KLASSER, OBJEKTER OG REFERANSER

LÆRINGSMÅL FOR KLASSER, OBJEKTER OG REFERANSER

- Kjenne til motivasjon og bakgrunn for objektorientert programmering
- Kunne definere en klasse med instansvariabler, metoder og konstruktør
- Kunne opprette objekter av egendefinert klasse og bruke deres tjenester gjennom metodekall
- Forstå sentrale begreper som grensesnitt og innkapsling
- Kjenne til utviklingsprosessen for en klasse gjennom design, implementasjon og testing
- 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 enkle programmer med flere klasser og objekter som refererer til andre objekter

OBJEKTORIENTERT PROGRAMMERING

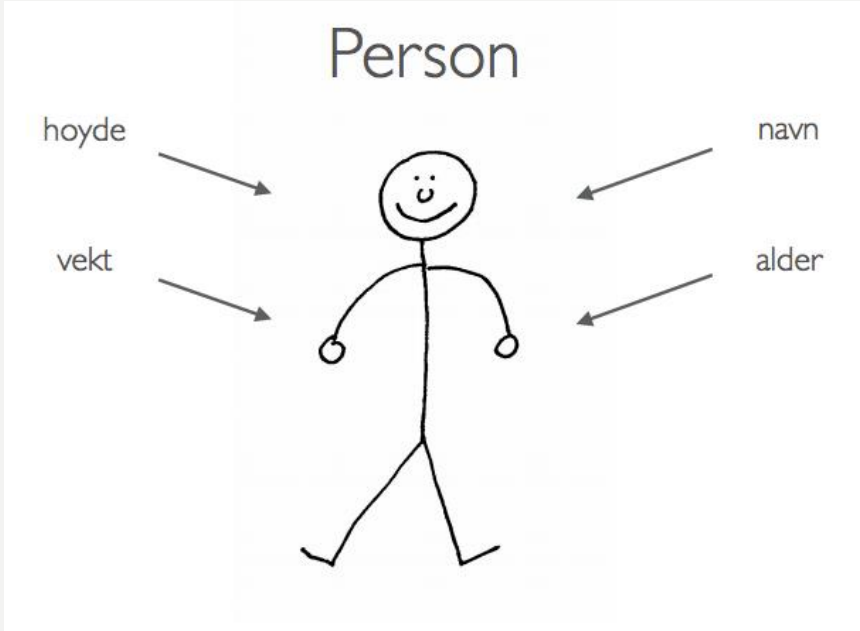
- Handler om å modellere verden
  - Vi kan lage en modell av noe som finnes i verden i en datamaskin
  - Denne modellen har egenskaper og handlinger den kan gjøre
  - Egenskaper = instansvariabler
  - Handlinger = instansmetoder

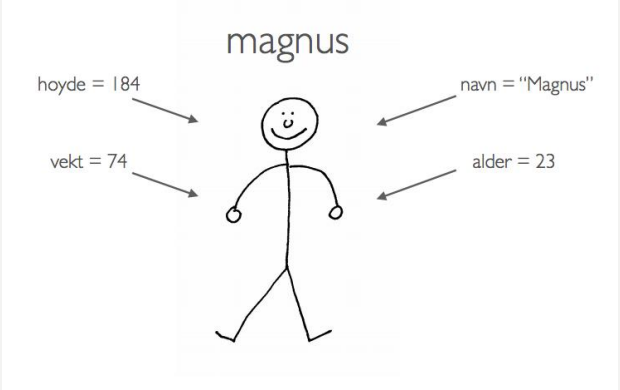
- Eksempel:
  - Vi kan modellere en person
  - En person har egenskaper/instansvariabler:
    - navn, alder, høyde osv.
  - En person har handlinger/instansmetoder:
    - si navnet sitt, bli eldre, vokse høyere osv.

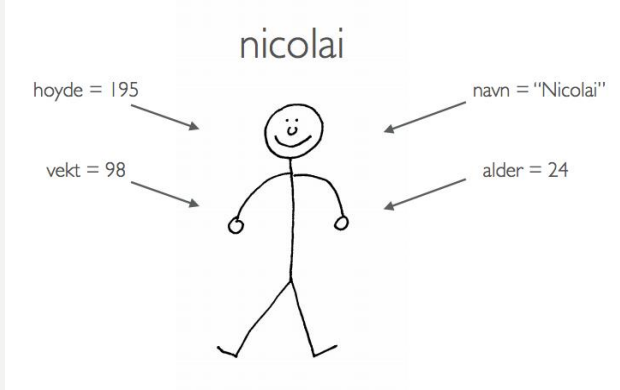
KLASSER OG OBJEKTER

- En klasse er en mal eller oppskrift

- Vi kan lage objekter/instanser av en klasse
  - Person er en klasse.
    Klassen har egenskaper og handlinger som er felles for alle personer
  - magnus og nicolai er objekter.
    De er instanser av klassen Person.
    De er ulike objekter av samme klasse.







GRENSESNITT OG INNKAPSLING

- Alle klasser har et public interface – et grensesnitt
- Grensesnittet inneholder metodene i klassen

- Eksempel:
  - Vi har et objekt «bil», og den har en metode kjoer(km).
  - Vi vet ikke hvordan metoden er definert, men vi skjønner hva den gjør.
    Den tar inn hvor mange km bilen kjører, og kanskje oppdaterer kilometerstanden og drivstoffet.
  - Vi vet ikke hvordan det gjøres, vi vet bare at det skjer når vi kaller metoden.
    - Dette er **innkapsling**!

HVORDAN DEFINERE EN KLASSE

- Definer klassen
  - Definer konstruktøren og instansvariablene
  - Definer metodene
- Instansvariabler skrives med self. foran
  - De er tilgjengelige overalt i klassen
  - Self brukes til å referere til seg selv, altså «denne» instansen av klassen
- Instansvariabler og -metoder som ikke skal aksesseres utenfra, skrives med _ foran

```
# Definer klassen
class Person:

    # Konstruktør
    def __init__(self, navn, alder):
        # Instansvariabler
        self._navn = navn
        self._alder = alder

    def si_hei(self):
        print("Hei")
```

KONSTRUKTØR

- Bestemmer hva som må være med ved opprettelsen av ett objekt

- Eks: Å lage en person uten navn er litt rart
  - Navn kan sendes med som parameter til konstruktøren
  - Parametere må lagres i instansvariabler

```
# Konstruktør
def __init__(self, navn, alder):
    # Instansvariabler
    self._navn = navn
    self._alder = alder
```

HVORDAN OPPRETTE OG BRUKE OBJEKTER

- Lag en ny fil – f.eks. hovedprogram.py
- Importer klassene som skal brukes
- Lag en prosedyre som skal holde på koden – f.eks. hovedprogram()
- Lag objektene og sørg for å lagre dem i variabler eller i en samling

```
from person import Person
```

```
def hovedprogram():
    person1 = Person("Kari", 20)
    person2 = Person("Ola", 23)
```

MAGISKE METODER

```
def __init__(self):
    pass

def __str__(self):
    pass

def __repr__(self):
    pass

def __eq__(self, other):
    pass

def __ne__(self, other):
    pass

def __hash__(self):
    pass
```

- Det finnes flere magiske metoder
- Pensum i IN1000 er:
  - __init__
  - __eq__
  - __str__

__EQ__

- eq sammenligner to objekter
- Vi kan definere hva som skal til for at to objekter regnes som like ved hjelp av __eq__
- __eq__ er en magisk metode som kjøres automatisk når vi bruker == mellom to objekter
- __eq__ må alltid ta objektet vi skal sammenligne med som parameter og returnere True/False

__EQ__

- Hvis vi skal holde styr på et lite antall katter kan vi f.eks. sammenligne navnene

```
class Katt:
def __init__(self, n, a, f, k):
    self._navn = n
    self._alder = a
    self._farger = f
    self._kjonn = k

def hent_navn(self):
    return self._navn

def __eq__(self, annen):
    return self._navn == annen.hent_navn()
```

__EQ__

- Hvis vi skal holde styr på et stort antall katter kan vi bruke et ID-nummer

```
class Katt:
def __init__(self, id):
    self.id = id

def hent_id(self):
    return self._id

def __eq__(self, annen):
    return self._id == annen.hent_id()
```

__STR__

- __str__ representerer et objekt som en streng

- Vi bestemmer hva som er en god print-setning for klassen.
  Vi kan definere hvilke instansvariabler som skal vises når vi printer et objekt.

- __str__ er en magisk metode som kjøres automatisk når vi bruker print() eller str() på et objekt

- __str__ tar ikke imot noen parametere (bortsett fra self) og må returnere en streng
  - Metoden skal ikke printe, kun returnere!

__STR__

- Hvis vi skal holde styr på et lite antall katter kan vi returnere bare navnet

```
class Katt:
def __init__(self, n, a, f, k):
    self._navn = n
    self._alder = a
    self._farger = f
    self._kjonn = k

def __str__(self):
    return self._navn
```

__STR__

- Hvis vi skal holde styr på et stort antall katter kan vi returnere navn og et ID-nummer

```
class Katt:
def __init__(self, n, id):
    self._navn = n
    self._id = id

def __str__(self):
    return f"{self._id}: {self._navn}"
```

REFERANSER

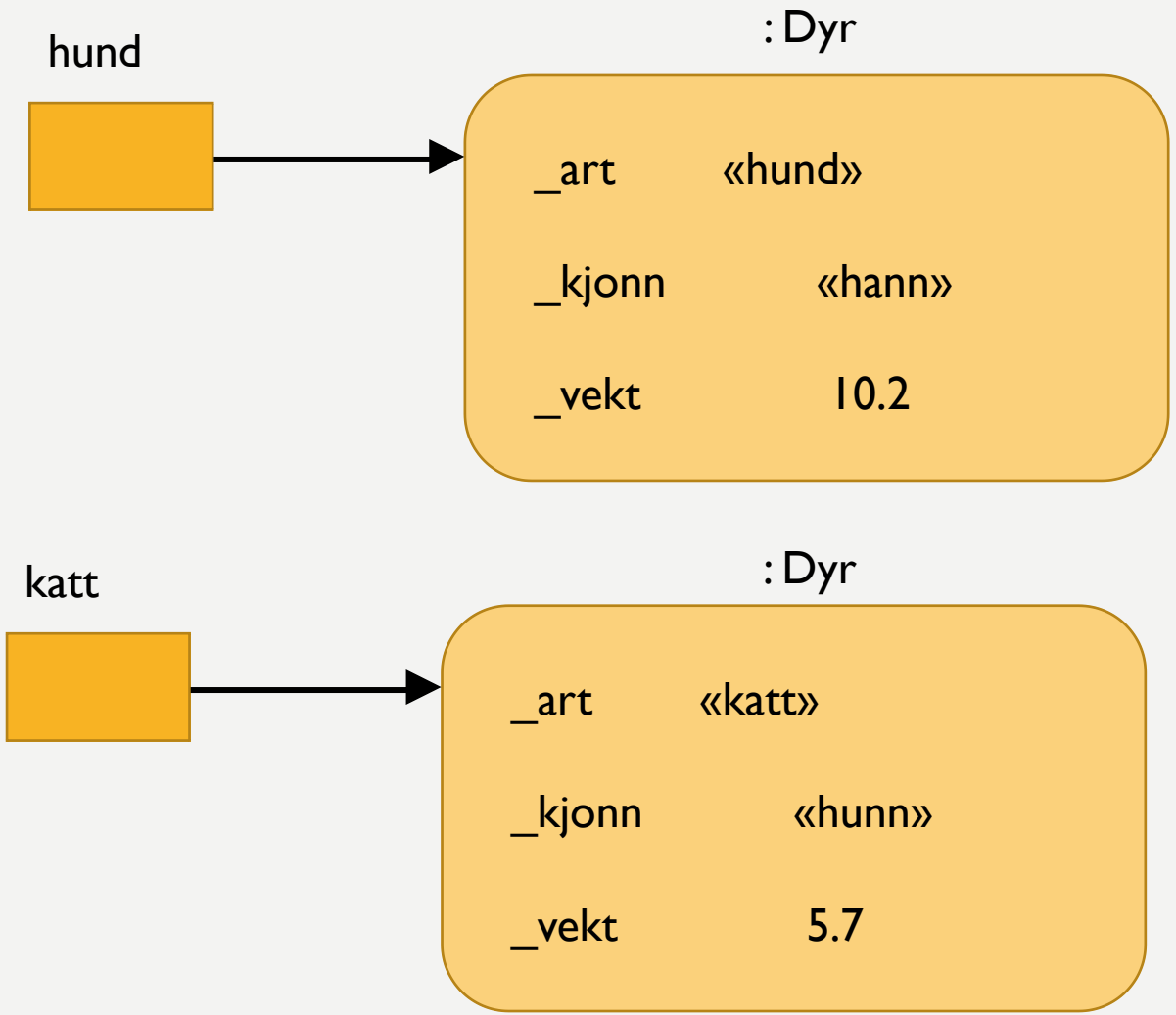
- Referanser er en måte å lagre og få tak i objekter på
- Objekter lagres ikke direkte i variabler
  - variablene inneholder i stedet en referanse (adresse) til objektet
- Variabler som holder rede på objekter kalles referansevariabler
- Referansevariabler kan brukes for å kalle på metoder i objektet

REFERANSER

Eksempel

```
hund = Dyr(«hund», «hann», 10.2)
katt = Dyr(«katt», «hunn», 5.7)
```

- «hund» er en referanse til et objekt av typen dyr
- Hva skjer om referansen (pilen) endrer seg dersom man f.eks. skriver:
  - katt = hund

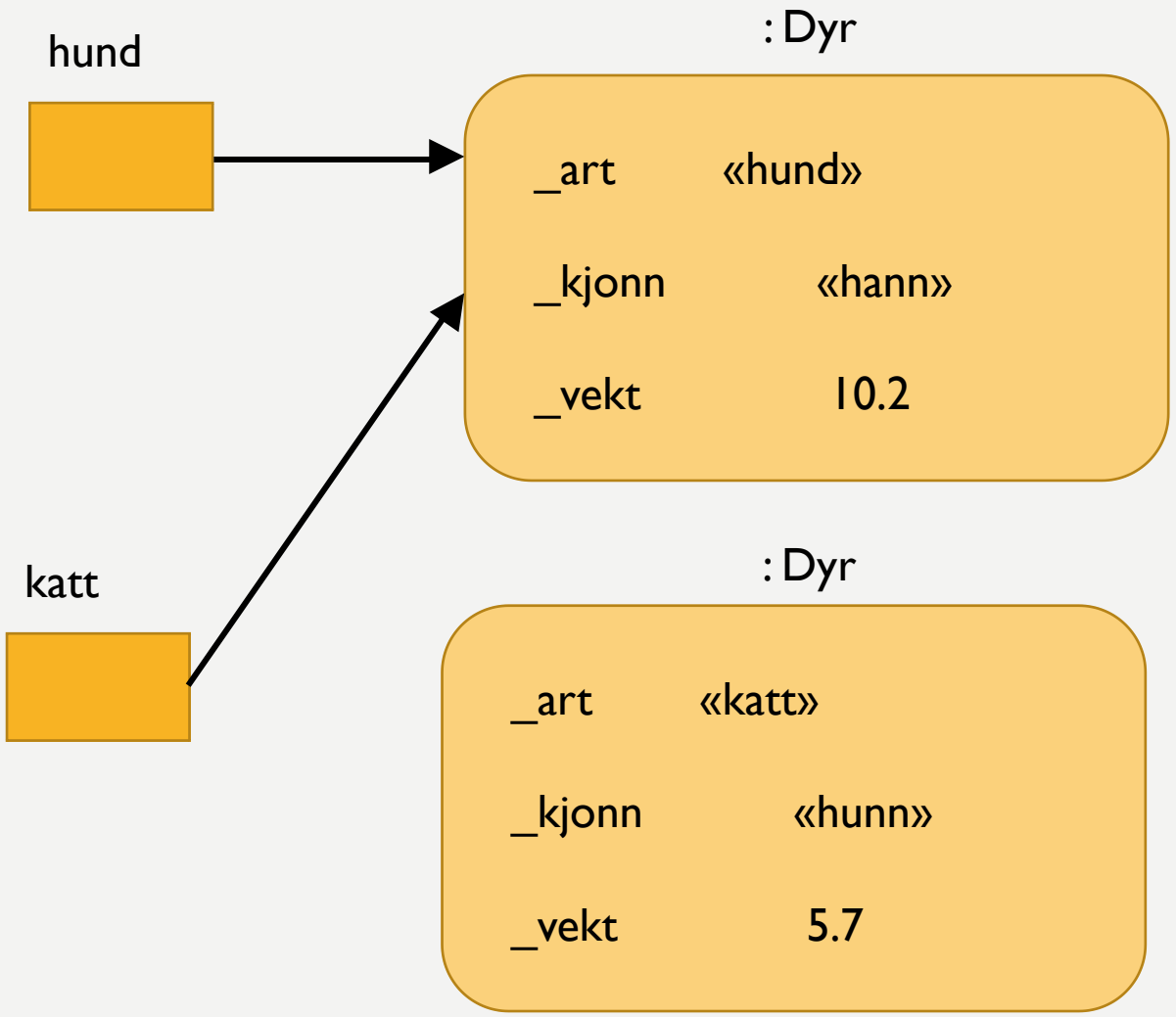


REFERANSER

Eksempel

```
hund = Dyr(«hund», «hann», 10.2)
katt = Dyr(«katt», «hunn», 5.7)
```

- «hund» er en referanse til et objekt av typen dyr
- Hva skjer om referansen (pilen) endrer seg dersom man f.eks. skriver:
  - katt = hund



EKSAMENSOPPGAVE

Hvor mange Ukedag-objekter eksisterer (kan man få tak i) etter at koden på bildet har kjørt?

```
1   class Ukedag:
2       def __init__(self, dag):
3           self.dag = dag
4
5   torsdag = Ukedag(4)
6   imorgen = Ukedag(4+1)
7   fredag = imorgen
8   helg = Ukedag(6)
9   fredag = None
10  helg = None
11  lordag = helg
12
```

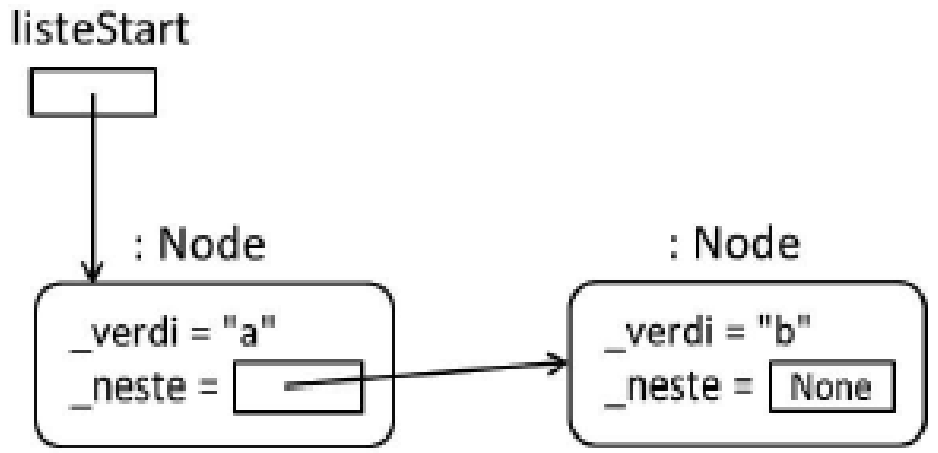
EKSAMENSOPPGAVE

Hvor mange bil-objekter eksisterer (kan man få tak i) etter at koden på bildet har kjørt?

```
1   class Bil:
2       def __init__(self, merke):
3           self._merke = merke
4
5   opel = Bil("Opel")
6   toyota = Bil("Toyota")
7   jaguar = Bil("Jaguar")
8   jaguar = opel
9   opel = toyota
10  toyota = None
11  elbil = opel
12
```

EKSAMENSOPPGAVE

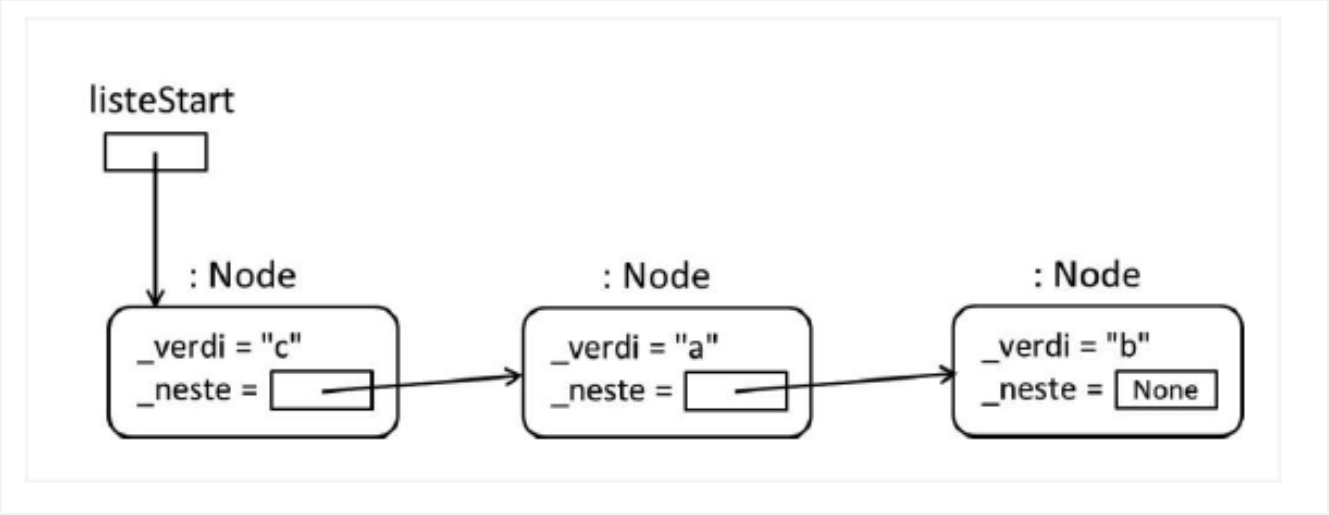
```
class Node :
    def __init__ (self, verdi) :
        self._verdi = verdi
        self._neste = None

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

    # ...flere metoder vi ikke trenger..
```



Gitt klassen **Node** som vist, skriv en prosedyre som oppretter 2 objekter med verdiene "a" og "b" i en struktur som vist i figuren. I figuren er variabler vist som en firkantet boks, piler angir objektreferanser, og objektene er vist som rektangler med avrundede hjørne.

Variabelen listeStart refererer til objektet med verdi "a". Du kan anta at klassen **Node** er importert til programmet ditt.

EKSAMENSOPPGAVE



Utvid **hovedprogrammet** fra oppgave e) med kode som oppretter et nytt objekt med verdi «c» og og oppdaterer referanser slik at du får en datastruktur som vist i vedlagte figur.

Klassen **Node** er uendret, og objekter av klassen skal kun aksesseres ved hjelp av de oppgitte metodene.