

Vi starter 10:15. Imens, les gjerne gjennom mini-notat fra sist under ressurser på uke8-siden: [Hva er klasser og objekter](#)

Objektorientert programmering i Python

IN1000
Høst 2019 – uke 8
Siri Moe Jensen

Plan for i dag

Chat holdes åpen – husk at det er noen hundre deltakere

- Informasjon om innhentings-seminar og **obligatoriske** oppgaver
- Kort repetisjon
- Referanser og objekter
 - oppgave i breakoutrooms

Pause

- Svar på evt chat-spørsmål
- Eksempel – hva skjer i detalj når vi oppretter objekter og kaller metoder?
- Mengder av objekter
- (Forsmak på neste uke: Flere klasser, objekter som refererer til andre objekter)

Å lære objektorientert programmering

- En ny måte å tenke på
 - Alt foregår "inne i maskinen" og "inne i vårt hode"
 - Kan kjennes veldig abstrakt og u håndterlig
 - Nye begreper, konsepter, og språk-elementer
 - Og grunnelementene fra uke 1-6 sitter kanskje ikke 100% - ennå
- ⇒ Gi det tid!
- Se på eksempler, se på livekoding, løs mange oppgaver selv
 - Lær mønstrene (syntaks) og begrepene, aksepter at ikke alt kan "forstås" umiddelbart
 - Bruk læreboken, gå på grupper, se på livekoding og løs oppgaver
- ⇒ De mentale modellene av hva som foregår kommer etter hvert som du jobber med stoffet

Dette skal vi bruke de neste to månedene på!

IN1000-seminar torsdag: Innhenting

- Ta et kraftttak med pensum fra før objektorientert programmering
 - korte teori-introduksjoner
 - livekoding
 - oppgaveløsning med støtte fra gruppelærere
- Mulighet for felles mengdetrening for de som er litt mer ajour
- Enkel servering på fysisk tilbud, Zoom-tilbud for deg som ikke kan/ vil møte fysisk
- Smittevern-regler følges, så forbehold om plass for fysisk deltakelse
 - Påmeldingsfrist kl 15:00 I DAG ONSDAG
 - Lenke og mer info fra semestersiden (se under Beskjeder)

Obligatoriske krav for å få gå opp til eksamen

- 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 for innlevering!**
- Lever i god tid i tilfelle tekniske problemer eller annet. Hvis du leverer flere ganger er det siste versjon som gjelder.
- Behov for utsettelse – se under Obligatoriske innleveringer

Læringsmål uke 8

- Stoff fra forrige uke: Definere klasser, opprette objekter, kalle metoder.
- 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 sette seg inn i enkle programmer med objekter av flere klasser, samlinger av objekter, og objekter som refererer andre objekter)

Enkle og mer sammensatte typer

Repetisjon

Har brukt verdier av ulike *innebygde typer*. Enkle typer som

- Heltall (1, 45, -1)
- Sannhetsverdier (True, False)

og mer sammensatte typer som kan inneholde mer enn en verdi og tilbyr tjenester for å manipulere disse – lister, ordbøker og strenger.

Alt dette er generiske og anvendelige typer helt uavhengig av hva slags programmer vi skriver og hva de skal brukes til, og som følger med alle Python tolker.

Eksempler på sammensatte typer

Repetisjon

Lister som kan inneholde mange enkeltverdier og tilbyr tjenester som

- **append(element)**
- **pop()**

Strenger som består av 0 til mange tegn og kan manipuleres på ulike måter

- fjerne blanke med **strip()**
- gjøre om til små bokstaver med **lower()**

Filobjekter som lar oss åpne og lukke, lese og skrive på en fil

Repetisjon

Vi kan lage våre egne typer!

- Ved å definere *våre egne klasser* kan vi selv «konstruere» slike sammensatte – men mer skreddersydde – typer.
- Vi kan deretter opprette og bruke ett eller flere objekter av hver type.
- Dette er nyttig i veldig mange sammenhenger!
- Forrige forelesning laget vi klassen Student og Navn

Siri Moe Jensen

IN1000 – Høst 2020

9

Repetisjon

Eksempler fra uke 7

```
class Student:
    def __init__(self, navn):
        self._antDeltatt = 0
        self._navn = navn

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

    def hentDeltakelse(self):
        return self._antDeltatt

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

```
class Navn:
    def __init__(self, fornavn, mellom, etter):
        self._fornavn = fornavn
        self._mellom = mellom
        self._etter = etter

    def sortert(self):
        # innhold

    def naturlig(self):
        # innhold
```

Siri Moe Jensen

IN1000 – Høst 2020

10

Repetisjon

Så hva er en klasse?

- Et mønster
- Kan sammenlignes med en pepperkakeform
- Selv om vi har en klassedefinisjon i programmet vårt, har vi ingen data – og kan heller ikke gjøre noe med dem
- => Vi har kakeformen, men ingen pepperkaker.
- ..før vi oppretter *objekter* av klassen
- Ulike klasser (kakeformer) lager ulike objekter (pepperkaker)
- Vi kan lage så mange objekter (pepperkaker) vi trenger av en klasse (form)

Siri Moe Jensen

IN1000 – Høst 2020

11

Repetisjon

Å definere en klasse, opprette objekter og kalle på metoder

```
class Student:
    def __init__(self):
        self._antDeltatt = 0
    def registrer(self):
        self._antDeltatt += 1
    def hentDeltakelse(self):
        return self._antDeltatt
nyStudent = Student()
nyStudent.registrer()
```

Diagram labels:

- class Student: (klassenavn)
- def __init__(self): (konstruktør)
- def registrer(self): (metode)
- def hentDeltakelse(self): (funksjons-metode)
- nyStudent = Student() (nytt objekt)
- nyStudent.registrer() (metodekall på objektet)

Siri Moe Jensen

IN1000 – Høst 2020

12

Repetisjon

(Gjen)bruk av klasser

- Når klassen først er skrevet kan vi importere den (ved å oppgi filnavn) til andre programmer og opprette objekter av samme type der – uten å kopiere inn klassedefinisjonen

```
from navn import Navn

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

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

hovedprogram()
```

Siri Moe Jensen

IN1000 – Høst 2020

13

Repetisjon

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)

Siri Moe Jensen

IN1000 – Høst 2020

14

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 lagrer sammensatte verdier (objekter) trengs det et litt mer detaljert bilde
- Slike variabler inneholder ikke objektet selv –
men **en referanse til objektet**

Siri Moe Jensen

IN1000 – Høst 2020

17

Referanser til objekter

- Objekter lagres ikke direkte i variabler – variablene inneholder isteden bare referansen (minneadressen) 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 (referanse)

Siri Moe Jensen

IN1000 – Høst 2020

18

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()

Forskjellen på referansen og objektet

- Når endrer vi en referansevariabel og når endrer vi objektet?

```
nyDeltaker = Deltaker() # Her endres innholdet i referansevariabelen
                       # (den refererer til det nye objektet)
nyDeltaker.registrer() # Her endres innholdet i objektet variabelen
                       # refererer til ("peker på")
```

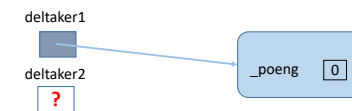
- Når vi snakker om *innholdet* i et objekt skjer det alltid med "dot-notasjon"

- Enten utenfor klassen: **nyDeltaker.registrer()**
- Eller i metodene inne i klassen: **self._poeng += 1**

Referanser til objekter

- Noen ganger er det spesielt 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?

```
deltaker1 = Deltaker()
deltaker2 = deltaker1
```



Å kopiere en referanse

- Vi kopierer *verdien* i variabelen på høyre side – akkurat som når vi kopierer en heltallsverdi
- Det er altså ikke objektet som blir kopiert – vi får bare en kopi av adressen/ referansen til objektet
- Denne verdien legges i variabelen på venstre side
- Begge referansevariablene refererer dermed til *samme objekt*

Siri Moe Jensen

IN1000 – Høst 2020

23

Referanser til objekter

```
deltaker1 = Deltaker()
deltaker2 = deltaker1
deltaker2.registrer()
print (deltaker1.hentPoeng())
```



- => Hva skrives ut i print-setningen?

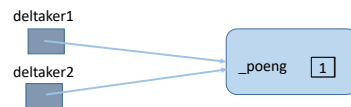
Siri Moe Jensen

IN1000 – Høst 2020

24

Referanser til objekter

```
deltaker1 = Deltaker()
deltaker2 = deltaker1
deltaker2.registrer()
print (deltaker1.hentPoeng())
```



Situasjonen før
rød pil utføres

Siri Moe Jensen

IN1000 – Høst 2020

25

Dette har du kanskje opplevd med lister?

```
liste = [1,2,3]
kopi = liste

kopi.append(5)

print (kopi)
print (liste)
```

- [Utfør i Pythontutor](#)

Siri Moe Jensen

IN1000 – Høst 2020

26

Hva skjer her? Fibonaccitall

```
def nyFib (fibTallene):
    nyttTall = fibTallene[-1] + fibTallene[-2]
    fibTallene.append(nyttTall)
    print("Liste med nytt tall: ", fibTallene)

start = [0,1]
print("Start-listen: ", start)

nyFib(start)

print("Start-listen: ", start)
```

- Fibonaccitallene: Hvert tall er summen av de to foregående
- Starter med 0, 1
- Hva skrives ut?

Hva skjer her? Fibonaccitall

```
def nyFib (fibTallene):
    nyttTall = fibTallene[-1] + fibTallene[-2]
    fibTallene.append(nyttTall)
    print("Liste med nytt tall: ", fibTallene)

start = [0,1]
print("Start-listen: ", start)

nyFib(start)

print("Start-listen: ", start)
```

- parameteren **fibTallene** får samme verdi som argumentet **start**
- den refererer med andre ord det samme liste-objektet
- append-metoden endrer liste-objektet den kalles på
- når nyFib returnerer, er listen som argumentet refererte til endret

[pythontutor](#)

Noen objekter kan ikke endres

- Noen klasser har ikke metoder som endrer instansvariablene, de kalles *immutable* – "uforanderlige"
- Hindrer feil som følge av at flere variabler refererer til samme objekt
- Strenger er eksempler på dette – alle metoder som endrer en streng returnerer et NYTT objekt i stedet for å endre det eksisterende objektet
- Må se på dokumentasjonen av klassens grensesnitt (metodene) for å vite

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

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

Strenger endres aldri – oppretter bare en ny

- Metoder for å endre en streng endrer aldri objektet de kalles på
- I stedet er de laget som funksjonsmetoder som *returnerer en ny streng*
- Denne returverdien må vi evt ta vare på for å benytte videre
- I [pythontutor](#) vises streng-variabler på samme måte som tall- og sannhetsvariabler

```
navn = "Erna Solberg"

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

print(navnMedSmaaBokstaver)
print(navn)
```

Eksempel: Klassen Rektangel

Rektangler er nyttige elementer i mange sammenhenger
=> Mulighet for gjenbruk

[Breakout-rooms: Førstemann på listen oppretter og deler en fil eller ber nestemann gjøre det. De andre dikterer. Jeg prøver å flytte folk til små grupper \(ca 3-6 i hver\)](#)

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

[Bruk chat ved behov/ spørsmål](#)

Klassen Rektangel

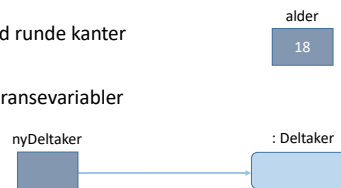
```
class Rektangel:
    def __init__(self, lengde, bredde):
        self._lengde = lengde
        self._bredde = bredde

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

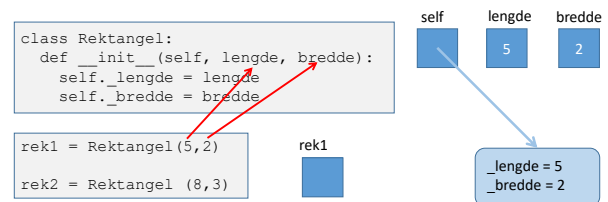
    def endre (self, lengde, bredde):
        self._lengde += lengde
        self._bredde += bredde
```

Kode og tegninger

- nyttig med tegninger når datastrukturene blir mer kompliserte etter hvert
- variabler tegnes som en navngitt boks med verdi inni – kan være en referanse
- objekter tegnes som rektangler med runde kanter
- referanser tegnes som piler fra referansevariabler



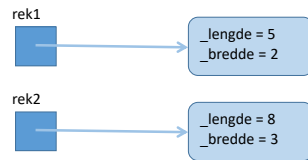
Hva skjer når vi oppretter objekter?



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?

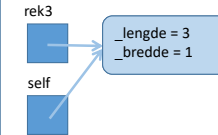
```
class Rektangel:
    def __init__(self, lengde, bredde):
        self._lengde = lengde
        self._bredde = bredde

    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 *i noen tilfeller* 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)
- Bruker **is** i stedet for **==** og **is not** i stedet for **!=**
- Gjelder all sammenligning av referanser (om de refererer samme objekt)
- Sammenligning av objekter => Neste uke

```
rek3 = None
:
:
areal = rek3.areal()
```

kjøretidsfeil!

```
if rek3 is not None:
    areal = rek3.areal()
```

ok, blir ikke utført

Klassen Navn fra uke 7

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

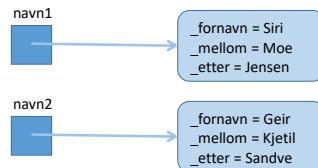
    def naturlig(self):
        natNavn = self._fornavn + " " + self._mellom + " " + self._etter
        return natNavn
```

Klassen Navn: Testprogram fra uke 7

```
from navn import Navn

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

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



```
Jensen, Siri Moe
Sandve, Geir Kjetil
Geir Kjetil Sandve
```

Flere objekter av samme klasse

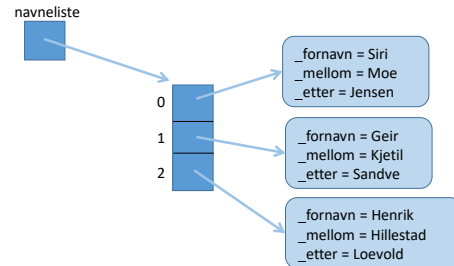
- En klasse er et mønster å lage objekter av
- Nyttig for å representere mange "ting" som følger samme mønster, f. eks. navn:

```
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 etNavn in navneliste:
    print(etNavn.sortert())
```

Datastrukturen etter innlesing av tre navn



Eksempel: Klasse Person

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

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))
  
```

eksempelperson



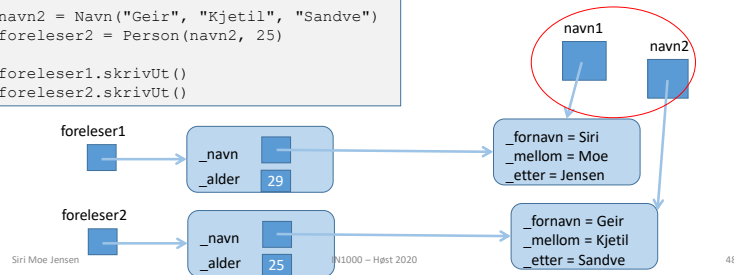
Klassen Person: Testprogram

```
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()
```



Siri Moe Jensen

IN1000 – Høst 2020

48

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")
```

Siri Moe Jensen

IN1000 – Høst 2020

49

Neste uke

- "Magiske (spesielle) metoder" for egendefinerte klasser
 - Hvordan representere objekter med innhold 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

Siri Moe Jensen

IN1000 – Høst 2020

50