Introduksjon til objektorientert programmering

INF1001 Høst 2016
Uke 7

Læringsmål uke 7

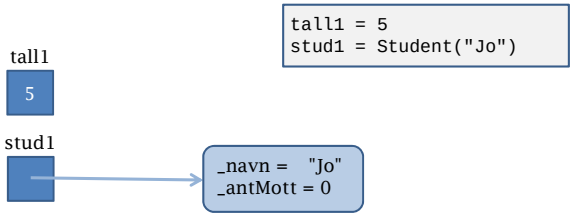
- Forstå (mer av) hva som skjer bak kulissene når vi oppretter og bruker objekter
- Kunne manipulere referanser og vite hvordan self og None brukes
- Kunne implementere "magiske metoder" for likhet i egendefinerte klasser
- Kunne skrive egne programmer med flere klasser og med instansvariable som refererer andre objekter
- Om tid: Demonstrere et verktøy som visualiserer hva som foregår i minnet ved utførelse av Python-programmer
  http://www.pythontutor.com/

Paradigmet objektorientering

- INF1001:
  Introduksjon til objektorientert programmering
- Python er verktøyet vi bruker
- Støtter, men krever ikke OO

Objektreferanser

- Objekter lagres ikke direkte i variable:

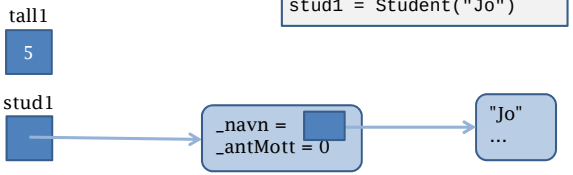


- Variabelen `stud1` inneholder en minne-adresse dvs den **refererer til** objektet (der dataene er lagret)

Objektreferanser

- Objekter lagres ikke direkte i variable:

```
tall1 = 5
stud1 = Student("Jo")
```

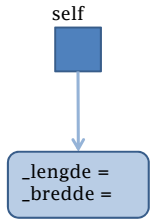


- Variabelen `stud1` inneholder en minne-adresse dvs den **refererer til** objektet (der dataene er lagret)

Objektreferansen `self`

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

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



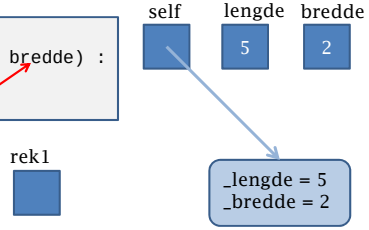
- Inne i klassemetodene brukes referansen `self` om objektet metodene er kallt på

Hva skjer når vi oppretter et objekt?

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

```
rek1 = Rektangel(5,2)

rek2 = Rektangel (8,3)
```

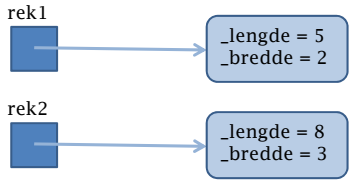


Hva skjer når vi oppretter et objekt?

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

```
rek1 = Rektangel(5,2)

rek2 = Rektangel (8,3)
```



Hva skjer når vi oppretter et objekt?

```
stud1 = Student("Petter")
```

- Konstruktøren i klassen Student kalles automatisk når et nytt objekt av klassen opprettes
- Konstruktøren
  - oppretter objektet med referanse `self`
  - oppretter og initierer instansvariablene i objektet
  - returnerer en referanse til objektet til kallstedet
- **Instansvariable** opprettes og initialiseres i hvert nye objekt

Hva skjer når vi kaller objekt-metoder?

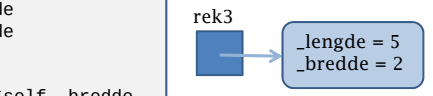
```
class Rektangel :
  def __init__(self, lengde, bredde) :
    self._lengde = lengde
    self._bredde = bredde

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

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

```
rek3 = Rektangel(5,2)
print (rek3.areal())

rek3.reduser(2,1)
print (rek3.areal())
```



Kjøring

```
M:> rektangel.py
10
3
M:>
```


Hva skjer når vi kaller objekt-metoder?

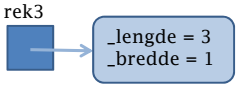
```
class Rektangel :
  def __init__(self, lengde, bredde) :
    self._lengde = lengde
    self._bredde = bredde

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

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

```
rek3 = Rektangel(5,2)
print (rek3.areal())

rek3.reduser(2,1)
print (rek3.areal())
```



Kjøring

```
M:> rektangel.py
10
3
M:>
```


Hva skjer når vi kaller objekt-metoder

- Objektet er endret!! Trenger ikke ta med noen returverdi for å spare på de nye verdiene, disse ligger i objektet til senere bruk!
- Returverdier brukes når vi skal bruke verdier eller resultater fra objektet på kallstedet
- Må se på dokumentasjonen av klassens grensesnitt for å vite hvordan kall på metoder virker

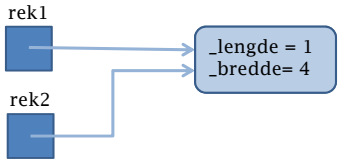
  eks: navn.upper() eller liste.append("Per")

NB for objektreferanser

- ved kopiering:

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

rek2.reduser(2,1)
```



- Når vi endrer rek2 gjelder endringene også for rek1!

NB for objektreferanser

- ved kopiering:

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

- ved sammenligning:

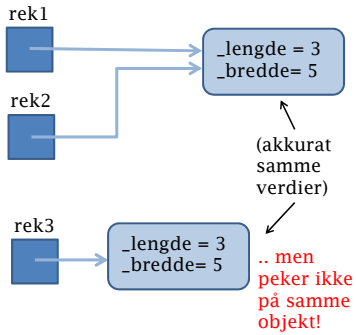
```
print(rek2 is rek1)
print(rek2 == rek1)
```

True

```
rek3 = Rektangel(3, 5)

print(rek3 == rek2)
```

False



Sammenligning

- For innebygde typer, String, integer, List osv: == som før
- For egendefinerte typer (klasser:
  - To referanser til samme objekt? Bruk **is**
  - Samme *innhold* i to objekter? Må sjekke instansvariablene
- Vi kan lage en egen metode i klassen som gjør dette. Hvis metoden får navnet __eq__ vil operatoren == sammenligne objekter av klassen slik vi implementerer det i metoden

eksempel på __eq__ metode

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

  def __eq__(self, annen) :
    return (self._lengde == annen._lengde and
            self._bredde == annen._bredde)

r1 = Rektangel(8,6)
r2 = r1
print (r1 == r2)

r3 = Rektangel(6,4)
print (r3 == r2)
```

```
M:> rekt2.py
True
False
M:>
```

None referansen

- En variabel som ikke refererer til noe objekt har verdien `None`
- `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

Familie og navn

- Skriv en klasse Navn som skal huske og presentere på flere formater navn bestående av fornavn, mellomnavn og etternavn

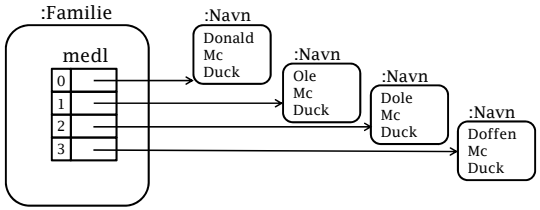
Grensesnitt for Navn

- Konstruktør med verdier for alle navn
- Metoder for å
  - hente sortert
  - hente naturlig

Klasse Familie

- Skal holde orden på familiemedlemmer
- Legg til nytt medlem
- Sorter alfabetisk
- Skriv ut alle i familien

Hvordan kan vi se for oss familien?



Klassen Navn (fra livekoding)

```
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
```


Klassen Familie (fra livekoding)

```
class Familie :
  def __init__(self) :
    self._medl =[]

  def leggTil(self,ny) :
    self._medl.append(ny)

  def skrivUt(self) :
    for nvn in self._medl :
      print(nvn.sortert())
```


Testprogram (fra livekoding)

```
siri = Navn("Siri", "Moe", "Jensen")
n1 = Navn ("a", "b", "c")

f1 = Familie()
f1.leggTil(siri)
f1.leggTil(n1)

f1.skrivUt()
```