Kort fra uke 7

Objektorientert programmering og objekt-orienterte språk kom som svar på utviklingen av større og mer komplekse programmer for stadig nye domener (anvendelsesområder). Objektorientering ga muligheter for å gjøre sammenhengen mellom viktige data og operasjoner i virkeligheten og programmet tydeligere, og støttet mer lesbare, oversiktlige og strukturerte programmer. Kristen Nygård og Ole-Johan Dahl spilte en viktig rolle i utviklingen av dette nye *paradigmet*, ikke minst gjennom utvikling av Simula, det første objektorienterte programmeringsspråket. For dette fikk de Turing-prisen («informatikkens Nobel-pris») i 2001 – og oppkalt de to byggene våre etter seg.

Med objektorienterte språk kan programmere bruke *objekter* av ulike *klasser* til å samle data og operasjoner som hører sammen, og la objektene tilby tjenester som kan brukes uten å vite noe om hvordan dataene er representert internt, eller hvordan tjenestene er implementert. I uke 7 lærer du hvordan du kan definere egne klasser, for deretter å opprette objekter og bruke metodene (tjenestene) disse tilbyr, i programmet ditt.

Så hva er egentlig klasser og objekter?

I Python så langt har dere brukt verdier av ulike innebygde **typer**: Heltall, sannhetsverdier, strenger og ordbøker. Strenger, lister og ordbøker er eksempler på mer sammensatte typer enn for eksempel hel- og flyttall. Dels inneholder de mer enn en verdi, dels tilbyr de tjenester for å manipulere disse dataene. Eks:

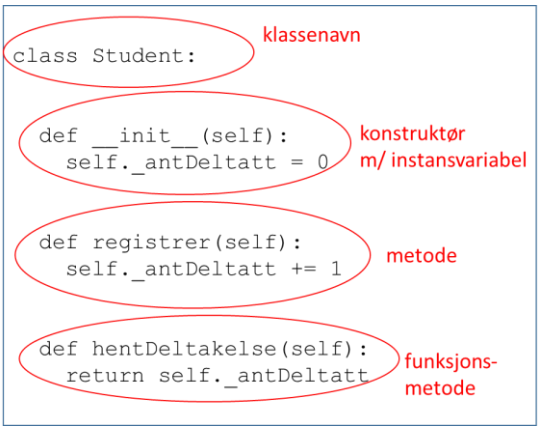
- Lister kan inneholde mange enkeltverdier og tilbyr tjenester som
  - append(element)
  - pop()
- Strenger består av 0 til mange tegn og tilbyr en rekke tjenester, f eks:
  - fjerne blanke med strip()
  - gjøre om store til små bokstaver med lower()

(merk at når man endrer en streng, blir det egentlig laget en ny streng – som returneres fra metoden)

Dette er veldig generiske og anvendelige typer helt uavhengig av applikasjonsområde. Ved å definere **våre egne klasser** kan vi selv «konstruere» slike sammensatte – men mer skreddersydde – «typer», og deretter opprette og bruke ett eller flere objekter av hver klasse. Dette er nyttig i veldig mange sammenhenger!

Vi ser på to enkle eksempler (NB – andre eksempler enn i forelesningen!!) – klassene

- Student (som kun skulle brukes for å registrere en students oppmøte på gruppetimer)
- Navn (som lagret 3 deler av et navn og tilbød tjenester for å presentere dem på ulike måter)




```
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
```

En klassedefinisjon er med andre ord en beskrivelse av et **mønster** – omtrent som 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 kaker.

Når vi **oppretter et objekt** av klassen vår (lager en pepperkake) ved å bruke klassenavnet med parenteser bak (og eventuelt noen argumenter i parentesen) kan vi ta vare på dette objektet i en variabel – og bruke tjenestene vi har definert i klassen ved å kalle på metoder for dette objektet.

```
stud1 = Student()

stud1.registrer()
```

oppretter objekt

kaller på metoden **registrer** i objektet

Når vi oppretter et objekt av en klasse, kalles konstruktøren i klassen «bak kulissene». Den avgjør hvilke instansvariabler som skal finnes i objektet, og gir dem en initiell verdi. Den eneste instansvariabelen i Student-objekter er antDeltatt, og den skal alltid starte med verdien 0.

Hvis man skal kunne velge verdi på instansvariabler når objekter av en klasse opprettes, kan dette gjøres ved å gi konstruktøren parametere som i klassen Navn – da sender vi argumenter til konstruktøren når et nytt objekt opprettes.

```
navn1 = Navn("Jonas", "Gahr", "Støre")
```

Merk at vi aldri sender med noe argument til parameteren **self** – mer om dette neste uke.

Vi kan lage så mange slike objekter av en klasse vi ønsker i et program:

```
from navn import Navn

def hovedprogram():
    navn1 = Navn("Jonas", "Gahr", "Støre")
    navn2 = Navn("Jan", "Tore", "Sanner")

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

hovedprogram()
```

Hvert objekt har sitt eget sett med instansvariabler, og vi kan håndtere så mange ulike objekter vi ønsker – for eksempel som her, der navn1 og navn2 gir tilgang til to ulike objekter med hver sine verdier i instansvariablene. Når vi kaller på en metode må vi angi hvilket objekt den skal utføres i.

Merk at mens metoden **registrer** i Student-klassen endrer verdien i objektet den blir kalt på, gjør metodene **sortert** og **naturlig** i Navn-klassen ingen endringer – de returnerer bare strenger som inneholder navnene på ulike formater. Metoder som endrer innholdet i et objekt kalles **mutatorer**, mens metoder som kun leser av innhold kalles **aksessorer** (vis overbærenhet med anglifiseringen her!). Klasser uten mutatorer er **immutable**: Objektene kan ikke endres etter at de er opprettet.

Siste eksempel viser også at når klassen først er skrevet kan vi importere den til andre programmer og opprette objekter av klassen der – uten å kopiere inn koden som definerer klassen.