Objektmanipulasjon, referanser, flere objekter og klasser

IN1000

Høst 2018 – uke 8

Siri Moe Jensen

Læringsmål uke 8

Repetisjon fra forrige uke
- Definere en klasse, opprette og arbeide med objekter: How-to

- Forstå (mer av) hva som skjer bak kulissene når vi oppretter og bruker objekter
- Kunne manipulere referanser og vite hvordan self og None brukes
- Kunne sette seg inn i enkle programmer med objekter av flere klasser, samlinger av objekter, og objekter som refererer andre objekter

Beskjeder: Oblig 1-6

- Godkjenning Oblig 1-6
  - Nytt krav av administrative grunner: 15 poeng.
  - Samme pensum og progresjon dvs ingen faglig betydning.
  - Devilry vil være oppdatert ca 20. oktober
  - Kan også selv gå gjennom egne innleveringer og sjekke poengsum

Beskjeder: Oblig 7 (og 8)

- Både oblig 7 og oblig 8 må godkjennes for å gå opp til eksamen

- Fusk blir fulgt opp - se tekst og lenker på semestersiden (under Obligatoriske innleveringer) om krav til eget arbeid.
- Samarbeid om teori og andre oppgaver – skriv egen kode.
- Lever i god tid før fristen i tilfelle tekniske problemer eller annet. Hvis du leverer flere ganger er det siste versjon som gjelder.

- Oblig 7 er en større oppgave enn dere har hatt før, men morsom ☺
  - Beregn god tid på å forstå spillet og oppgaven.
  - Klassen Celle bør være skrevet og testet denne uken.
  - Seminartimene vil gi nyttig input til denne oppgaven.

Arbeid med emnet (13 t/ uke)

- 2 t: Forelesningen: Overblikk over pensum
- 2 - ∞ t: Lab og programmering på egen hånd: Eksperimentere med nye ting, la gammelt stoff "sette seg". Identifisere spørsmål. Feil er læringspunkter!
- 2 - ∞ t: Seminartimer, kollokvier, Fredags-Python: Oppklaring, detaljer og eksempler, live programmering, diskusjon av alternative løsninger, trene på å snakke om kode.
- 1 - ∞ t: Emneressurser, nettet (obs kilder...): Teori, "hvorfor"

Hvordan lage egne klasser?

```
class Student :

  def __init__(self) :
    self._antMott = 0

  def registrer(self) :
    self._antMott = self._antMott + 1

  def lesOppmote(self) :
    return self._antMott
```

klassenavn — konstruktør — metode — funksjons-metode

Innkapsling

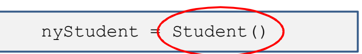
- I IN1000 følger vi læreboka for gjennomføring av innkapsling:
  - Kun et definert grensesnitt skal aksesseres fra utsiden av klassen (public)
  - Vi aksesserer ikke instansvariable utenfra klassen
- For å indikere at instansvariable og eventulle metoder utenom grensesnittet kun skal brukes i klassen (non-public) bruker vi _ som første tegn i alle navn på
    - instansvariable og
    - metoder som ikke skal kalles fra utenfor klassen
  - Dette er i henhold til PEP 8, Style Guide for Python Code
  - Kun en anbefaling til medprogrammere, Python hindrer ikke bruk

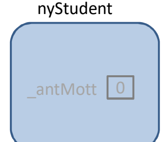
Sitat PEP8:

- A style guide is about consistency. Consistency with this style guide is important. Consistency within a project is more important. Consistency within one module or function is the most important.

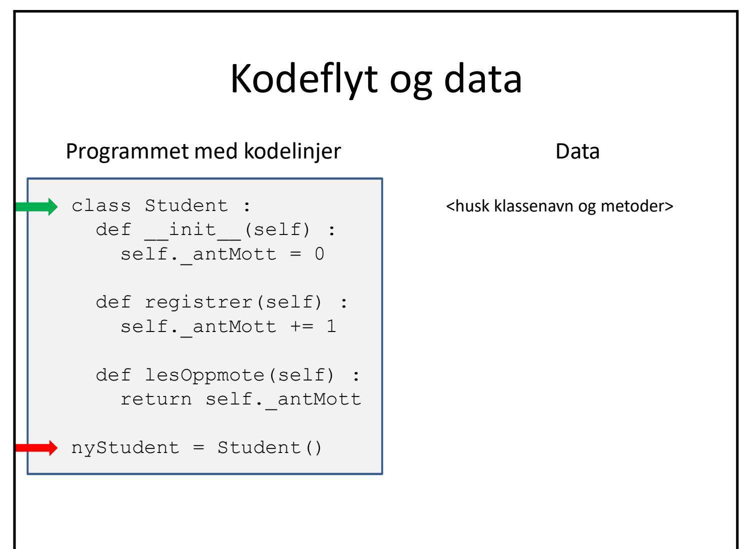
..og hvordan opprette et objekt?

- Opprette objekter av egen klasse:

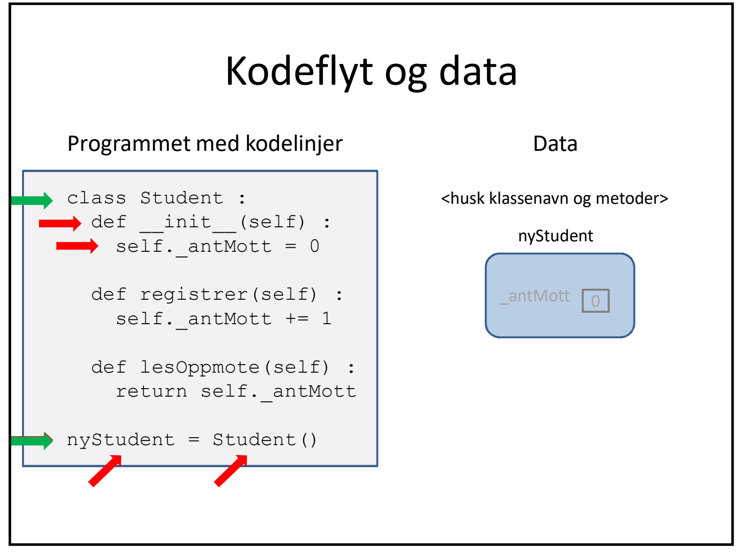




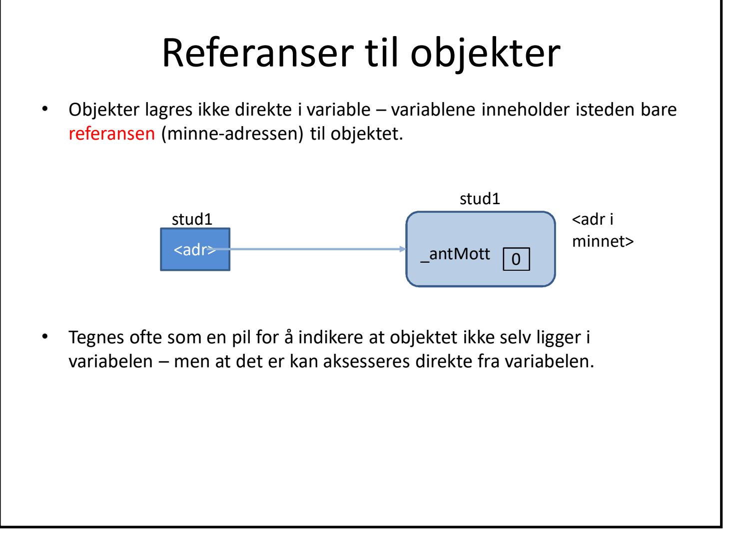
- Noen innebygde klasser har snarveier for opprettelse av objekter (i stedet for klassenavnet)
  - "" oppretter en ny **`str`**
  - [] oppretter en ny **`list`**



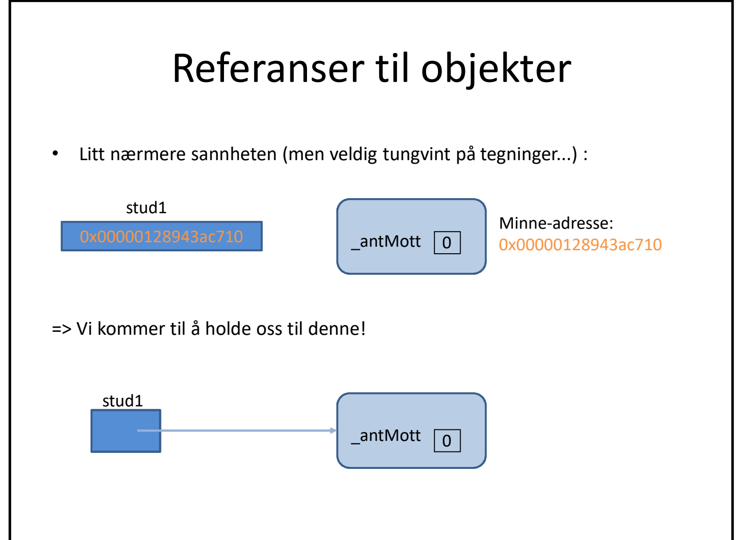
Kodeflyt og data
Programmet med kodelinjer
Data
class Student :
def __init__(self) :
self._antMott = 0
def registrer(self) :
self._antMott += 1
def lesOppmote(self) :
return self._antMott
nyStudent = Student()
<husk klassenavn og metoder>



Kodeflyt og data
Programmet med kodelinjer
Data
class Student :
def __init__(self) :
self._antMott = 0
def registrer(self) :
self._antMott += 1
def lesOppmote(self) :
return self._antMott
nyStudent = Student()
<husk klassenavn og metoder>
nyStudent
_antMott
0



Referanser til objekter
Objekter lagres ikke direkte i variable – variablene inneholder isteden bare referansen (minne-adressen) til objektet.
stud1
<adr>
stud1
_antMott
0
<adr i minnet>
Tegnes ofte som en pil for å indikere at objektet ikke selv ligger i variabelen – men at det er kan aksesseres direkte fra variabelen.



Referanser til objekter
Litt nærmere sannheten (men veldig tungvint på tegninger...) :
stud1
0x00000128943ac710
_antMott
0
Minne-adresse:
0x00000128943ac710
=> Vi kommer til å holde oss til denne!
stud1
_antMott
0

Referanser til objekter

- Variabler som holder objekter kalles derfor gjerne referansevariable
- Gjør det mulig å ta vare på og bruke objekter når vi trenger det, akkurat som heltallsvariable husker heltall til vi trenger dem.
- Selve objektet kan lagres "hvorsomhelst" i minnet, og være stort eller lite – referansevariabelen trenger bare lagre en adresse
- Spesielt for referansevariable: Kan kalle på metoder i objektet:

```
refVariabel.metode()
```

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
  - om vi sammenligner to referanser
  - nå vi sender med en referanse som argument til en funksjon/ metode der objektet endres

Kode og tegninger

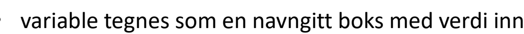
```
kode-segmenter er stort sett i grå bokser
```

- variable tegnes som en navngitt boks med verdi inni

alder

18

- objekter tegnes som rektangler med runde kanter
- referanser tegnes som piler



- nyttig med tegninger når datastrukturene blir mer kompliserte
- pythontutor viser kodeflyt og hvilke data som finnes i minnet

Eksempel: Klassen Rektangel

- Rektangler er et nyttige elementer i mange sammenhenger – for eksempel for å beskrive vinduer på en skjerm, eller for å beskrive, regne på eller dele opp et areal – eiendomsdeling, rom-planlegging, maling av husvegger, ..

=> Mulighet for gjenbruk

- Skriv en klasse `Rektangel` som
  - Har en konstruktør som tar to parametre `lengde` og `bredde`
  - En metode `areal` som returnerer rektangelets areal
  - En metode `reduser` som endrer størrelsen på rektangelet og har to parametere `lengde` og `bredde` som angir hvor mye sidene i rektangelet skal reduseres 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 reduser (self, lengde, bredde) :
    self._lengde -=lengde
    self._bredde -= bredde
```

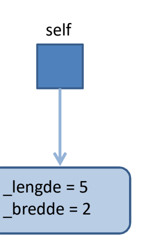
Hva skjer når vi oppretter et objekt?

```
rek1 = Rektangel(5,2)

rek2 = Rektangel (8,3)
```

- Det opprettes et nytt objekt av klassen Rektangel
- Konstruktøren kalles automatisk, og
  - en referanse til det nye objektet plasseres i parameteren `self`
  - evt andre parametere får verdi fra argumenter
  - Konstruktøren definerer og initierer instansvariablene i objektet
  - returnerer en referanse til objektet til kallstedet
- **Instansvariable** opprettes og initialiseres i *hvert nye objekt*

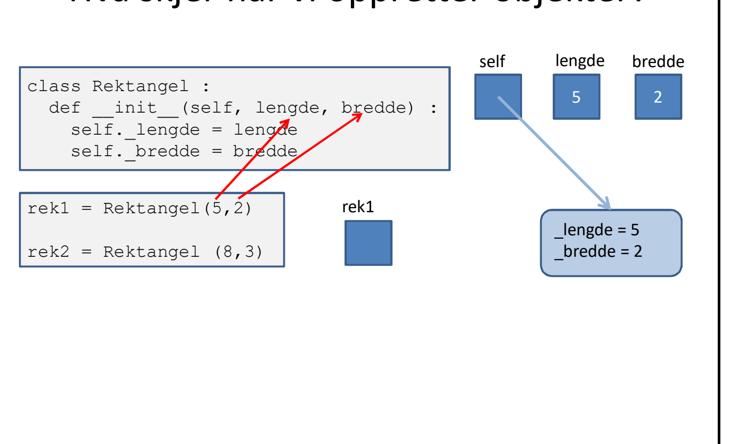
Objektreferansen `self`

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

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

  def reduser (self, lengde, bredde) :
    self._lengde -=lengde
    self._bredde -= bredde
```



- I konstruktøren får `self` referansen til det nye objektet
- Inne i en metode får `self` samme verdi som objektreferansen metoden er kallt på (variabelen før .)

Hva skjer når vi oppretter objekter?



Hva skjer når vi oppretter objekter?

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

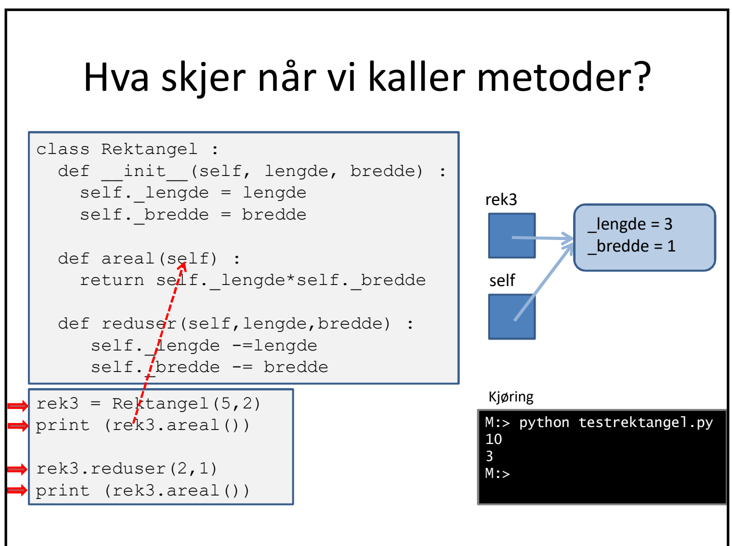
```
rek1 = Rektangel(5,2)

rek2 = Rektangel (8,3)
```

rek1

_lengde = 5
_bredde = 2

rek2

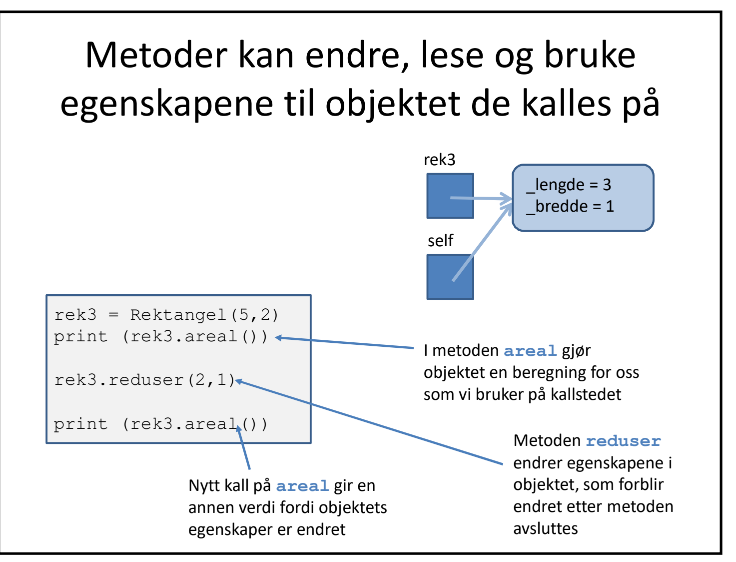
_lengde = 8
_bredde = 3



Hva skjer når vi kaller metoder

- Et objekt «lever» og kan brukes så lenge vi har en referanse til det
- Objektene kan gi oss «svar» på noe eller bli endret, ved kall på sine metoder. Om vi endrer objektet blir disse liggende i objektet
- Returverdier fra metoder brukes når vi skal bruke verdier eller resultater fra objektet på kallstedet
- Må se på dokumentasjonen av klassens grensesnitt for å vite nøyaktig hvordan kall på metoder virker

    eks: navn.upper() eller liste.append("Per")



Objekter kopieres ikke med =

- Tilordning til objektreferanse

```
rek1 = Rektangel (3, 5)
rek2 = rek1
# ikke nytt objekt!

rek2.reduser(2,1)
```

rek1

rek2

_lengde = 1
_bredde= 4

- Når vi endrer rek2 gjelder endringene også for rek1!

Verdien None

- Verdien `None` signaliserer at en referansevariabel ikke holder på noe objekt i øyeblikket.
- `None` kan 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 feil under kjøring)

```
rek3 = None
areal = rek3.areal()
```

kjøretidsfeil!

```
if rek3 is not None :
  areal = rek3.areal()
```

ok, blir ikke utført

Eksempel: Navn

- 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 (Hareide, Knut Arild)
    - hente naturlig (Knut Arild Hareide)

Klassen Navn implementert

Filen navn.py

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

  def sortert(self) :
    alfNavn = self._etter + ", "\
              + self._fornavn + " " + self._mellom
    return alfNavn

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

Klassen Navn: Testprogram

```
from navn import Navn

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

print (navn1.sortert())
print (navn2.sortert())
print (navn2.naturlig())
```

navn1

_fornavn = Siri
_mellom = Moe
_etter = Jensen

navn2

_fornavn = Geir
_mellom = Kjetil
_etter = Sandve

```
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, for eksempel navn:

```
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 navn in navneliste :
        print(navn.sortert())
```

Eksempel: Klasse Person

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

```
class Person :
  def __init__(self,fulltNavn,alder) :

  def skrivUt(self) :
```

- Vil gjenbruke klassen vi har laget for navn

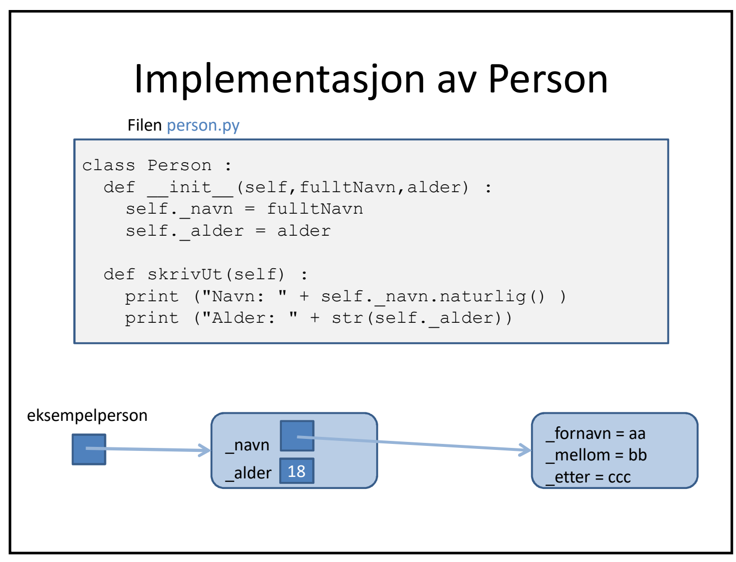
Implementasjon av Person

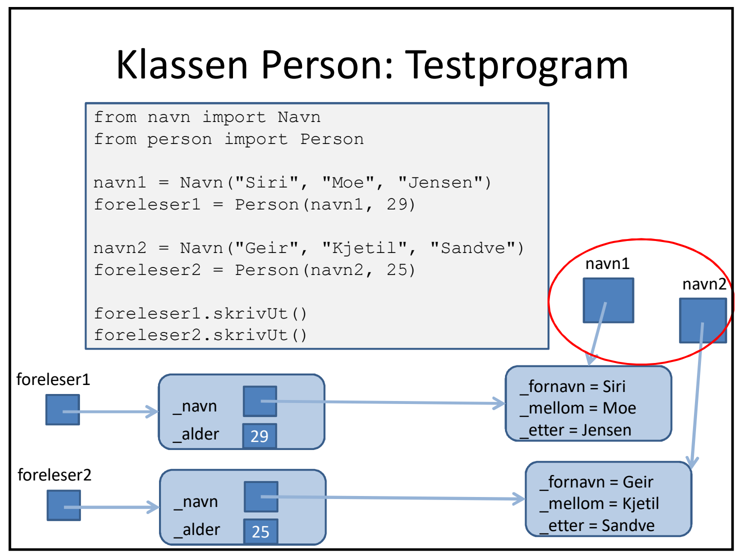
- Hvordan representere navnet til personen i klassen?
- Kan ha en instansvariabel som inneholder en tekst

  ... 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 få tak i navnet sitt på flere former!





Testprogram: Flere navn

```
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 navn in navneliste :
      print(navn.sortert())
```

Testprogram: Flere personer

```
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")
```

Neste uke

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