returverdi fra en funksjon eller metode (som igjen kan være kalt på en referanse som var returverdi fra en metode – osv osv)
- Et uttrykk som evaluerer til en referanse kan brukes der en referansevariabel kan brukes

```
from rektangel import Rektangel
print(Rektangel(10,15).areal())
```

# Oppretter nytt Rektangel objekt og  
# kaller på metoden areal for dette,  
# printer til slutt returverdi fra areal



\_lengde = 10  
\_bredde = 15

# Vi kan bruke "dot-notasjon" i flere ledd

- Kaller da metoder på returverdi fra en annen metode

```
ordliste = "Dette er en tekst ".lower().strip().split()  
print(ordliste)
```

Spørsmål:

- Kunne vi byttet rekkefølgen på de to siste?
- Hva skrives ut?

- Svar:
  - nei, `split()` returnerer (referanse til) en liste, ikke en tekst (som `strip()` må kalles på)
  - ['dette', 'er', 'en', 'tekst']
- NB: Lesbarhet – og rekkefølgen er ikke likegyldig!

# Samlinger av verdier (se uke 3)

- Beholdere (containers) er viktige verktøy i programmering
- Gjør det mulig å organisere og arbeide med samlinger av objekter
- Beholdere tilbyr ulike egenskaper – velges ut fra behov
- Så langt har vi sett på
  - Lister (List). Inneholder verdier med fast rekkefølge, nummerert
  - Mengder (Set). Unummerert, uten dubletter
  - Ordbøker (Dictionary). Par av unik *nøkkel* (typisk tekst) og *verdi*. Ingen konstant rekkefølge.
- Verdiene kan være referanser til objekter
- Slike referanser kan selv være samlinger – for eksempel lister

# Innhold i lister (se også uke 3)

- Kan ha lister av heltall, strenger, eller (referanser til) objekter av egne klasser
- Kan også ha en liste med lister:
  - En liste er et objekt av klassen (den innebygde typen) `list`
  - .. som kan holde orden på og manipulere en samling av objekter
  - Og tilbyr et grensesnitt for dette (med metoder som `minListe.append(elem)` og `minListe.pop()`)
  - `elem` kan her være en annen liste, med f eks heltall

# Eksempel: Informatikk-emner (kurs)

- Vi skal lage et program for å velge informatikk-emner
- Initielle krav: Kunne liste opp alle emner med id (emnekode), antall poeng og høst eller vår-semester
- Designer en klasse **Emne** med instansvariabler som over
- Bruker en liste for å organisere **Emne**-objekter

Oppgave: Skriv klassen Emne

# En klasse for emner

```
class Emne :
    def __init__(self, emnekode, sem, stp) :
        self._emnekode = emnekode
        self._semester = sem
        self._studiepoeng = stp

    def __str__(self):
        linje = (self._emnekode + "(" +
                self._semester + ") : " +
                str(self._studiepoeng) + " studiepoeng")
        return linje
```



# Samlinger av Emne-objekter

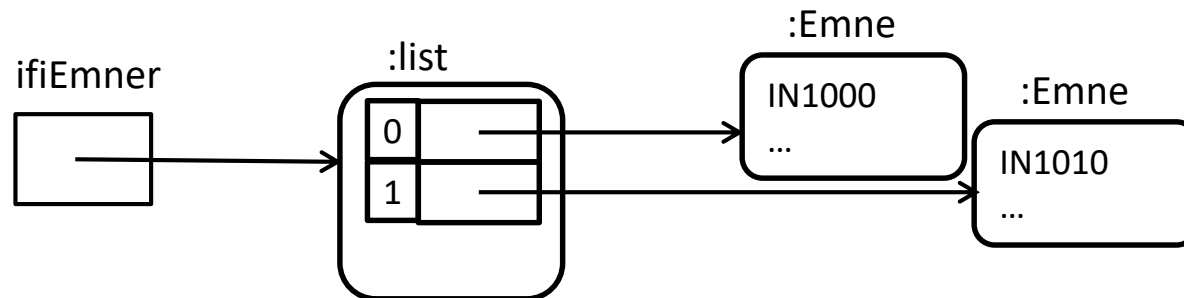
- Mer presist: Samlinger av referanser til Emne-objekter
- En liste kan inneholde (referanser til) objekter
- Dictionaries (ordbøker, maps) kan også ha referanser som *verdier*, typisk med en (unik) instansvariabel fra objektet som nøkkel
- Eks: Dictionary med Emne-objekter
  - Nøkkel: Emnekode
  - Verdi: Referanse til objektet for det emnet

(referanser til)

# Liste med objekter av egen klasse

- Eksempel: ifiEmner
- Hvert element i listen er (en referanse til) et emne-objekt

```
ifiEmner = []  
ifiEmner.append(Emne("IN1000", "host", 20))  
ifiEmner.append(Emne("IN1010", "var", 20)) # ..osv osv  
  
for ettEmne in ifiEmner :  
    print(ettEmne) # kaller __str__()
```

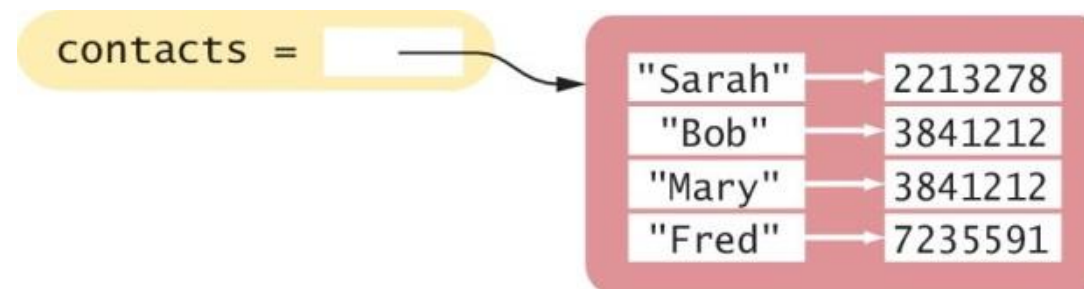


```
IN1000(host): 20 studiepoeng  
IN1010(var): 20 studiepoeng
```

# Opprette Dictionary (ordbok)

- Et program som slår opp telefonnummer
- Bruker en ordbok der navn er nøkkel, og telefonnummer er verdien

```
contacts = { "Fred": 7235591, "Mary": 3841212, "Bob":  
            3841212, "Sarah": 2213278 }
```



# Ordbok med referanser

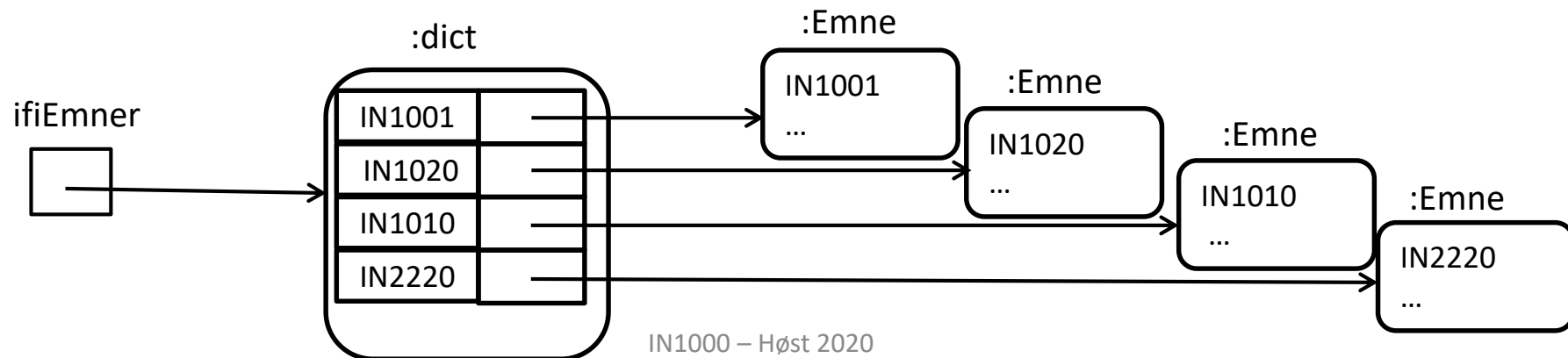
- Verdiene i en ordbok (dictionary) kan være referanser til objekter
- Eksempel: Ordbok `iflEmner` med emner
- Nøkkel (entydig): Emnekode
- Verdi: Referanse til et Emne-objekt

# Ordbok med referanser II

```
ifiEmner = {}                                #opprettet tom ordbok

ifiEmner["IN1000"] = Emne("IN1000","host",10)
ifiEmner["IN1010"] = Emne("IN1010","var",10)
# etc

for ettEmne in ifiEmner.values() :
    print(ettEmne)        # Kaller __str__()
```



# Fra uke 8: Klassen Person

- Hvordan representere navnet til personen i klassen?
- Kan ha en instansvariabel som inneholder en tekst  
... men da vet vi ingenting om hvordan navnet er bygd opp

⇒ 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!
- Trening: Utvid og skriv om klassen Person med en `_str_` metode, og bruk denne i `skrivUt`-metoden.

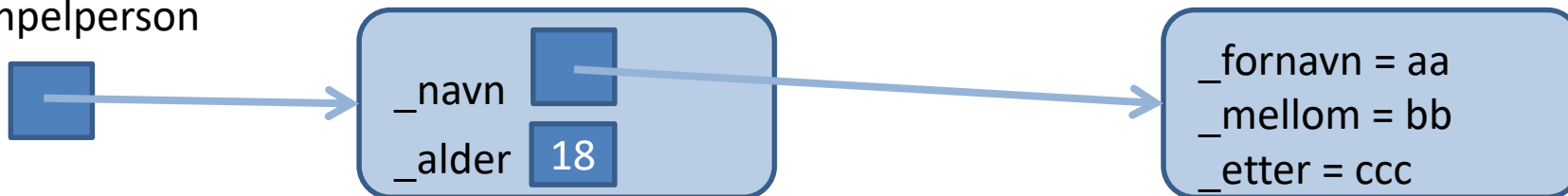
# Implementasjon av Person

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))
```

eksempelperson



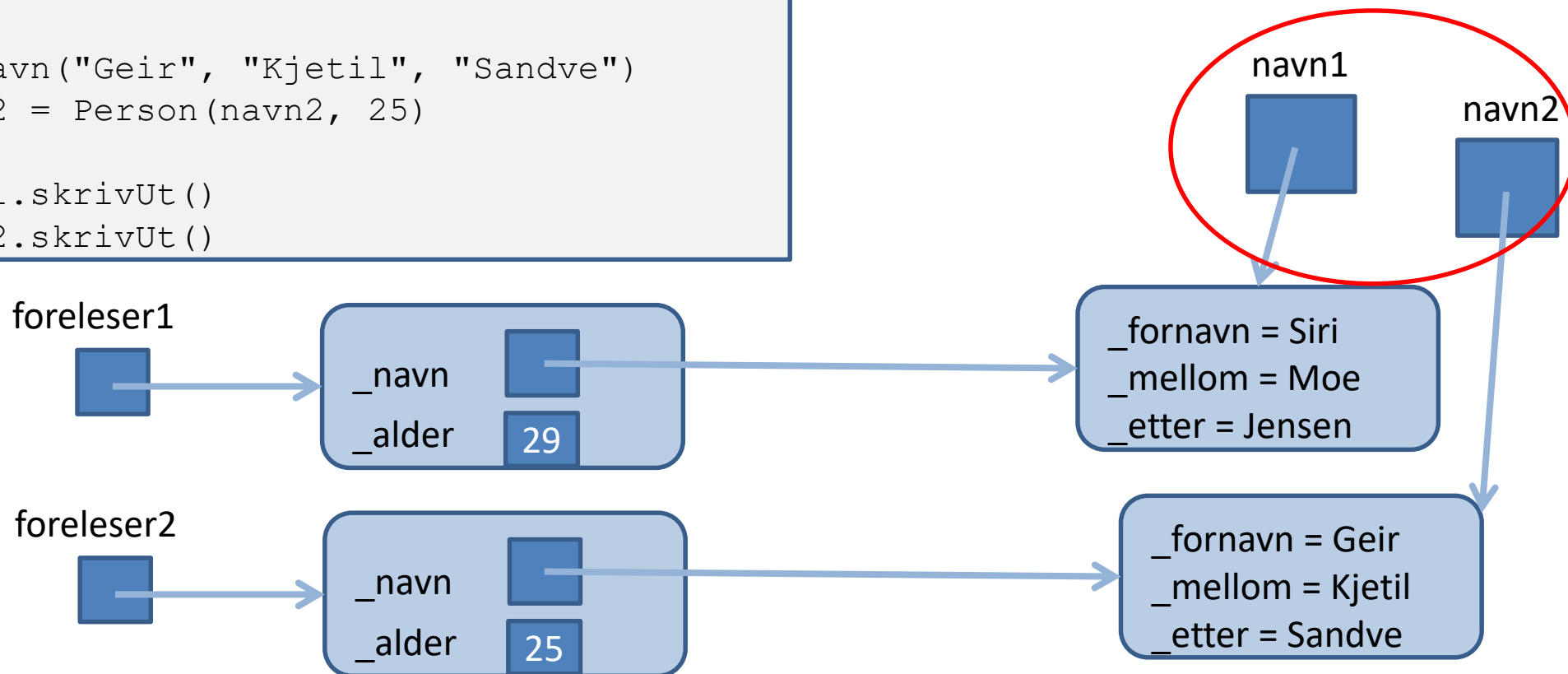
# Klassen Person: Testprogram

```
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()
```





# Testprogram I: Bygge struktur

```
from navn import Navn
from person import Person

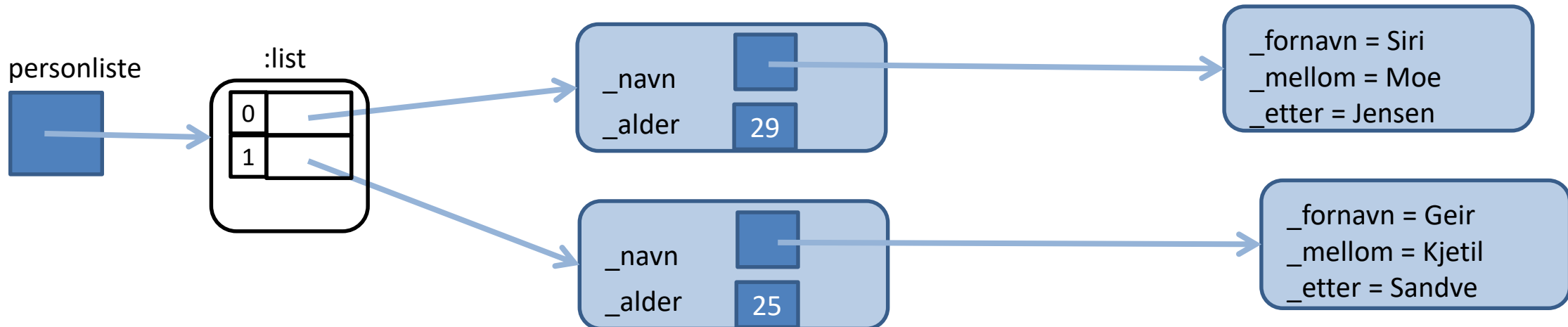
personliste = []
les = input("Oppgi navn på naturlig form: ")
while les != "":
    navnene = les.split()
    nytt = Navn(navnene[0],navnene[1],navnene[2])
    alder = int(input("Oppgi alder: "))
    nyPerson = Person(nytt, alder)
    personliste.append(nyPerson)
    les = input("Oppgi navn på naturlig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

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

# Testprogram I: Bygge struktur

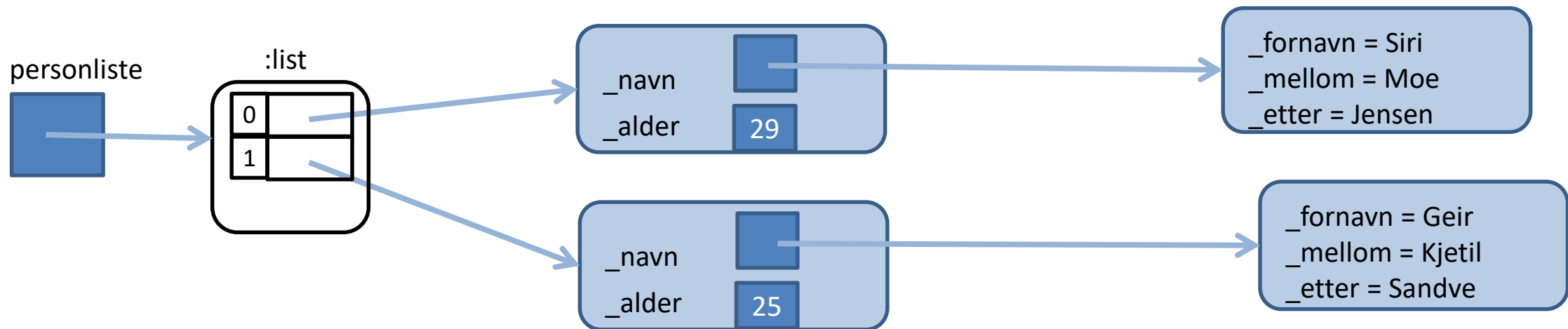
```
from navn import Navn
from person import Person

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: ")
```



# Testprogram II: Skrive ut alle personer

```
:  
:  
while les != "" :  
    :  
    :  
  
for person in personliste :  
    person.skrivUt()  
    print("\n")
```



# Oppsummert spesielle (magiske) metoder

- Er ikke innebygget fordi de avhenger av semantikken i den enkelte klassen – som bare du som programmerer klassen kan bestemme
- Ofte nyttig å skrive `__str__` (og noen ganger `__eq__`) for egne klasser
- Kan være behov for spesielle metoder for andre logiske operatorer (<, >, !=, etc) – for eksempel for å sortere
- Skal aldri kalles direkte/ med navn. For eksempel
  - ikke `rek.__str__()`, men `str(rek)`
  - ikke `rek1.__eq__(rek2)`, men `rek1 == rek2`

# Neste uke

- Nye anvendelser av det vi har vært gjennom
- Todimensjonale strukturer – representasjon og visualisering
- Viktige strukturer i informatikk: Lenkelister, trær, grafer
- Livekoding av kollektiv-rutenettet i Oslo

# Menti.com; kode 48 55 55 6

- Gir muligheter for tilpasninger dette semesteret
- Anonymt
- Holdes åpen litt utover forelesningstid, lenke kommer på semestersiden
- Alt leses og tas med videre!

Erstatter *\*ikke\** Fagutvalgets koursevaluering

# Takk for i dag!