GRUPPETIME
UKE 9

IN1000 GRUPPE 4 - 21.03.22

LÆRDOMMER FRA OBLIG 6

- Husk underscore på instansvariabler
  - Dette viser at variablene er private og ikke synlige utenfor klassen.
- Pass på å utføre **ALT** en oppgave ber om - kjipt å miste poeng på eksamen fordi du har glemt å gjøre noe
- I hund.py skulle _metthet ikke tas inn som et parameter
  - self._metthet = 10
- `if dato_objekt.sjekkDag(15)`
  - ```
    def sjekkDag(self, sjekkDag):
        return self._dag == sjekkDag
    ```
    ```

PLAN FOR GRUPPETIMEN

- Jobbe sammen i Breakout Rooms
- Repetisjon
- Magiske metoder
- Oblig 7

LÆRINGSMÅL UKE 9

- Spesielle metoder for sammenligning og utskrift i egendefinerte klasser.
- Samlinger av objekter i beholdere (containers) som liste og ordbok.
- Strukturer med objekter av flere egendefinerte klasser.
- Repetisjon og (enda) mer detaljer om grensesnitt, innkapsling, hva skjer med referanser og objekter under kjøring

OPPGAVE 1

Hvilke verdier får variablene til instansene av objektene av type Person her, gitt opprettelsen av person slik:

```
test_person = Person(13, "Kari")
```

`def __init__(self, alder, navn):`
`  self._alder = 0`
`  self._navn = " "`
	`_alder = 0`
	`_navn = " "`

`def __init__(self, alder, navn):`
`  self._alder = navn`
`  self._navn = alder`
	`_alder = "Kari"`
	`_navn = 13`

`def __init__(self, alder, navn):`
`  self._alder = alder`
`  self._navn = navn`
	`_alder = 13`
	`_navn = "Kari"`

`def __init__(self, a, b):` `  self._alder = a`	`_alder = 13`

`def __init__(self, alder, navn):`
`  self._a = alder`
`  self._n = navn`
	`_a = 13`
	`_n = "Kari"`

OPPGAVE 2

LØSNING: OPPGAVE 2

```
class Bil:
     def __init__(self, regnr, type, aarsModell):
          self._regnr = regnr
          self._type = type
          self._aarsModell = aarsModell
```

```
bil1 = Bil("AB123", "Toyota", 1997)
bil2 = Bil("EL456", "BMW", 2016)
bil3 = Bil("DB5435", "Citroen", 2004)
```

LØSNING: OPPGAVE 2

```
class Koffert:
	def __init__(self, toalettsakerListe, klesListe):
		self._toalettsaker = toalettsakerListe
		self._klaer = klesListe
```

```
bag1 = Koffert([“tannborste”, “tannkrem”], [“jakke”, “genser”, bukse”])

toalettS = [“haarborste”, “neglsaks”, “tannborste”]
klaer = [“sokker”, “tskjorte”, “shorts”]
bag2 = Koffert(toalettS, klaer)

bag3 = Koffert(toalettS, [“badetoy”, “skjorte”, “jeans”])
```

LØSNING: OPPGAVE 2

```
class Hund:
     def __init__(self, a, b, c):
          self._alder = a
          self._rase = b
          self._bjefferMye = c
          self._metthet = 10

"""
dersom metthet skal alltid settes til 10 ved start trenger
ikke metthet aa vaere med som parameter i konstruktoeren.
"""
```

```
labrador = Hund(10, “Labrador”, False)
shiba = Hund(4, “Shiba Inu”, True)
dachs = Hund(7, “Dachs”, True)

"""
boolske verdier kan ogsaa brukes og vaere
nyttige variabler i objekter! Her burde
True/False brukes i stede for feks. “Ja”
eller “Nei” som verdi for variablen
._bjefferMye
"""
```

RETURN VS. PRINT

- Hvorfor kan det være nyttig med return?
  - Det er ikke alt man ønsker å skrive ut på skjermen
  - Ved print() så får man ikke tak i verdien (som vi kan bruke til andre ting senere!)
- return og print() er to helt forskjellige ting!
- Om du får beskjed i en oblig/eksamen om å “lage en funksjon/metode som bla… og returnerer svaret” → bruk **return**
- Hva er forskjellen på følgende kodesnutter?

```
def get_navn1():
    return self._navn
```

```
def get_navn2():
    print(self._navn)
```

```
def get_navn3():
    return print(self._navn)
```

return_vs_print.py

FINN FEILEN

```
def listeHarNegativtTall (liste):
    for tall in liste:
        if tall < 0:
            return True
        else:
            return False

liste = [-10, 4, 3, -9, 100]
print(f"listeHarNegativtTall på listen:  {liste} returnerer {listeHarNegativtTall (liste)}")

liste = [10, 4, 3, -9, 100]
print(f"listeHarNegativtTall på listen: {liste} returnerer {listeHarNegativtTall (liste)}")

liste = [10, 4, 3, 9, 100]
print(f"listeHarNegativtTall på listen:  {liste} returnerer {listeHarNegativtTall (liste)}")
```

```
def listeHarNegativtTall (liste):
    for tall in liste:
        if tall < 0:
            return True
        else:
            return False
```

logisk feil

```
def listeHarNegativtTall (liste):
    check = False
    for tall in liste:
        if tall < 0:
            check = True
    return check
```

Hva er forskjellen på disse to løsningene?

```
def listeHarNegativtTall (liste):
    for tall in liste:
        if tall < 0:
            return True
    return False
```

MAGISKE METODER: __eq__ & __str__

__eq__

- Kalles når vi bruker referansevar == referansevar
- Hvordan kan vi vite at to objekter er like?
  - Hvis de har samme verdi for instansvariabelen navn?

    Men hva om navn ikke er en passende instansvariabel?
    - Vi kan definere hva som skal til for at to objekter regner som like med __eq__

__str__

- Kalles når vi bruker
  - print(referansevar)
  - str(referansevar)
- Lager brukervennlig utskrift, lag den slik du ønsker
  - Det er opp til programmereren å lage gode print-setninger

hund.py

```
class Hund:
     def __init__(self, navn, kjonn alder):
          self._navn = navn
          self._kjonn = kjonn
          self._alder = alder

     def __str__(self):
          return self._navn

     def hent_streng(self):
          return self._navn

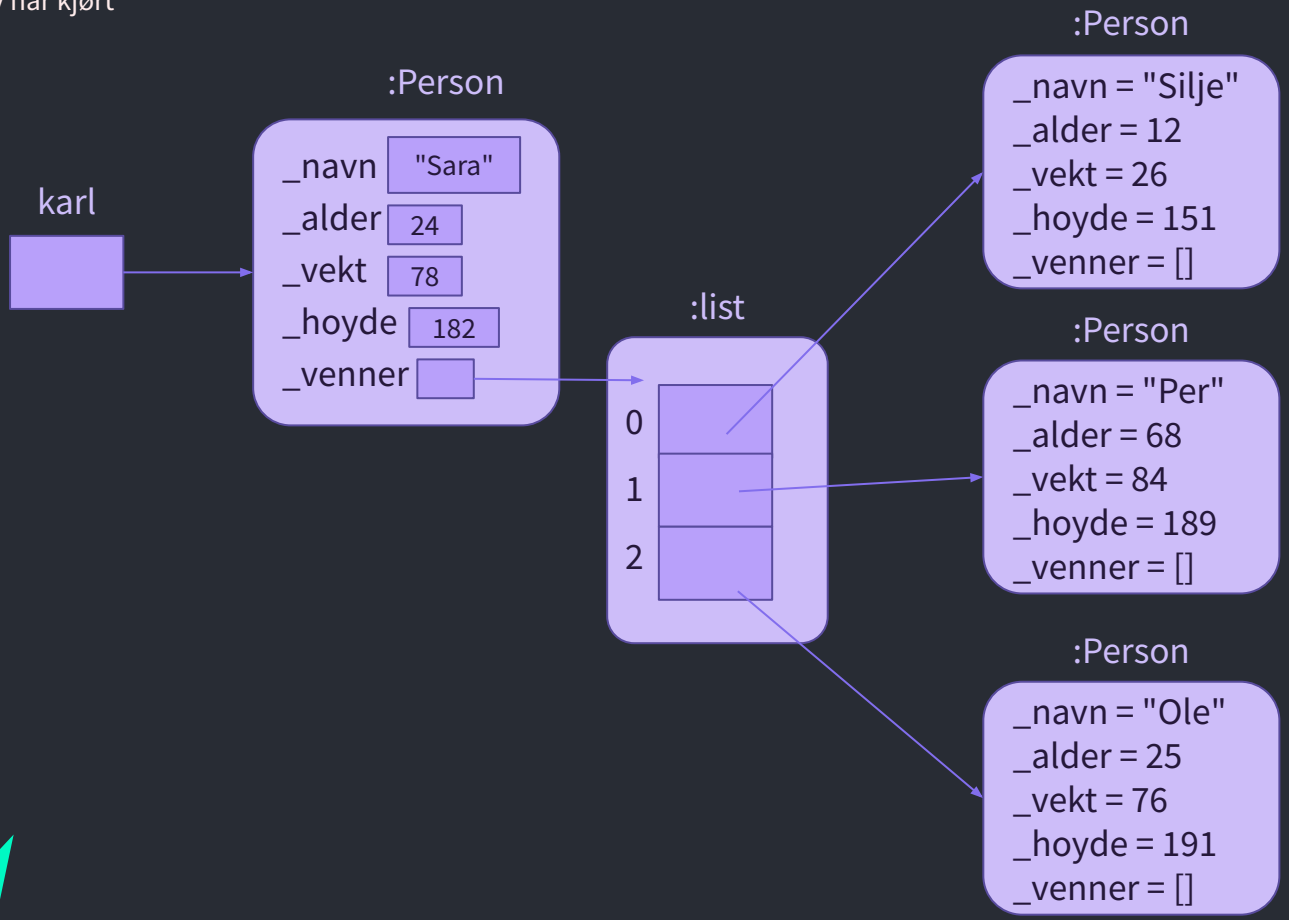

ny_hund = Hund("Ola", "M", 9)
print(ny_hund.hent_streng())
print(ny_hund)

if ny_hund.hent_streng() == "Ola":
    print("Ola Normann?")

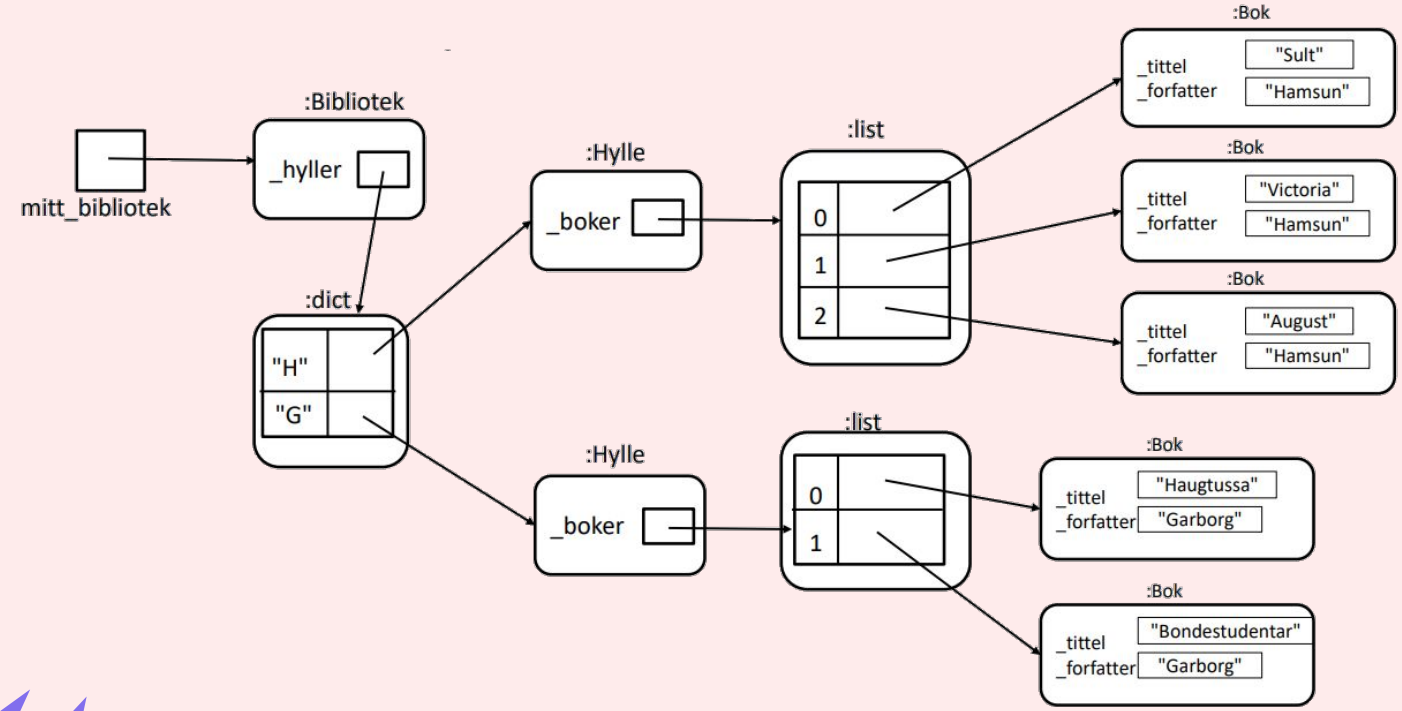
if str(ny_hund) == "Ola":
    print("Ola Normann?")
```

DATASTRUKTUR TEGNING EKSEMPLER

Datastruktur tegning til hvordan referansevariabelen karl ser ut etter kodelinje 1- 26 i test_person.py har kjørt



Bokhylle eksempel (uke 10)



KONTAKT



...

sirisoll@uio.no
@sirisoll på Mattermost