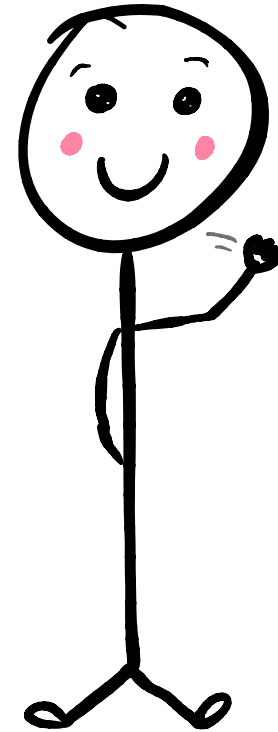


OBJEKTORIENTERT PROGRAMMERING

Klasser, objekter og referanser

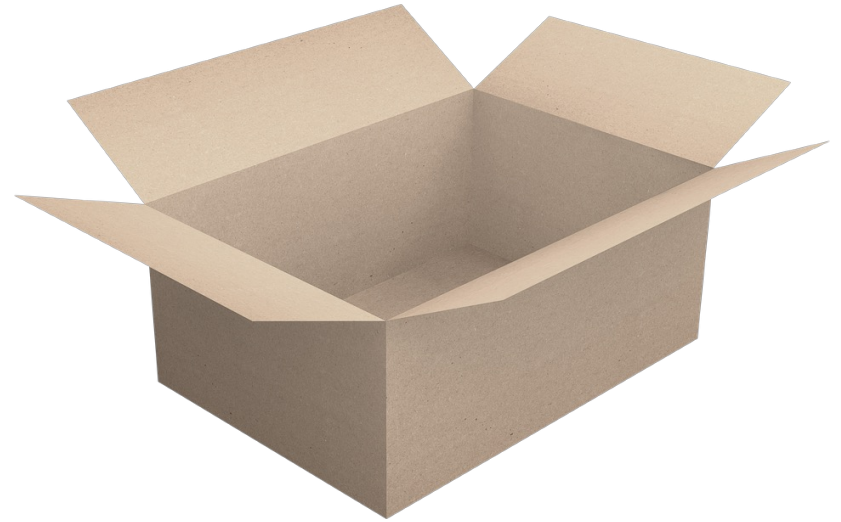
Repetisjonstime – IN1000



DAGENS PLAN – OOP

1. Objektorientert programmering
2. Klasser og objekter
3. Konstruktør
4. Metoder
5. Referanser
 1. Self
6. Datastrukturer
 1. Tegning :)
7. Magiske metoder
 1. Sammenlikning: `is` vs. `==`

(med eksempler fra [prøveeksamen 2022](#))

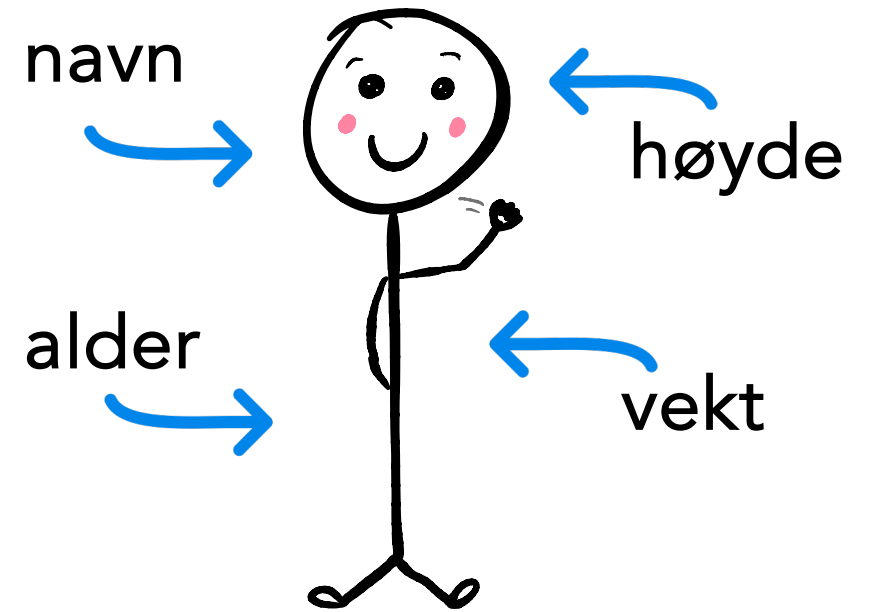


OBJEKTORIENTERT PROGRAMMERING (OOP)

- *Modellerer verden*
 - Forenkle virkeligheten gjennom *gruppering*
- En **klasse** beskriver objekter med lik oppførsel
 - F.eks. *hva kjennetegner en person?*

OBJEKTORIENTERT PROGRAMMERING (OOP)

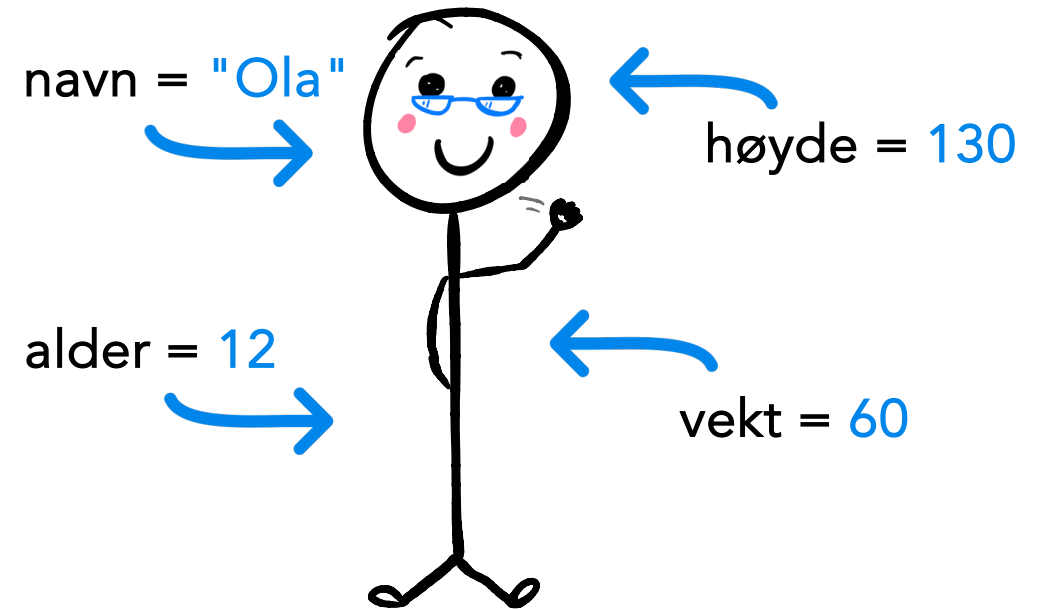
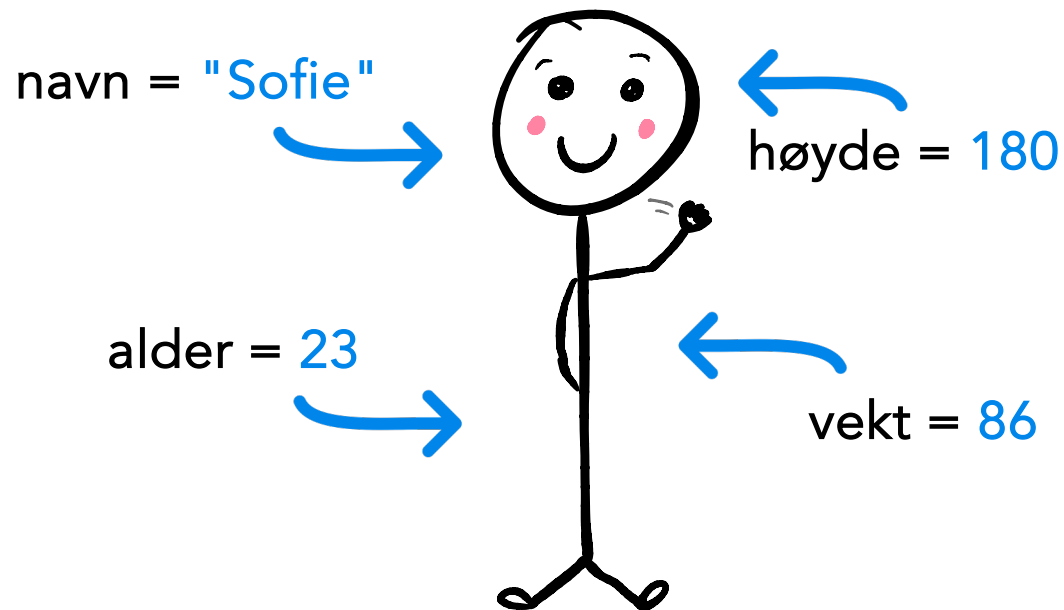
- *Modellerer verden*
 - Forenkle virkeligheten gjennom *gruppering*
- En **klasse** beskriver objekter med lik oppførsel
 - F.eks. *hva kjennetegner en person?*



(...og mye mer)

OBJEKTORIENTERT PROGRAMMERING (OOP)

- Kan lage instanser av klassen Person (=objekter)



DEFINERE EN KLASSE

- For å definere en klasse:
`class`

DEFINERE EN KLASSE

- For å definere en klasse:
`class`

```
# Definere en klasse Klasse
class Klasse:
    ...
```

DEFINERE EN KLASSE

- For å definere en klasse:
`class`
- **Konstruktøren** bestemmer hva som må være med ved opprettelsen av et objekt
 - Instansvariabler
 - `def __init__(self, ...)`

```
# Definere en klasse Klasse
class Klasse:
    ...
```

DEFINERE EN KLASSE

- For å definere en klasse:
`class`
- **Konstruktøren** bestemmer hva som må være med ved opprettelsen av et objekt
 - Instansvariabler
 - `def __init__(self, ...)`

```
# Definere en klasse Klasse
class Klasse:
    def __init__(self, <evt. parametre>):
        self.verdi1 = ...
```

DEFINERE EN KLASSE

- For å definere en klasse:
`class`
- **Konstruktøren** bestemmer hva som må være med ved opprettelsen av et objekt
 - Instansvariabler
 - `def __init__(self, ...)`
- Oppretter nytt objekt med:
`Klassenavn()`

```
# Definere en klasse Klasse
class Klasse:
    def __init__(self, <evt. parametre>):
        self.verdi1 = ...
```

DEFINERE EN KLASSE

- For å definere en klasse:
`class`
- **Konstruktøren** bestemmer hva som må være med ved opprettelsen av et objekt
 - Instansvariabler
 - `def __init__(self, ...)`
- Oppretter nytt objekt med:
`Klassenavn()`

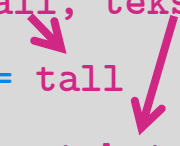
```
# Definere en klasse Klasse
class Klasse:
    def __init__(self, <evt. parametre>):
        self.verdi1 = ...

# Opprette et objekt av klassen Klasse
klasse1 = Klasse(<parametre>)
```

KONSTRUKTØREN: `__init__`

- Kjøres når man oppretter et objekt
 - **Rekkefølgen** på parametere og argumenter er viktig når man oppretter objektet
 - Ingen direkte sammenheng mellom navnene på parametere og **variabelnavn** i objektet

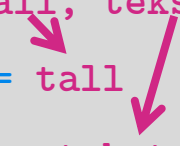
```
def __init__(self, tall, tekst):  
    self._tallverdi = tall  
    self._tekstverdi = tekst
```



KONSTRUKTØREN: `--INIT--`

- Kjøres når man oppretter et objekt
 - **Rekkefølgen** på parametere og argumenter er viktig når man oppretter objektet
 - Ingen direkte sammenheng mellom navnene på parametere og **variabelnavn** i objektet
- Definerer instansvariabler
 - Har vi en forhåndsdefinert instansvariabel trenger vi ikke ha denne med som parameter i konstruktøren

```
def __init__(self, tall, tekst):  
    self._tallverdi = tall  
    self._tekstverdi = tekst
```

A diagram with two red arrows. The first arrow starts at the parameter 'tall' in the function signature and points to the variable 'tall' in the assignment 'self._tallverdi = tall'. The second arrow starts at the parameter 'tekst' in the function signature and points to the variable 'tekst' in the assignment 'self._tekstverdi = tekst'.

```
def __init__(self, tekst):  
    self._tallverdi = 0  
    self._tekstverdi = tekst
```

DEFINERE METODER

- **Metoder** er funksjoner/prosedyrer som hører til en klasse
- For å definere en metode i en klasse: **def** i klassen
 - Ha alltid med **self** som parameter
- Kaller metoden **på** et objekt

```
# Definere en klasse Klasse med en metode
class Klasse:
    def __init__(self, <evt. parametre>):
        # Instansvariabler

    def ny_metode(self, <evt. parametre>):
        # Gjør noe

# Opprette et objekt av klassen Klasse
objekt1 = Klasse(<parametre>)

# Kalle på metoden ny_metode
objekt1.ny_metode(<parametre>)
```

GRENSESNITT OG INNKAPSLING

- Alle klasser har et **grensesnitt/public interface**
 - Hvilke metoder du kan bruke og hva de gjør

GRENSESNITT OG INNKAPSLING

- Alle klasser har et **grensesnitt**/public interface
 - Hvilke metoder du kan bruke og hva de gjør
- Vi vet ikke **hvordan** metodene er implementert → **innkapsling**
 - Kan kjøre en bil uten å vite hvordan motoren fungerer → kan bruke metoder uten å vite hvordan de er implementert

LASTE INN KLASSEN

- I *hovedprogrammet* eller *testprogrammet* vårt kan vi **laste inn** klassen slik:

```
from <filnavn> import <klasse>
```

LASTE INN KLASSEN

- I *hovedprogrammet* eller *testprogrammet* vårt kan vi **laste inn** klassen slik:

```
from <filnavn> import <klasse>
```

- For eksempel for **Person**-klassen:

```
from person import Person
```

↑
filen heter **person.py**

↑
klassen heter **Person**

OOP PÅ EKSAMEN

- Stor oppgave!

- Ofte *mange* poeng

- ...og mye tekst

- Få oversikt over hva du *faktisk* skal gjøre

- Les gjennom hele oppgaven

- Marker viktige stikkord

- Tegn opp ved behov

Del 4: Julegavefikseren: En kombinert ønskeliste og juleferiekalender

Totalt 50 poeng

Du kan anta i programmet du skriver at input (argumenter til metodene dine eller input fra fil/bruker) er lovlige verdier av riktig type, du trenger altså ikke sjekke disse om det ikke nevnes spesielt.

Dolly Duck har tenkt ut en løsning på to problemer: Hun er lei av å få julegaver hun ikke ønsker seg og som bare står og støver ned – og hun har ingenting å gjøre i juleferien etter julaften. Løsningen hennes er en app som du skal få utvikle – der hun kan registrere ønsker om opplevelser og aktiviteter i en ønskeliste, og de som vil gi henne gaver kan velge et ønske fra ønskelisten som så legges i en «juleferiekalender».

I juleferiekalenderen kan hun åpne en ny gave hver dag, fra og med 25. desember og så lenge juleferien varer. «Gaven» som ligger i kalenderen er bare en tekst som beskriver opplevelsen eller aktiviteten, sammen med navnet på giveren – men da vet hun jo hvem hun kan kontakte. En vanlig julekalender kan åpnes hver dag fra 1. til 24. desember – men her skal vi altså lage en kalender som brukes *etter* julaften i stedet.

Julegavefikseren er bygget opp av 5 klasser som du skal skrive:

- En klasse **Onskeliste** som inneholder ønskelisten (en liste med **Onske**-objekter) og metoder for å legge inn ønsker, velge et ønske å oppfylle, og for å hente ut en liste med ønsker.
- En klasse **Onske** med metoder for å sjekke om ønsket passer for en giver eller ikke, for å velge et ønske og for returnere ønsket som en string.
- En klasse **Juleferiekalender** med gaver for hver dag i juleferien, fra og med 25. desember og et valgfritt antall dager fremover (men aldri mer enn 31 dager). Klassen har metoder for å legge til en ny gave, hente gaven for en bestemt dag, og lese av hvor mange dager ferie juleferiekalenderen er laget for.
- En klasse **Gave** som inneholder beskrivelsen av gaven og hvem den er fra. Denne informasjonen kan hentes ut som en streng.
- Den siste klassen er **Julegavefikseren** som tar antall dager i juleferien som parameter til konstruktøren. Klassen har metoder for å lese inn Dollys ønsker i ønskelisten fra fil, for å velge et ønske å oppfylle og legge som en gave i juleferiekalenderen (for givere) og for å åpne en ny dag i kalenderen (for Dolly).

Du trenger ikke skrive noe hovedprogram – dette vil lese inn alle ønsker fra fil og ha et brukergrensesnitt der en som vil gi Dolly en gave oppgir navnet sitt og velger et ønske å oppfylle (og hvilken dag det skal gis), og der Dolly senere (når ferien starter) kan «åpne» en ny dag i kalenderen for hver dag i ferien.

Pass på å bruke metoder du allerede har skrevet i andre klasser der det er naturlig. Gave-objekter skal kun refereres til og brukes i Juleferiekalenderklassen, og **Onske**-objekter skal kun refereres til og brukes i **Onskeliste**-klassen. Instansvariabler skal aldri aksepteres direkte fra utsiden av klassen de hører til.

4a) 10 poeng. Skriv klassen **Onske**. Klassen har instansvariabler **beskrivelse**, **antall** (om hun gjerne vil ha ønsket oppfylt flere ganger) og **minimumspris** (den billigste måten å oppfylle ønsket på). Initialverdiene oppgis i parametere til konstruktøren. Klassen har metodene (som du skal skrive):

- **passer** med parameter som angir maksimumspris. Denne metoden sjekker om et ønske er for dyrt for giveren (minimumsprisen til gaven er høyere enn maksimumskostnaden giveren kan akseptere) eller om ønsket er «brukt opp» (dvs antall = 0)
- **valgt** som kalles når en giver har valgt ønsket. Denne oppdaterer antall og returnerer beskrivelsen
- **__str__** som returnerer beskrivelse og minimumspris

4b) 10 poeng. Skriv klassen **Onskeliste** med metoder (som du skal skrive):

- **nytt_onske** med parametere etter behov, oppretter et nytt ønske og legger det til ønskelisten
- **hent_onsker** som tar maksimumspris (det meste en giver har råd til) som parameter og returnerer en liste med tekststrenger, der hver streng representerer et ønske i ønskelisten. For *valgbare* ønsker (som kan oppfylles innenfor giverens maksimumspris, og som ikke er «brukt opp») skal strengen være en representasjon av objektet med beskrivelse og pris, mens for ønsker som ikke oppfyller kravene skal det bare stå «Ikke valgbart ønske».
- **onske_oppfylles** kalles når en giver har valgt hvilket ønske hen vil oppfylle. Metoden tar som parameter hvilket ønske som er valgt (et heltall), oppdaterer ønsket (at det er «brukt» en gang), og returnerer beskrivelsen av ønsket. Heltallet kan for eksempel være indeks til listen av ønsker.

4c) 5 poeng. Skriv klassen **Gave**. Konstruktøren tar parametere med initialverdier til to instansvariabler: **beskrivelse** og **giver** (hvem gaven er fra). Klassen har dessuten en metode som returnerer en streng med beskrivelse og giver (som du også skal skrive)

4d) 10 poeng. Skriv klassen **Juleferiekalender** med en konstruktør som tar antall dager i kalenderen som parameter og oppretter en ordbok der dagnummer er nøkkel og verdiene foreløpig er **None**. Dagnummer er et tall som angir datoen i desember eller januar: Fra 25 (siden kalenderen alltid starter 25. desember) til siste dag i ferien (juleferien varer max 31 dager dvs til og med dagnummer 24, men kan også være kortere enn dette). Tallet 28 representerer altså 28. desember, mens tallet 8 representerer 8. januar. Klassen har metodene (som du skal skrive):

- **ny_gave** som oppretter en ny Gave og legger den på en dag i juleferiekalenderen. Parametere er **beskrivelse** av gaven, **giver** (hvem gir gaven) og **dagen** den skal plasseres på (heltallet dagnummer). Du kan anta at den oppgitte dagen er gyldig og trenger ikke sjekke den (selv om dette skulle bety at du overskriver en eksisterende gave).
- **hent_dagens_gave** som returnerer en tekst som skal skrives ut når gaven på en bestemt dag (angitt av parameter dagnummer) åpnes. Teksten skal starte med datoen på formen dagnummer.månedsnavn (for eksempel 25. desember eller 1. januar). Hvis det ikke ligger noen gave i kalenderen for denne dagen skal teksten opplyse om det, ellers skal den inneholde beskrivelsen av gaven og hvem den er fra.
- **hent_ant_dager** returnerer antall dager i denne kalenderen

4e) 7 poeng. Skriv klassen **Julegavefikser** med en konstruktør som tar antall dager i kalenderen som parameter, og oppretter en Juleferiekalender og en Onskeliste. Konstruktøren skal også initialisere en instansvariabel **neste_dag** som holder rede på hvilken dag i julegavekalenderen som skal åpnes neste gang. Skriv også metoden:

- **les_onsker_fra_fil**. Metoden tar en parameter filnavn, og leser ett og ett ønske fra filen som har følgende format på hver linje:

beskrivelse; antall; minimumspris

For hver linje opprettes et ønske som legges til i ønskelisten.

4f) 8 poeng. Utvid klassen **Julegavefikser** med metodene (som du skal skrive):

- **velg_gave** leser inn maksimumspris fra en bruker som vil gi en gave, og skriver ut på terminalen en nummerert oversikt over alle ønskene tilpasset denne. Brukeren kan deretter velge et ønske å oppfylle ved å oppgi nummeret, dette registreres i ønskelisten som gitt én gang, og beskrivelsen tas vare på. Til slutt spørres brukeren om hvilken dag i kalenderen gaven skal «legges» i, og en ny gave som oppfyller det valgte ønsket legges der. Du trenger ikke sjekke om det allerede finnes en gave på angitt dag – det er greit at denne i så fall overskrives.
- **ny_dag** kalles når Dolly vil åpne neste dag i kalenderen. Metoden returnerer informasjon om gaven på denne dagen i kalenderen (om det er noen der) og oppdaterer instansvariabelen som holder rede på hvilken dag i kalenderen som skal åpnes neste gang. Husk at etter 31. desember (dagnummer 31) kommer 1. januar (dagnummer 1).

REFERANSER

- Vi bruker **referanser** for å få tak i objekter
 - En **referanse** er noe (f.eks. et navn) som peker på/henviser til en verdi eller et objekt i minnet



REFERANSER

- Vi bruker **referanser** for å få tak i objekter
 - En **referanse** er noe (f.eks. et navn) som peker på/henviser til en verdi eller et objekt i minnet
 - Den er ikke objektet, men *refererer* til det

hund1



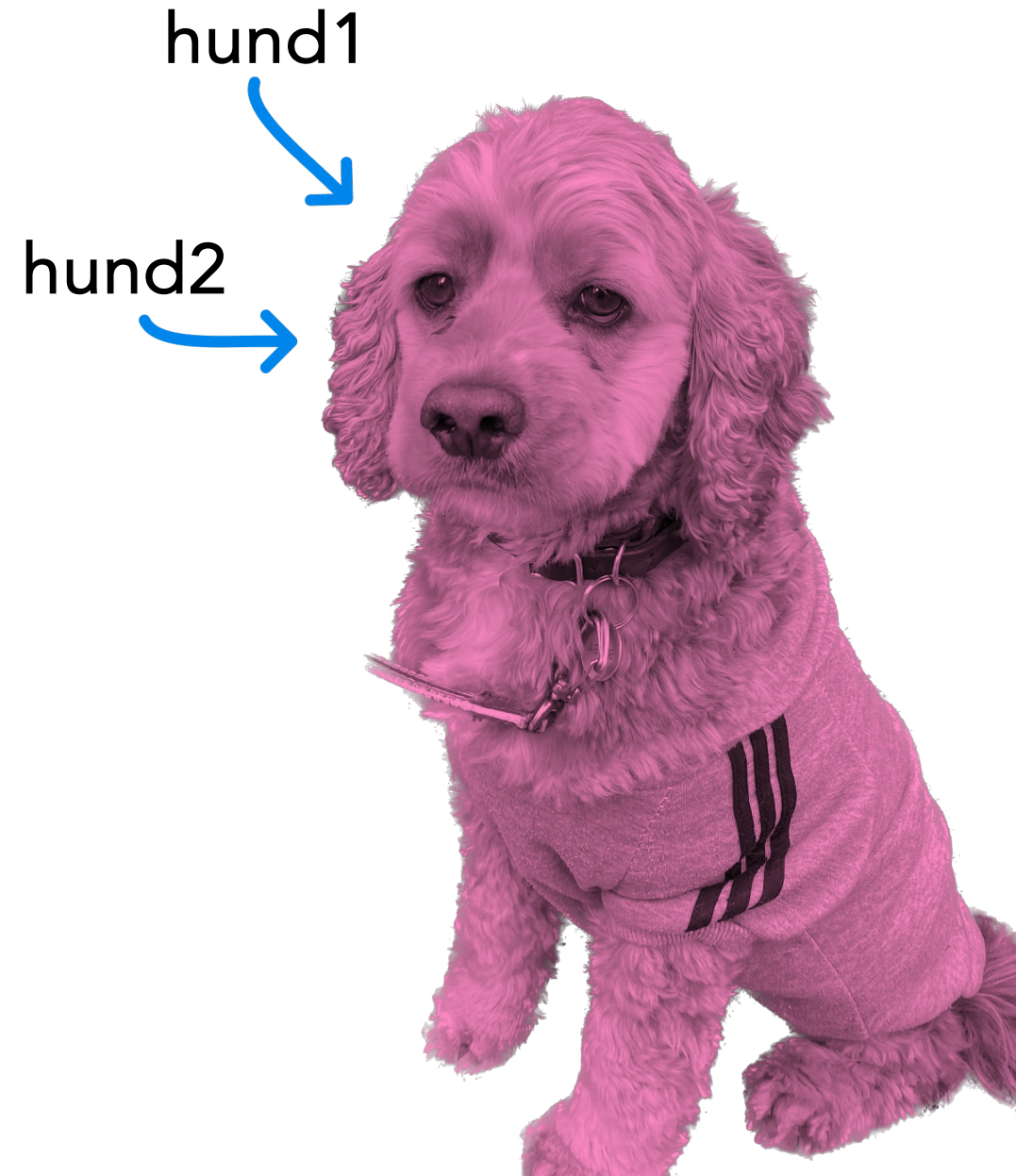
REFERANSER

- Vi bruker **referanser** for å få tak i objekter
 - En **referanse** er noe (f.eks. et navn) som peker på/henviser til en verdi eller et objekt i minnet
 - Den er ikke objektet, men *refererer* til det
 - Flere navn kan referere til samme objekt
 - F.eks. `hund2 = hund1`



REFERANSER

- Vi bruker **referanser** for å få tak i objekter
 - En **referanse** er noe (f.eks. et navn) som peker på/henviser til en verdi eller et objekt i minnet
 - Den er ikke objektet, men *refererer* til det
 - Flere navn kan referere til samme objekt
 - F.eks. `hund2 = hund1`
 - Endrer vi det refererte objektet vha. et av navnene, endrer vi det for alle
 - F.eks. farge objektet fra `hund2` rosa → objektet `hund1` refererer til blir også rosa!



SELF

- Brukes til å referere til **seg selv**
 - Altså «denne» instansen av klassen.
 - **Self** gir, som metode-parameter, tilgang til å endre objektet «**self**»
- Bruker vi ikke **self**, kan vi bl.a. ikke kalle metoden på objektet: `<objektnavn>.metode()`

REFERANSER – PÅ EKSAMEN

Oppgaver

1. «Hva skrives ut her?»

Hva blir resultatet av å kjøre denne koden?

```
a=1
b=2
c=3
listeA = [a,b,c]
listeB = [1,2,3]
listeC = listeA
listeA.append(5)
listeC.append(6)
print(listeC[4])
```

- ☐ 4 skrives ut
- ☐ 5 skrives ut
- ☐ IndexError: list index out of range
- ☐ 6 skrives ut

REFERANSER – PÅ EKSAMEN

Oppgaver

1. «Hva skrives ut her?»

Hva blir resultatet av å kjøre denne koden?

```
a=1
b=2
c=3
listeA = [a,b,c]
listeB = [1,2,3]
listeC = listeA
listeA.append(5)
listeC.append(6)
print(listeC[4])
```

- ☐ 4 skrives ut
- ☐ 5 skrives ut
- ☐ IndexError: list index out of range
- ☒ 6 skrives ut

DATASTRUKTURTEGNING

- Datastrukturtegninger gir oss oversikt over de ulike delene av programmet vårt
 - Bruk det som et verktøy

DATASTRUKTURTEGNING

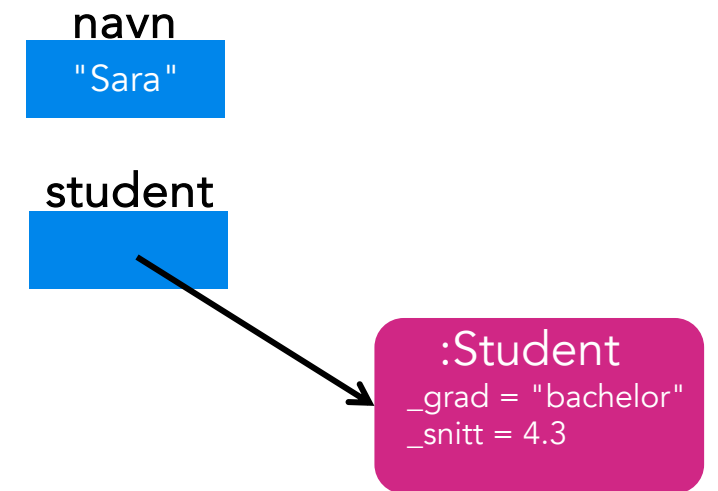
- Datastrukturtegninger gir oss oversikt over de ulike delene av programmet vårt
 - Bruk det som et verktøy
- Tegnetips:
 - **Variabler:** navngitt boks med verdi inni
 - **int**, **boolean** og **string** skrives vanligvis direkte inn i boksen

navn

"Sara"

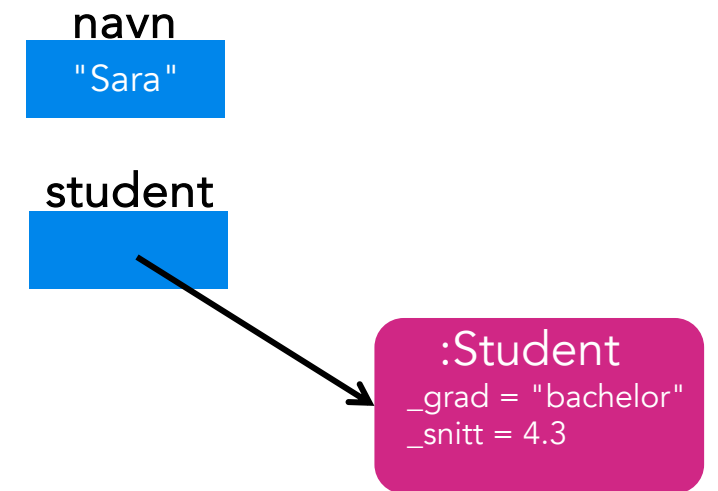
DATASTRUKTURTEGNING

- Datastrukturtegninger gir oss oversikt over de ulike delene av programmet
 - Bruk det som et verktøy
- Tegnetips:
 - **Variabler:** navngitt boks med verdi inni
 - `int`, `boolean` og `string` skrives vanligvis direkte inn i boksen
 - **Referanser:** piler fra referansevariabel til objekt
 - **Objekter:** rektangler med runde kanter
 - Ofte angir vi klassen/typen over objektet, med `:` foran klassenavnet
 - **Instansvariabler:** tegnes inni objektet



DATASTRUKTURTEGNING

- Datastrukturtegninger gir oss oversikt over de ulike delene av programmet
 - Bruk det som et verktøy
- Tegnetips:
 - **Variabler:** navngitt boks med verdi inni
 - `int`, `boolean` og `string` skrives vanligvis direkte inn i boksen
 - **Referanser:** piler fra referansevariabel til objekt
 - **Objekter:** rektangler med runde kanter
 - Ofte angir vi klassen/typen over objektet, med `:` foran klassenavnet
 - **Instansvariabler:** tegnes inni objektet



DATASTRUKTURTEGNING: EKSEMPEL

```
class Dyr:
    def __init__(self, art, kjonn, vekt):
        self._art = art
        self._kjonn = kjonn
        self._vekt = vekt

    def skriv_info(self):
        print("Art: ", self._art,
              ", kjonn: ", self._kjonn,
              ", vekt: ", self._vekt)

hund = Dyr("hund", "hann", 10.2)
```

hund



:Dyr

_vekt = 10.2

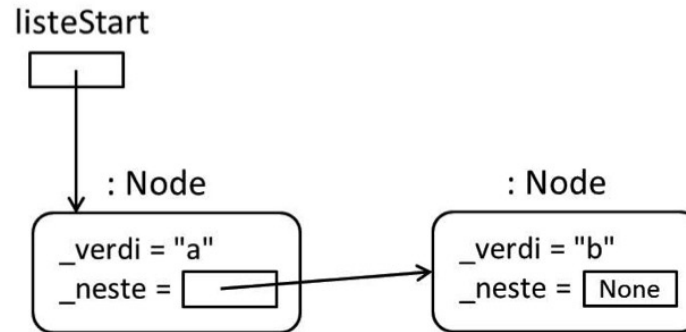
_art = "hund"

_kjonn = "hann"

DATASTRUKTURTEGNING PÅ EKSAMEN

- Kode til tegning

```
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.

[\(oppgave 3e, eksamen 2017\)](#)

MAGISKE METODER

- **Magiske metoder** er spesielle metoder i Python som vi kan *overlaste* for å bruke bestemte operatører
 - Markerres med **to** understreker før og etter metodenavnet
 - F.eks. `__init__`
 - Vi kaller aldri på magiske metoder direkte med metodenavnet
 - Altså ikke ~~`obj.__init__()`~~

MAGISKE METODER

- `__str__` : lar oss representere objektet som en streng
 - Kalles når vi bruker `str(obj)` og `print(obj)`
 - Vi definerer brukervennlig utskrift

MAGISKE METODER

- `__str__` : lar oss representere objektet som en streng
 - Kalles når vi bruker `str(obj)` og `print(obj)`
 - Vi definerer brukervennlig utskrift
- `__eq__` : lar oss sammenlikne objekter med `==`
 - Basert på det vi definerer til å være like objekter

EKSEMPEL: `__eq__`

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

EKSEMPEL: `__eq__`

```
class Rektangel :
    def __init__(self, lengde, bredde):
        self._lengde = lengde
        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)  # -> True
```

EKSEMPEL: `__eq__`

```
class Rektangel :
    def __init__(self, lengde, bredde):
        self._lengde = lengde
        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)  # -> True

r3 = Rektangel(6,4)
print (r2 == r3)  # -> False
```

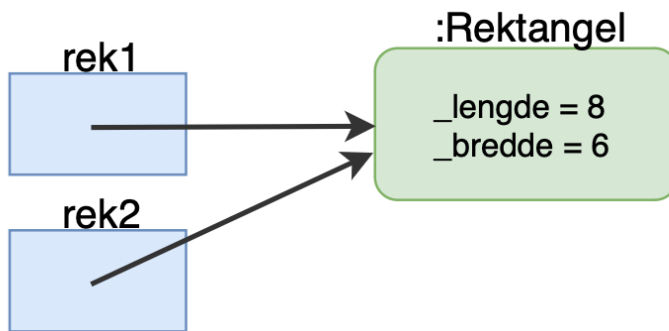
SAMMENLIKNING

- Vi kan sammenlikne med `is` og `==` :
 - `X is Y` : Er det **samme** objekt (lik adresse)?
 - `X == Y` : Er det **like** objekt (likt innhold)?

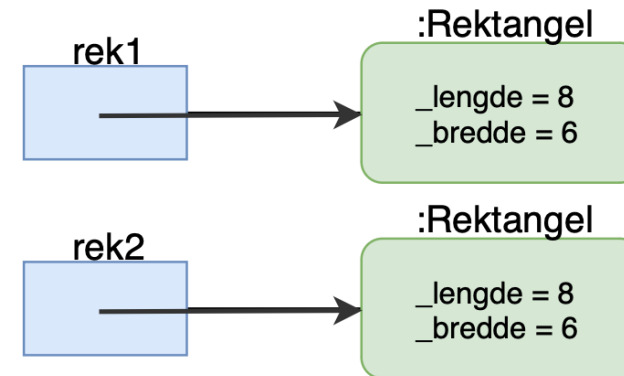
SAMMENLIKNING

- Vi kan sammenlikne med `is` og `==` :
 - `X is Y` : Er det **samme** objekt (lik adresse)?
 - `X == Y` : Er det **like** objekt (likt innhold)?

Samme objekt



Like objekt



MAGISKE METODER PÅ EKSAMEN

- Ofte del av klasseoppgaven
 - Eksamen 2022: `__str__` som returnerer beskrivelse og minimumspris
 - Eksamen 2017: `__str__` som returnerer en bruker-vennlig streng med hyttas navn, antall senger og pris per seng.
`__eq__` som sammenligner objektet den kalles på med et annet objekt (referansen er parameter til metoden). To **Hytte**-objekter er like hvis de har samme navn, da returneres **True**, ellers **False**.

LYKKE TIL 🎉