Objektorientert programmering i Python

IN1000

Høst 2019 – uke 8

Siri Moe Jensen

Læringsmål uke 8

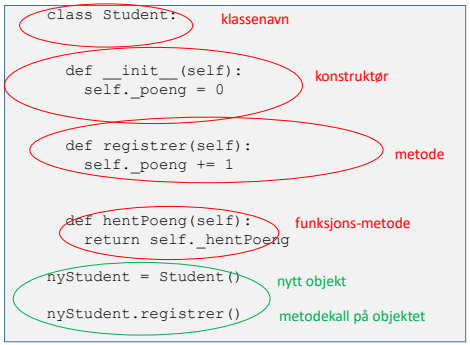
- Repetisjon fra forrige uke
  - Definere en klasse, opprette og arbeide med objekter: How-to
  - <Se ordliste på semestersiden for terminologi>
- Forstå (mer av) hva som skjer bak kulissene når vi oppretter og bruker objekter
- Kjenne til referansevariabler og hvordan de fungerer
- Kunne manipulere referanser og vite hvordan None brukes
- Kunne skrive programmer med flere objekter av samme klasse, sette seg inn i programmer med flere klasser

Obliger

- Godkjenning Oblig 1-6
  - Devilry vil være oppdatert ila neste uke
  - Kan gå gjennom egne innleveringer og sjekke poengsum (>=19)
- Både oblig 7 og oblig 8 må godkjennes for å gå opp til eksamen
- Se tekst og lenker på semestersiden (under Obligatoriske innleveringer) om krav til eget arbeid:

  Samarbeid om teori og andre oppgaver – skriv egen kode.
- Lever i god tid i tilfelle tekniske problemer eller annet. Hvis du leverer flere ganger er det siste versjon som gjelder.

Å definere en klasse, opprette objekter og kalle på metoder



..og hvordan opprette et objekt?

- Opprette objekter av egen klasse:

```
nyStudent = Student()
```

nyStudent

_antMott 0

- Noen innebygde klasser har snarveier for opprettelse av objekter (i stedet for klassenavnet)
  - "" oppretter en ny **str**
  - [] oppretter en ny **list**

Program med prosedyrer og funksjoner

mittprogram.py

```
def prosedyre (p1, p2, ..):
    …

def funksjon (p1, p2,…):
    …
    return …

def hovedprogram ():
    …
    # Kall på funksjon
    # Kall på prosedyre

hovedprogram()
```

parameter-verdier

retur-verdier

Program med klasser

mittklasseprogram.py

```
class Student:
    def __init__(self,p1):
        self._instvar1 = p1
        self._instVar2 = 3.14

    def metode1 (self, p2,..):
        …

def hovedprogram ():
    …
    # oppretter og bruker objekter

hovedprogram()
```

Kodeflyt og data

```
class Student :
  def __init__(self):
    self._poeng = 0

  def registrer(self):
    self._poeng += 1

  def hentPoeng(self):
    return self._poeng

nyStudent = Student()
```

Data

<husk klassenavn og metoder>

Kodeflyt og data

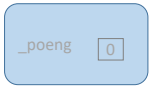
Programmet med kodelinjer

```
class Student :
  def __init__(self) :
    self._poeng = 0

  def registrer(self) :
    self._poeng += 1

  def hentPoeng(self) :
    return self._poeng

nyStudent = Student()
```

Data

<husk klassenavn og metoder>

nyStudent

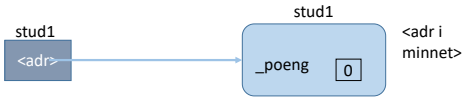


Representasjon av verdier i Python

- En variabel kan sees på som en "boks" som inneholder en verdi – et heltall, True eller False, eller et flyttall
- For variabler som representerer objekter trengs det et litt mer detaljert bilde
- Slike variabler inneholder ikke objektet selv – men en referanse til objektet

Referanser til objekter

- Objekter lagres ikke direkte i variable – variablene inneholder isteden bare referansen (minne-adressen) til objektet.



- Tegnes ofte som en pil for å indikere at objektet ikke selv ligger i variabelen – men at variabelen kan "vise vei".
- Minneadressen – innholdet i variabelen – er en *objektreferanse* eller bare *referanse*

Referanser til objekter

- Litt nærmere sannheten (men veldig tungvint på tegninger…) :

=> Vi kommer til å holde oss til denne!

Referanser til objekter

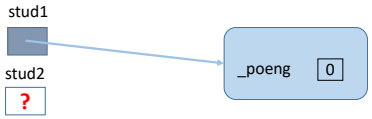
- Variabler som holder rede på objekter kalles *referansevariabler*
- Gjør det mulig å ta vare på og bruke objekter når vi trenger dem, akkurat som heltallsvariabler husker heltall til vi trenger dem.
- Selve objektet kan lagres "hvorsomhelst" i minnet, og være stort eller lite – referansevariabelen trenger bare plass til en adresse
- Referansevariabler kan brukes for å kalle på metoder i objektet:

```
refVariabel.metode()
```

Referanser til objekter

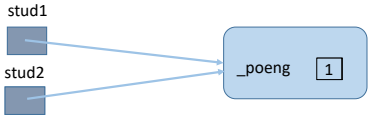
- Ofte trenger vi ikke tenke på at selve objektet ikke ligger i variabelen
- Men noen ganger er det viktig å ta hensyn til at *referansen* og *objektet* er to ulike ting!
- Hva skjer for eksempel om vi tilordner verdien fra en referansevariabel til en annen?

```
stud1 = Student()
stud2 = stud1
```



Referanser til objekter

```
stud1 = Student()
stud2 = stud1
stud2.registrer()
print (stud1.hentPoeng())
```



Situasjonen før rød pil utføres

=> Hva skrives ut i print-setningen?

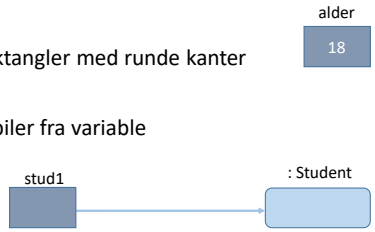
Noen objekter kan ikke endres

- Noen klasser har ikke metoder som endrer instansvariablene, de kalles immutable – "uforanderlige"
- Hindrer feil som følge av at flere variable refererer til samme objekt
- Streng-objekter er et eksempel på dette – alle metoder som endrer returnerer et NYTT objekt i stedet for å endre det eksisterende objektet
- Må se på dokumentasjonen av klassens grensesnitt for å vite

```
# Strenger er immutable, returnerer en endret kopi
navnMedSmaaBokstaver = navn.lower()

# Lister er mutable - objektet endres
liste.append("Per")
```

Kode og tegninger

- nyttig med tegninger når datastrukturene blir mer kompliserte etter hvert
- variable tegnes som en navngitt boks med verdi inni
- objekter tegnes som rektangler med runde kanter
- referanser tegnes som piler fra variable



Eksempel: Klassen Rektangel

- Rektangler er nyttige elementer i mange sammenhenger

=> Mulighet for gjenbruk

Skriv en klasse Rektangel som
- Har en konstruktør som tar to parametre: lengde og bredde
- En metode areal som returnerer rektangelets areal
- En metode endre som endrer størrelsen på rektangelet og har to parametere - lengde og bredde som angir hvor mye sidene i rektangelet skal endres med

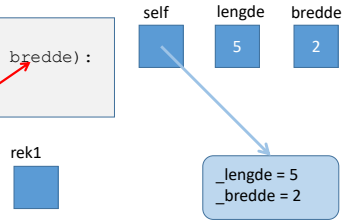
Klassen Rektangel

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

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

  def endre (self, lengde, bredde):
    self._lengde +=lengde
    self._bredde += bredde
```

Hva skjer når vi oppretter objekter?

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

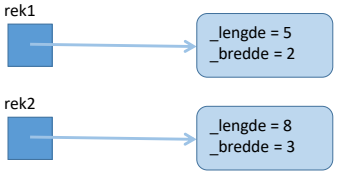
```
rek1 = Rektangel(5,2)

rek2 = Rektangel (8,3)
```



Hva skjer når vi oppretter objekter?

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

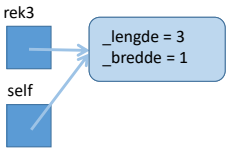
```
rek1 = Rektangel(5,2)

rek2 = Rektangel (8,3)
```



Hva skjer når vi kaller metoder?

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

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

  def endre(self,lengde,bredde):
     self._lengde +=lengde
     self._bredde += bredde
```

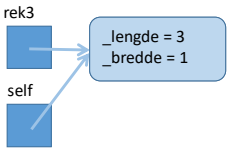
```
rek3 = Rektangel(5,2)
print (rek3.areal())

rek3.endre(-2,-1)
print (rek3.areal())
```



Kjøring

```
M:> python testrektangel.py
10
3
M:>
```

Metoder kan endre, lese og bruke egenskapene til objektet de kalles på




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

rek3.endre(-2,-1)

print (rek3.areal())
```

I metoden **areal** gjør objektet en beregning for oss som vi bruker på kallstedet

Metoden **endre** endrer egenskapene i objektet, som forblir endret etter metoden avsluttes

Nytt kall på **areal** gir en annen verdi fordi objektets egenskaper er endret

Lokale variabler i metoder

Hører ikke til objektet slik som instansvariablene

Oppstår og dør hver gang metoden kalles (som i funksjoner)

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

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

Verdien None

- Verdien None signaliserer at en referansevariabel ikke holder på noe objekt i øyeblikket.
- 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

Eksempel: Navn

- Skriv en klasse Navn som skal huske og presentere på flere formater navn bestående av fornavn, mellomnavn og etternavn
- Uformelt grensesnitt
  - Konstruktør m/ parametere for for-, mellom- og etternavn
  - Metoder for å
    - hente på form egnet for sortering (Hareide, Knut Arild)
    - hente naturlig (Knut Arild Hareide)

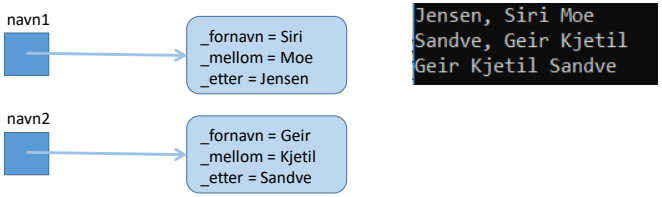
Klassen Navn implementert

Filen navn.py

```
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

  def naturlig(self):
    natNavn = self._fornavn + " "\
       + self._mellom + " " + self._etter
    return natNavn
```

Klassen Navn: Testprogram

```
from navn import Navn

navn1 = Navn("Siri", "Moe", "Jensen")
navn2 = Navn("Geir", "Kjetil", "Sandve")

print (navn1.sortert())
print (navn2.sortert())
print (navn2.naturlig())
```



Flere objekter av samme klasse

- En klasse er et mønster å lage objekter av
- Nyttig for å representere mange "ting" som følger samme mønster, for eksempel navn:

```
from navn import Navn

navneliste = []
les = input("Oppgi navn på naturlig form: ")
while les != "" :
    navnene = les.split()
    nytt = Navn(navnene[0],navnene[1],navnene[2])
    navneliste.append(nytt)
    les = input("Oppgi navn på naturlig form: ")

for navn in navneliste :
    print(navn.sortert())
```

Eksempel: Klasse Person

- Skal lagre og skrive ut data om en person
  - Navn
  - Alder
  - (og mye mer etter hvert)

```
class Person:
  def __init__(self,fulltNavn,alder):

  def skrivUt(self):
```

- Vil gjenbruke klassen vi har laget for navn

Implementasjon av Person

- Hvordan representere navnet til personen i klassen?
- Kan ha en instansvariabel som inneholder en tekst

  ... men da vet vi ingenting om hvordan navnet er bygd opp

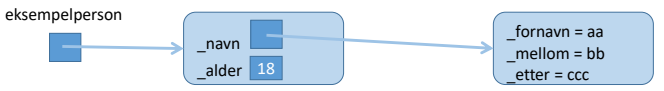
⇒ velger å bruke vår klasse Navn som en mer intelligent og fleksibel representasjon av personnavn

- Hvert Person-objekt får en instansvariabel som refererer et Navn-objekt
- Da kan et Person-objekt oppgi navnet sitt på flere former!

Implementasjon av Person

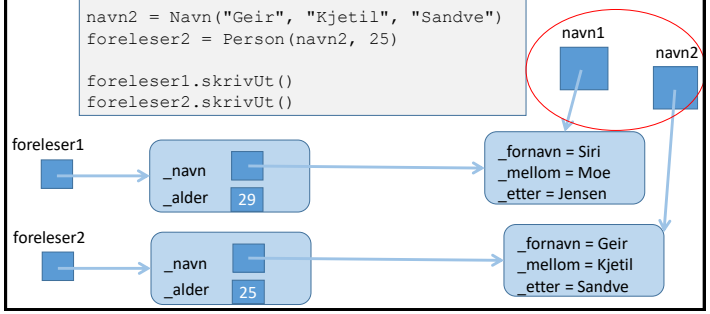
Filen person.py

```
class Person:
  def __init__(self,fulltNavn,alder):
    self._navn = fulltNavn
    self._alder = alder

  def skrivUt(self):
    print ("Navn: " + self._navn.naturlig() )
    print ("Alder: " + str(self._alder))
```



Klassen Person: Testprogram

```
from navn import Navn
from person import Person

navn1 = Navn("Siri", "Moe", "Jensen")
foreleser1 = Person(navn1, 29)

navn2 = Navn("Geir", "Kjetil", "Sandve")
foreleser2 = Person(navn2, 25)

foreleser1.skrivUt()
foreleser2.skrivUt()
```



Testprogram: Flere navn

```
from navn import Navn

navneliste = []
les = input("Oppgi navn på naturlig form: ")
while les != "" :
   navnene = les.split()
   nytt = Navn(navnene[0],navnene[1],navnene[2])
   navneliste.append(nytt)
   les = input("Oppgi navn på naturlig form: ")

for navn in navneliste :
   print(navn.sortert())
```

Testprogram: Flere personer

```
from navn import Navn
from person import Person

personliste = []
les = input("Oppgi navn på naturlig form: ")
while les != "" :
    navnene = les.split()
    nytt = Navn(navnene[0],navnene[1],navnene[2])
    alder = int(input("Oppgi alder: "))
    nyPerson = Person(nytt, alder)
    personliste.append(nyPerson)
    les = input("Oppgi navn på naturlig form: ")

for person in personliste :
    person.skrivUt()
    print("\n")
```

Neste uke

- "Magiske metoder" for egendefinerte klasser
  - Hvordan representere objekter som en streng?
  - Hvordan sjekke om to objekter er "like"?
- Objekter i ulike strukturer
  - Samlinger av objekter
  - Referanser mellom objekter
  - Referanser mellom objekter av ulike klasser