

Objektorientert programmering i Python

IN1000
Høst 2020 – uke 10
Siri Moe Jensen og Geir Kjetil Sandve

Innhold uke 10

- Noen sentrale datastrukturer for programmering
 - lenkede lister
 - trær
 - grafer
- Eksempler med livekoding: Offentlig transport som
 - lenket liste
 - graf
- Nøstede lister, 2-dimensjonale strukturer
- Underveisevaluering og studietips

Siri Moe Jensen

IN1000 – Høst 2020

2

Å lage egne beholdere (containers)

- Trenger en datastruktur for å lagre ukjent antall (like) objekter. I Python har vi **lister**, **ordbøker** og **mengder** – hva om disse ikke fantes?
- Hvordan opprette en ny variabel hver gang vi trenger den – og få tak i den senere?
- Vi kan gå i løkke og opprette objekter etter behov – og vi har sett at et objekt deretter kan brukes fra ulike steder i programmet vårt, bare vi har en variabel med en referanse til objektet.
- Men hvordan sørge for nok referansevariable til et ukjent antall objekter, slik at vi kan finne dem igjen og bruke dem senere?

⇒ en klassisk datastruktur for et ukjent antall elementer er en *lenket liste*

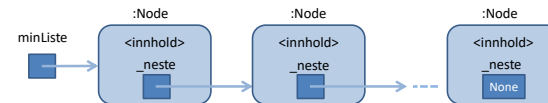
Siri Moe Jensen

IN1000 – Høst 2020

3

Lenkede lister

- Poenget med denne strukturen er at for hvert nye objekt vi lager – så lager vi samtidig en referansevariabel som kan referere til et nytt objekt
- dvs hvert objekt må kunne referere til et annet objekt
- dermed får vi en lenket liste av objekter – og trenger bare ha én referanse til det første objektet fra der vi skal bruke listen



Siri Moe Jensen

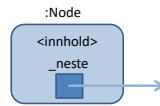
IN1000 – Høst 2020

4

Lenket liste

Lager en generell klasse for objektene i lenkelisten

- Klassen Node
 - lese, endre, skrive ut, ... [innhold](#)
- Grensesnitt for klassen Node:
 - legg til ny etterfølger
 - hent etterfølger
 - (fjern etterfølger)



Klassen Node - implementasjon

- Datastruktur Node-klassen
 - en referansevariabel så vi får tak i neste objekt
 - innholdet objektene skal representere

```
class Node :
    def __init__(self, nytt) :
        self._innhold = nytt
        self._neste = None

    def nyEtterfølger (self, ny) :
        self._neste = ny

    def hentNeste (self) :
        return self._neste

# + metoder for manipulering av selve innholdet
```

Live eksempel I - trikk

Et eksempel der lenkene representerer viktig informasjon om objektene våre (ikke bare hjelper oss å få tak i flere objekter):

Her er nodene *stasjoner på en trikkerute*, dvs. den lenkede listen består av objekter av klassen Stasjon som inneholder referanser til nabostasjoner.

<se egen presentasjon/ kildekode>

Innkapsling og generalisering

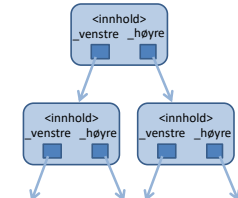
- Vi kan organisere et «uendelig» antall noder (f eks stasjoner) ved å opprette og lenke inn nye objekter av klassene Node (eller Stasjon) ved behov.
- Noen designmessige ulemper:
 - den som skal bruke klassen må kjenne til sammenlenkingen av objektene
 - for hver klasse vi ønsker å lage lister av, må vi skrive all koden som har med liste-håndtering å gjøre, om igjen – den ligger jo innbakt i klassen sammen med informasjon om innholdet
- Tiltak:
 - Skjule klassen Node (Stasjon) inne i en klasse Lenkeliste (LagRute) som håndterer (og skjuler) alle lenke-operasjoner, og kun tilbyr brukervennlige tjenester for å opprette og administrere noder (Stasjoner) i sitt grensesnitt
 - Ta emnet IN1010 til våren ☺

Andre sentrale datastrukturer

- Lenkede lister har flere ulemper
 1. Hvis man har mange noder, blir veien gjennom strukturen veldig lang, spesielt til siste node
 2. Hvis man skal representere noder med flere enn en relasjon til andre noder, holder det ikke med én neste-referanse
- Punkt 1 kan avhjelpes noe vha endringer i implementasjon av nodene og listene. Mer om dette i IN1010.
- Kan være nyttig med andre strukturer, for eksempel *trær*.

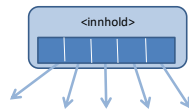
Trær

- Et tre har en node definert som **roten** i treet.
- Hver node kan ha flere **etterfølgere, barn**
- Trær har ikke sykler, dvs om man følger **kanter** fra roten eller en annen node kan man aldri komme tilbake til en man har vært i tidligere
- Spesielt binære trær (hver node har maks to etterfølgere) brukes ofte til å representere data for oppslag, sortering og traversering
- Antall etterfølgere kan ellers avhenge av hva datastrukturen skal representere



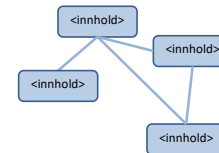
Trær

- Kan f eks modellere:
 - slektstrær
 - organisasjonskart



Generelt: Grafer

- En graf består av en mengde noder, og en mengde kanter, der hver kant forbinder to noder med hverandre.
- En vei i en graf består av en mengde kanter i rekkefølge, slik at to på hverandre følgende kanter alltid har en node felles. En sti er en vei hvor hver node besøkes høyst én gang.



Live eksempel II - kollektivnettet