# Objektorientert programmering i Python: Introduksjon

IN1000

Høst 2019 – uke 7

Siri Moe Jensen

## Læringsmål uke 7

- Kjenne til motivasjon og bakgrunn for objektorientert programmering
- Kunne definere en klasse, opprette og bruke objekter
- Forstå sentrale begreper som grensesnitt og innkapsling
- Kjenne til utviklingsprosessen for en klasse gjennom design, implementasjon og testing

## Undervisning og pensum IN1000

- Innhold:
  - Prosedural programmering
  - Objektorientert programmering
  - Å ta det i bruk

Mer avanserte mekanismer og mer komplekse programmer =>
- Abstraksjon og designvalg
- Forenklinger og utelatelser

- Effektiv læring
  1. Få oversikt fra forelesningene
  2. Praktiser: Programmer og diskuter
  3. Slå opp teori, hvorfor virket dette (ikke)?

# Hvorfor objektorientert programmering?

## Bakgrunn

- Store og komplekse systemer etter ~20 år programmering med (noen av) de mekanismene dere har lært hittil.
- Hovedsakelig matematiske beregninger – komplekse, men formelt definerte - store tallmengder, men repetitive operasjoner.
- Kompleksitet ga høye kostnader og feil. Arbeidsdeling og gjenbruk ble vanskelig.

## Hva skjedde ~1967?

Samtidig:

- Datamaskinene ble kraftigere
- billigere
- mer utbredt

=> Nye anvendelser!

Forsvaret
Trafikk-planlegging

Økonomisystemer

Pasientjournal-systemer?
Facebook??

## En måte å strukturere programmer på

- *Prosedyrer og funksjoner* kan brukes for å løse deloppgaver
- Hva om deloppgavene handler om å bearbeide felles, kanskje komplekse, data?

Løsning:

- Samle relaterte data og kode for å manipulere dataene i **objekter**
- Objektene tilbyr resten av programmet et sett **metoder** grensesnitt
- Hvordan objektet representerer og manipulerer sine data er skjult innkapsling

Gjenstår: Hvilke data og operasjoner trengs? Hvordan skal de grupperes?

## Konseptet objektorientering

- Å programmere er å modellere
  - Vi bygger en egen representasjon av fenomener (konkrete eller abstrakte) vi trenger å manipulere i datamaskinen
  - denne representasjonen er ikke virkeligheten – men kan holde rede på noen aspekter som er viktige for det vi skal lage et program for

=> Vi lager en *modell av "virkeligheten"*

- Å programmere er å forstå
  - virkeligheten
  - det problemet/ behovet vi skal løse
  - hva som kan/ vil endre seg i fremtiden

# Å programmere med objekter

Objekter er *instanser* av en klasse

## Vi har tidligere brukt objekter som

- ..representerer lister m/ tilhørende metoder (tjenester)

```
navneliste = []
navneliste.append("Per")
navneliste.append("Paal")
navneliste.append("Espen")
navn = navneliste.pop()
```

- ..representerer filer m/ tilhørende metoder

```
utfil = open ("navn.txt", "w")
utfil.write (navn)
utfil.close()
```

## Metoder

- Vi kaller på metoder omtrent som prosedyrer og funksjoner
- MEN en metode må alltid kalles for ett objekt, derfor bruker vi "dot-notasjon"

```
minListe.append(nyttElem)
```

- Vi utfører metoder på objektet vårt ved hjelp av kallet *< objekt.metode(parametere) >*
- En metode kan ta argumenter
- En metode kan returnere en verdi (som en funksjon)

## Grensesnitt

- Et objekt tilbyr *ett sett tjenester* i form av metoder som programmerere kan kalle på for å lese av, endre eller aktivisere et objekt: *Objektets grensesnitt*
- Grensesnittet bestemmes av *objektets klasse*
- Hvordan finner vi ut hvilke metoder som tilbys av objekter av en bestemt klasse?
- F eks i læreboka eller Python dokumentasjon
  - google search: python methods List
  - f eks https://docs.python.org/3/tutorial/datastructures.html
    *(nb bl.a. ulike versjoner av Python)*

## Et objekt har..

- et **grensesnitt**
- og en innmat: **Implementasjon**

# Å lage egne klasser

Hva finnes bak grensesnittet?

## Klasser

- En klasse er et mønster/ oppskrift for objekter av samme type. Eks:
  - Student
  - Dato
- Klassedefinisjonen beskriver hvilke data hvert objekt skal ha (i *instansvariabler*) og de metodene som skal operere på instansvariablene i objektet
- Du kan definere (programmere) dine egne klasser som du så kan opprette en eller flere instanser (objekter) av

## Klasse-definisjon (syntaks)

```
class Student:

    def __init__(self):
        self._antMott = 0

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

    def hentOppmote(self):
        return self._antMott
```

klassenavn

konstruktør

metode

funksjons-metode

## Opprette objekter (syntaks)

- Noen innebygde klasser har snarveier for opprettelse av objekter
  - "" oppretter en ny **`str`**
  - [] oppretter en ny **`list`**
- Ellers opprettes objekter vha klassenavn med argumentliste (som kan være tom):

```
nyStudent = Student()
```

## Fremgangsmåte

Eksempel Student

## Eksempel Student

- Skal lagre og bearbeide informasjon om studenter
- Studenter har MANGE egenskaper – ulike systemer vil "se" studenter på ulike måter
  - Devilry
  - Eksamenssystemet
  - Lånekassen
- Hva skal jeg bruke klassen til?

  => Hjelp for gruppelærer:
  Holde rede på oppmøte i gruppetimene

## Student: Grensesnitt uformelt

- Ønsker å kunne gjøre følgende:
  - Lage Student-objekt med null oppmøte ved oppstart
  - Øke oppmøte med 1 ved tilstedeværelse
  - Lese av (hente ut) oppmøte ved behov
- Grensesnittet må tilby metoder som:
  - Registrerer oppmøte
  - Henter antall ganger møtt

## Student: Grensesnitt i Python

```
## Behandler en students oppmote på gruppetime
#
class Student:
  ## Oppretter student med oppmote 0
  #
  def __init__(self):

  ## Registrer oppmote
  #
  def registrer(self):

  ## Hent ut oppmote
  #
  def hentOppmote(self):
```

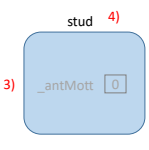
## Student: Implementasjon og bruk

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

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

  def hentOppmote(self):
    return self._antMott

stud1 = Student()
stud1.registrer()
antall = stud1.hentOppmote
```

## Bruk av klassen Student

- **Instansvariable** opprettes og initialiseres i __init__ metoden (konstruktøren)
- Konstruktøren kalles automatisk når et nytt objekt opprettes:
  1. oppretter objektet
  2. oppretter og initierer instansvariablene i objektet
  3. returnerer det nye objektet til kallstedet

```
class Student:
2) def __init__(self):
   3) self._antMott = 0

stud = Student() 1)
4)
```




## Oppgave

```
class Student:


  def __init__(self):
    self._antMott = 0


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


  def hentOppmote(self):
    return self._antMott
```

Utvid klassen student med studentens navn:

- instansvariabel _navn
- endre konstruktør: Ny parameter navn
- ny funksjonsmetode hentNavn

- Hvordan opprettes nå et nytt Student-objekt?

## Student-klasse utvidet med navn: Implementasjon i Python

```
<programfiler ligger under uke7 på semestersiden>
```

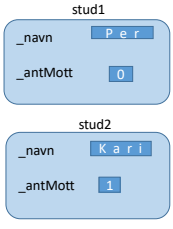
## Test av klassen

- Vi skriver et testprogram i en egen fil
- Importerer klassen Student (fra filen student.py)
- Oppretter (minst) et objekt
- Kaller på alle metoder (minst) en gang - ulike sekvenser kan være nødvendig for å avdekke feil siden objekter husker verdiene sine (har en tilstand)
- **`assert`** tvinger oss til å tenke ut på forhånd hva vi forventer

## Studentobjekter

- Vi har tidligere sett objektene vi bruker "fra utsiden", og nøyd oss med å bruke tjenestene deres
- Når vi lager egne klasser, bestemmer vi hvordan "innmaten" i objektene skal se ut.
- Kan være nyttig å tegne, spesielt når vi får mange objekter av forskjellige klasser

```
stud1 = Student("Per")
stud2 = Student("Kari")
stud2.registrer()
```




## Hvorfor lage en klasse?

- For å tilby et sett tjenester som hører sammen (verktøykasse)
- Spesielt nyttig for tjenester som trenger å dele data over tid (eks for håndtering av en liste, eller en fil, eller en tekst)
- For å kunne lage flere like objekter med samme data og oppførsel (eks representere studenter)
- For å simulere sammensatte fenomener vha objekter som representerer virkelige elementer (eks trafikk- simuleringer)

## Fremgangsmåte: OOP

1. Identifiser aktuelle klasser (hva er sentrale «ting»/ begreper programmet skal behandle?)
2. Design klassens grensesnitt (hvilke operasjoner trenger jeg for objekter av klassen?)
3. Design klassens datarepresentasjon (hvordan skal objektene representere dataene sine?)
4. Implementer (fyll ut) metodene (skriv klassen ferdig)

5. Lag et testprogram som oppretter et eller flere objekter og kaller på metodene i objektenes grensesnitt

## Innkapsling

- Prinsipp som innebærer at programmerere kun har tilgang til (utvalgte) metoder i objekter av en klasse: Klassens grensesnitt ("public interface")
- Data og evt andre metoder definert inne i klassen er "non-public" og brukes kun inne i klassedefinisjonen
- Sentralt prinsipp i objektorientert programmering
  - en klasse kan endres så lenge grensesnittet er det samme – uten at det får konsekvenser der klassen brukes
  - gjennomført ansvarsdeling (splitt og hersk)

## Innkapsling i Python og IN1000

- Python støtter innkapsling med *konvensjoner*, men har ikke mekanismer for å hindre tilgang
  - alle instansvariable og metoder som ikke skal inngå i grensesnittet, får navn som begynner med _
- Dette prinsippet skal dere bruke konsekvent i IN1000
- I praktisk bruk er det et *designvalg* om / når/ hvordan man anvender innkapsling
- Dere vil se eksempler på direkte aksess av data (utenom klassens metoder) blant annet i IN1010

## Et info-program om O-JD terminalrom

- Informasjon om rommene i O-JD er kopiert fra termvakt wiki til fil
- For hvert rom:
  - romnummer
  - romnavn
  - hva slags maskiner
  - hvor mange plasser
- Skal skrive et program som leser inn data om rommene, skriver ut en oversikt og etter hvert svarer på ulike spørsmål

## Uformelt grensesnitt

- leser alle data fra fil –> oppretter objekter etter hvert
- nye objekter opprettes med individuelle verdier for
  - romnummer
  - romnavn
  - hva slags maskiner
  - hvor mange plasser
- trenger metoder som
  - skriver ut en "pen" linje med info om rommet
  - sjekker om et rom tilfredsstiller krav til plass og op.sys.

## Eksempel Rom: Klassedefinisjon

```
class Rom :

  def __init__(self, nr, navn, opsys, ant) :
    self._nr = nr
    self._navn = navn
    self._type = opsys
    self._ant = ant


  def skrivLinje(self) :
    print ("%d %-15s %-15s %d" % (self._nr, self._navn,
                                  self._type, self._ant))
```

## OO støtter modularisering

- Både funksjoner og klasser er mekanismer som hjelper oss å strukturere koden vår gjennom modularisering
- Klassen definerer *data og operasjoner* som hører sammen, og kan definere et begrenset vindu (*grensesnitt*) som brukere av klassen (programmerere) skal få til objekter av klassen
- Klassedefinisjonen er mønsteret som blir fulgt når vi lager nye objekter av klassen: De har de samme instansvariablene, men instansvariablene kan ha ulike verdier
- Et objekt husker data i instansvariablene sine mellom hver gang en metode kalles for objektet

## Å definere en klasse, opprette objekter og kalle på metoder