UKE 8 SEMINAR

REFERANSER

PLAN

- None
- Referanser

NONE

- Brukes til å representere fraværet av en verdi
- Alle metoder returnerer None som default når ikke noe annet er spesifisert
- Hvis man har en variabel som refererer til et objekt og ønsker at det ikke skal ha verdi så kan man sette variablene til «None»

NONE

Et eksempel hvor None er brukt:

(Dette er en del av et større program. Her har man en liste av student-objekter og ønsker å se om man finner studenten med det oppgitte navnet)

```
def finn_stud(self, navn):
    student = None
    for s in self._studentListe:
        if str(s) == navn:
            student = s
    return student
```

Denne metoden vil returnere et objekt av typen student (dersom den finner riktig student), eller returnere None hvis den ikke finner riktig student.

NONE

```
def finn_stud(self, navn):
        student = None
        for s in self._studentListe:
                if str(s) == navn:
                        student = s
        return student
```

```
def finn_stud(self, navn):
        for s in self._studentliste:
                if str(s) == navn:
                        return s
```

Hva er forskjellen mellom de to kodesnuttene?

None er også default verdien som returneres fra enhver metode, med mindre noe annet er definert. Med andre ord: lager vi en metode uten returverdi, så ser det ut som om den ikke returnerer noe, men i virkeligheten returnerer den «None».

NONE

Vi kan bruke None til å sjekke om metoden fant en student.
For eksempel:

```
student = finn_stud(navn)

if student is None:
    print(«Fant ikke studenten»)
else:
    # gjøre andre ting, f.eks. kalle på metoder fra student-objektet
    # print(«Fant student!»)
```

Hvis vi forsøker å gjøre f.eks. *student.skriv_info()* og student er None, vil programmet krasje.

SELF

Self brukes til å referere til objektet selv, altså «denne» instansen av objektet.

For eksempel:

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

        def lagt_paa_seg(self, kilo):
                self._vekt += kilo

def hovedprogram():
        hund = Dyr(«hund», «hann», 10.2)
        hund.lagt_paa_seg(0.3)

        katt = Dyr(«katt», «hunn», 5.7)
        katt.lagt_paa_seg(0.5)
```

OPPGAVE 1

Skriv en klasse sirkel. En sirkel har en radius, lag en konstruktør som setter radius.

Lag tre metoder til:

- en som returnerer diameteren til sirkelen
- en som returnerer omkretsen til sirkelen
- en som returnerer arealet til sirkelen.

*Hint: omkrets av en sirkel er diameter * pi, arealet av en sirkel er radius² * pi.*
*Eksponenter skrives som <base>**<eksponent>, f.eks. 2^8 skrives 2 **8.*

Lag deretter 2 sirkler med ulik radius.
Skriv ut den ene sirkelens omkrets og areal, og den andre sirkelens diameter.

OPPGAVE 1 - LØSNINGSFORSLAG

```
# import math (kun hvis man vil bruke math.pi)

class Sirkel:
def __init__(self, radius) :
      self._radius = radius

def diameter(self) :
      return self._radius*2

def omkrets(self) :
      return self.diameter()* 3.14
    # eller (ikke pensum)
        # return self.diameter()* math.pi

def areal(self) :
      return (self._radius ** 2) * 3.14

I hovedprogrammet:
from sirkel import Sirkel

sirkel1 = Sirkel(2)
sirkel2 = Sirkel(4.5)
print("sirkel1, omkrets:", sirkel1.omrkets(),"areal:", sirkel1.areal())
print("sirkel2, diameter:",sirkel2.diameter())
```

REFERANSER

- Objekter lagres ikke direkte i variabler – variablene inneholder i stedet **referansen** (minneadressen) til objektet
- Referansen/Minneadressen – innholdet i variabelen – er en *objektreferanse* (*referanse)*
- Variabler som holder rede på objekter kalles referansevariabler
- Referansevariabler kan brukes for å kalle på metoder i objektet

DATASTRUKTUR OG KONTROLLFLYT

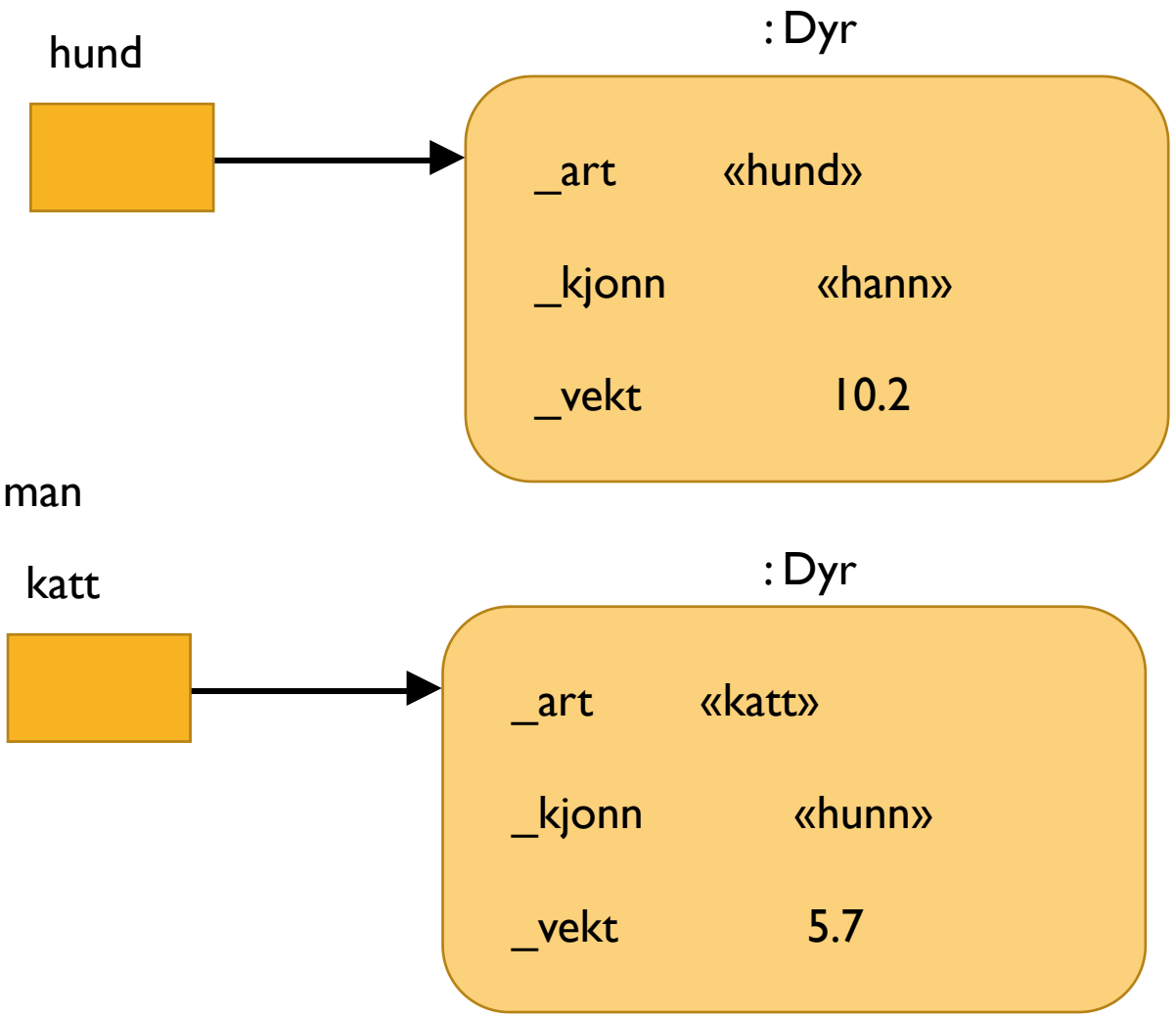
Eksempel

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

- Referanser er en måte å få tak i objekter på
- «hund» er en referanse til et objekt av typen dyr

Hva skjer om referansen (pilen) endrer seg dersom man f.eks. skriver

katt = hund



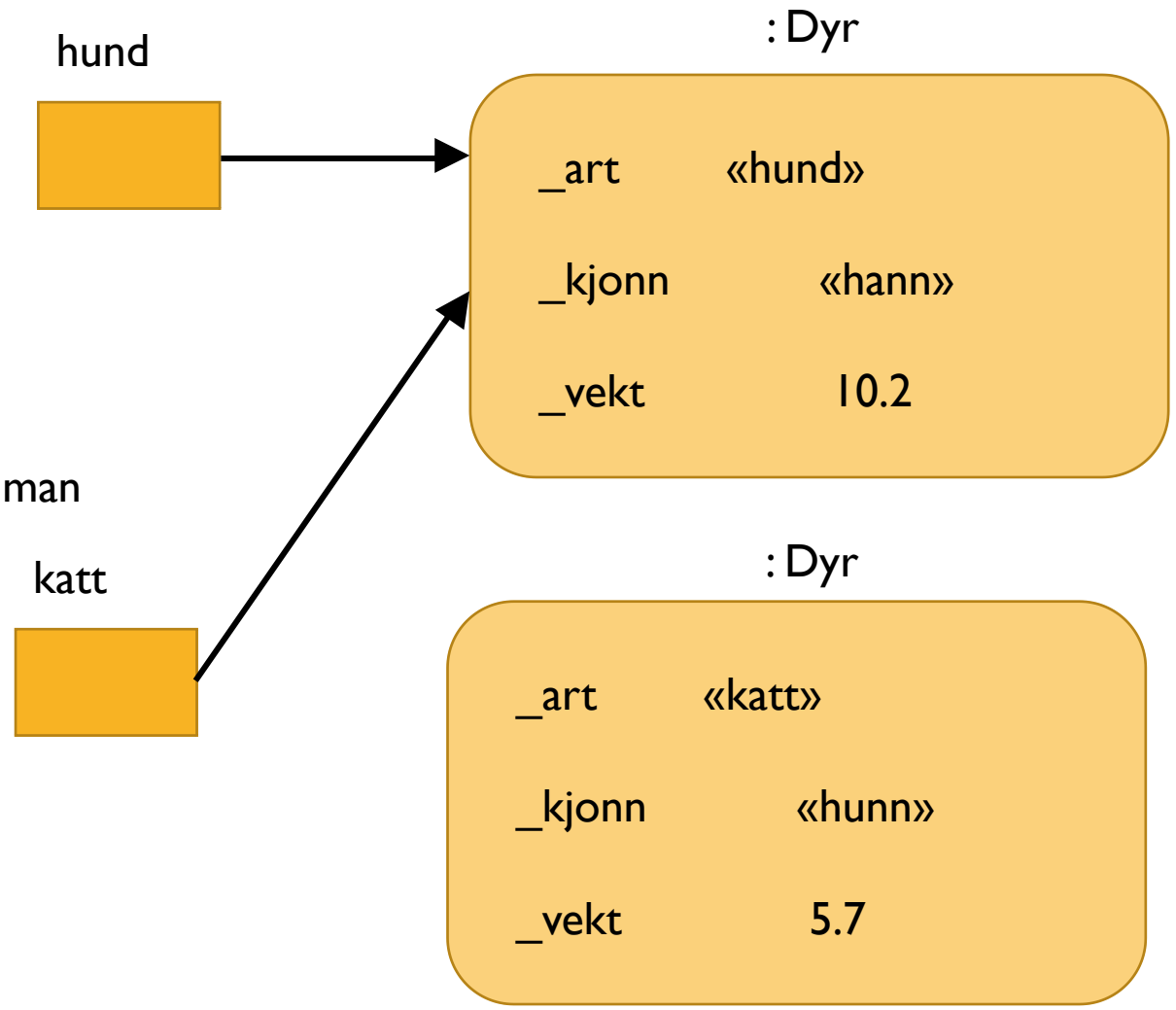
DATASTRUKTUR OG KONTROLLFLYT

Eksempel

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

- Referanser er en måte å få tak i objekter på
- «hund» er en referanse til et objekt av typen dyr

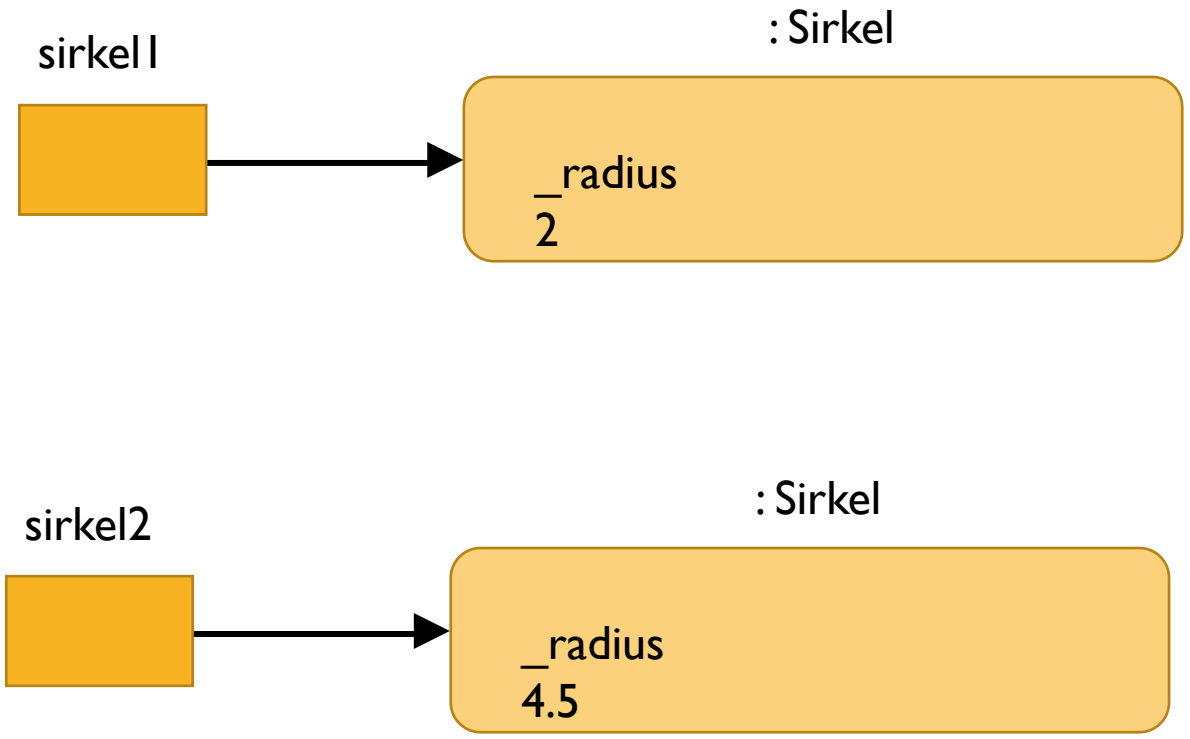
Hva skjer om referansen (pilen) endrer seg dersom man f.eks. skriver
katt = hund



OPPGAVE 2

Tegn datastruktur tegninger for oppgave 1.

```
class Sirkel:
def __init__(self, radius) :
      self._radius = radius

def diameter(self) :
      return self._radius*2

def omkrets(self) :
      return self.diameter()* 3.14

def areal(self) :
      return (self._radius ** 2) * 3.14

I hovedprogrammet:

sirkel1 = Sirkel(2)
sirkel2 = Sirkel(4.5)
```



OPPGAVE 3

Lag en klasse Blomst. En blomst har en art, en høyde, en instansvariabel som sier om blomsten har det bra eller ikke (denne verdien er en boolean), og en verdi som forteller hvor lenge siden blomsten ble vannet sist. Konstruktøren setter alle disse verdiene, når en blomst blir opprettet vil blomsten alltid ha det bra og det vil være 0 dager siden den ble vannet sist.

En blomst har en metode hentStatus som returnerer verdien som forteller om blomsten har det bra eller ikke.

I tillegg har den metoden skrivUtInfo, som skriver ut en info streng om blomsten

Videre har en blomst en metode nesteDag, som øker antall dager siden den ble vannet med en. Hvis det er mer enn tre dager siden blomsten ble vannet sist vil statusen til blomsten være at den har det dårlig. Hvis statusen til blomsten er at den har det bra vokser blomsten 1 cm.

Den siste metoden en blomst har er metoden vann. Hvis det er mindre enn 3 dager siden blomsten ble vannet vil statusen til blomsten bli dårlig (over vanning), ellers vil statusen være bra.

Skriv denne klassen + lag et testprogram for å teste klassen din. I testprogrammet skal du ha en variabel blomsterbed, som lagrer alle blomstene i en liste.

OPPGAVE 3 – HVORDAN STARTE

- Det er lurt å lese gjennom hele oppgaveteksten først.
- Tips: Noter ned variabler og metoder mens du leser, men uten å implementere dem. Da får man lettere oversikt over hva som skal gjøres.
- Eks:
  - class Blomst
  - Instansvariabler:
    - Art
    - Høyde
    - har_det_bra (boolean)
    - dager_siden_vanning
  - Instansmetoder:
    - def __init__(self, art, hoyde)
    - def hentStatus(self)
    - def skrivUtInfo(self)
    - nesteDag(self)
    - def vann(self)

OPPGAVE 3 - LØSNINGSFORSLAG

```
class Blomst:

    def __init__(self, art, hoyde):
        self._art = art
        self._hoyde = hoyde
        self._status = True
        self._dagerSidenVanning = 0

    def hentStatus(self):
        return self._status

    def skrivUtInfo(self):
        string = "Art: " + self._art
        string += "\nHoyde: " + str(self._hoyde)
        if self._status:
            string += "\nBlomsten har det: Bra"
        else:
            string += "\nBlomsten har det: Dårlig"
        string += "\nDager siden vanning: " + str(self._dagerSidenVanning)
```

```
# Fortsettelse av programmet til venstre

def nesteDag(self):
        self._dagerSidenVanning += 1
        if self._dagerSidenVanning > 3:
            self._status = False
        if self._status:
            self._hoyde += 1

    def vann(self):
        if self._dagerSidenVanning < 3:
            self._status = False
        else:
            self._status = True
        self._dagerSidenVanning = 0
```