UNIVERSITETET I TROMSØ UiT
FAKULTET FOR NATURVITENSKAP OG TEKNOLOGI

Løsningsveiledning for eksamensoppgaven i INF-1400 2012-05-24.

Denne veiledningen er ment å hjelpe for å forstå hvordan eksamensoppgaven det året var bygd opp og hvordan man kunne løse den. For å gjøre dette har jeg beholdt originaloppgaven og lagt til løsningsforslaget. Mørk rød tekst angir forklaringer på oppgaven.

/ John Markus Bjørndalen, 2014-05-12

Denne første delen angir vanlig bakgrunnsinformasjon. Den angir hvordan man kan besvare det (de angitte språkene var der siden det var morsmålet for noen av studentene). Det viktige under er at man ikke trenger å besvare med korrekt Python-kode. Dette er ikke et Python-kurs, derfor testes man ikke på om man husker ting som hvordan range fungerte osv. Det viktige var algoritmene og prinsippene bak objektorientert programmering.

Les gjennom hele oppgavesettet før du begynner å løse oppgavene.

Oppgavene kan besvares på Norsk, Engelsk, Svensk eller Dansk.

I besvarelsen kan du bruke Python, pseudokode eller en kombinasjon.

Noen norske termer som er brukt i stedet for de engelske fra boka:

- Klasse – class
- Arv – inheritance
- Atributt – Attribute
- Metode - Method

Del A, Nobina tar over

Denne første biten angir litt bakgrunnsinformasjon. Noe av det er en setting som gjør at man får noen knagger å henge ting på (og kanskje enklere å se at man tenker logisk). Det andre forteller litt om hva som skal gjøres i oppgaven. Det viktigste er: ikke overdriv og legg til ting som det ikke blir spurt om i oppgaven. Dere trenger f.eks. ikke å legge ting dører, billettsystem, regnummer og andre ting for busser.

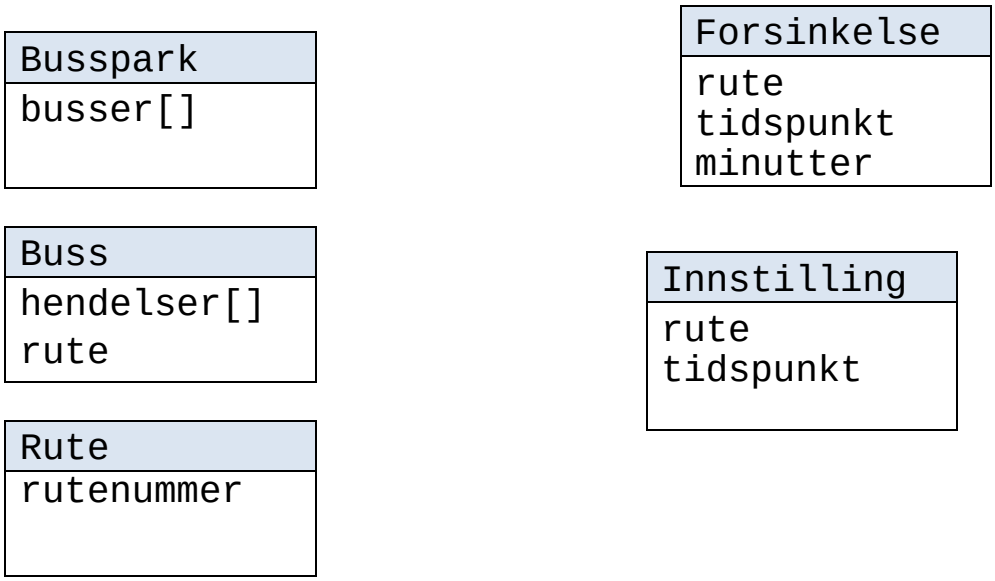
Nobina har tatt over bussdriften i Tromsø, og som vanlig er når man gjør større omlegginger dukker utfordringer opp.

Vi skal nå jobbe med et program som hjelper oss å holder oversikt over forsinkelser og andre utfordringer. Vi *forenkler også problemene* slik at vi ikke trenger å forholde oss til virkelige praktiske problemer (som f.eks. at bussene ikke starter og avslutter alle ruter på samme sted).

Merk at fokus er på objektorientering og bruk av datastrukturene vi bygger opp. Det er ikke anbefalt å prøve å utvide oppgaven på eksamen (som for eksempel å legge flere detaljer enn nødvendig i klassene).

Definisjon av oppgaven

Her begynner det som er viktig. Allerede nå kan det være lurt å ta fram et kladdeark og skissere klassene som er angitt under slik at dere har oversikt. Man kan for eksempel bruke noe UML-lignende ala dette:



Dette vil ikke være en del av besvarelsen i seg selv, men gjør det lettere å ordne neste del.

Vi definerer følgende klasser:

- Busspark – har en liste over *busser*.
- Buss – en buss skal ha en liste over *hendelser*, som kan være forsinkelser eller innstillinger. Den har også en en referanse til *ruten* den kjører i øyeblikket.
- Forsinkelse – siden en buss kan kjøre flere ruter må vi registrere følgende: *rute*, *tidspunkt* og hvor mange *minutter* forsinkelsen er på.
- Innstilling – her registrerer vi *rute* og *tidspunkt*.
- Rute - har et *rutenummer* (vi trenger ikke flere detaljer i denne oppgaven)

I oppgavene under er selve oppgaven angitt i **fet tekst (boldface)**, mens forklarende tekst er angitt som normal tekst.

Legg merke til vektingen av hver besvarelse. Den angir hvor mye
Oppgave 1 (15%)

Noen av bussene kan ikke kjøre effektivt med kjetting. En del av forsinkelsene skjer når disse bussene kjører ruter med mange bakker. For å skaffe oss bedre oversikt ønsker vi å skille mellom busstyper. Vi skal derfor introdusere to nye klasser *ElBuss* og *LangBuss* hvor langbussene er de eneste som kan kjøre med kjetting hele dagen.

Bussene trenger nå en metode som returnerer *True* for busser som kan bruke kjetting. Vi kaller denne metoden *kanBrukeKjetting()*.

Den andre endringen vi skal gjøre er å generalisere *Forsinkelse* og *Innstilling* slik at det blir enklere å jobbe med dem. Vi gjør dette ved å introdusere en ny klasse *Hendelse*.

Alle hendelses-objekter skal ha en metode vi kan kalle for å sjekke om forsinkelsen er lengre enn et angitt antall minutter, denne kaller vi *forsinketLengreEnn(minutter)*. For innstillings-objekter vil denne bestandig returnere *True*, mens for forsinkelses-objekter må vi sammenligne de angitte minuttene med hvor mange minutter som er angitt i forsinkelses-objektet.

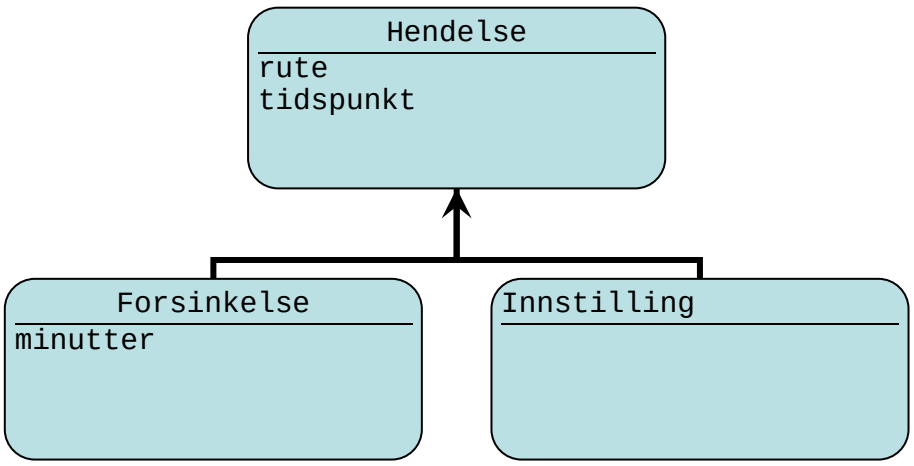
Forklar kort ved hjelp av et klassediagram hvordan du kan bruke arv når du legger til

a) **Hendelse og**
b) **de nye Buss-klassene**.

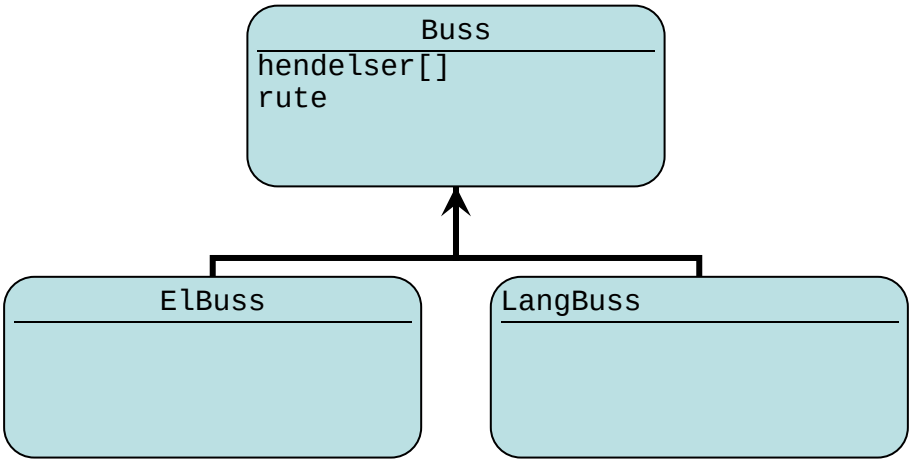
Det er kun behov for å fokusere på arv og på attributtene til klassene siden vi skal jobbe med metodene senere.

Vi tester her basal forståelse for arv og klasser. Teksten over angir at vi kun skal fokusere på attributter og arv. Dermed blir dette enkelt, og det under er en fullgod besvarelse.

a) Når vi introduserer Hendelse flytter vi alle felles attributter over i den slik at Innstilling og Forsinkelse arver fra den. Vist som her:



b) Når vi introduserer de to nye bussklassene skal de arve fra Buss. Ingen av dem introduserer noen nye attributter som vi trenger å ta hensyn til her. Dermed blir klassediagrammene som følger.



Oppgave 2 (10%)

I oppgaven over brukte vi to måter å endre på klassehierarkiet på.

a) **Generalisering var den ene, hva er navnet på den andre?**
b) **Hva er forskjellen på disse to? Vi trenger bare en kort forklaring av hovedforskjellen, ikke en lang utdypende forklaring.**

Svarene trenger ikke å være lange. Det som står under er nok (spesielt når jeg ber om korte svar).

a) Spesialisering.
b) Generalisering er når vi lager en felles base/parent-klasse for et sett med klasser slik at de arver fra baseklassen. Man samler felles funksjonalitet i attributter i baseklassen. Spesialisering er når man lager underklasser (child class) av en eksisterende klasser.

Eventuelt kan dere koble det til hvor dere brukte generalisering og spesialisering over.

Oppgave 3 (15%)

Implementer klassene vi har spesifisert i oppgave 1. Du trenger ikke å ta med noen andre metoder enn __init__() og metodene vi spesifiserte i oppgave 1 siden vi skal jobbe videre med klassene senere.

Følgende er en kjapp kladdeimplementasjon, så jeg har ikke sjekket om det vil fungere korrekt Python-kode. Den viser likevel det vi er ute etter i besvarelsen, hvorav de viktigste er:

- Dere får lagd klasser som kan initialisere objekter, og at hver klasse initialiserer de rette attributtene.
- Dere får vist arv (arving av attributter, angiving av arv når dere lager klasser)
- At de rette klassene arver fra de rette parent-klassene (hierarkiet er korrekt)

- At metodene er på korrekt sted og gjør rett ting i forhold til spesifikasjonen. F.eks. hvordan og hvor dere implementerer forsinketLengreEnn.

```
# Det er ikke angitt noe sted at man skal ta med initielle verdier
# til attributtene i__init__, derfor kan vi godt utelate det og
# anta at det settes på et eller annet vis i koden ellers.

class Hendelse(object):
    def __init__(self):
        self.rute = None
        self.tidspunkt = None
    def forsinketLengreEnn(self, minutter):
        # Dette er "optional" siden det ikke er spesifisert i
        # oppgaven, men det er ofte en god idé å kaste en
        # exception hvis det ikke er meningen at noen skal
        # bruke metoden i baseklassen.
        raise Exception("not allowed on the base class")

class Forsinkelse(Hendelse):
    def __init__(self):
        Hendelse.__init__(self)
        self.minutes = None

    def forsinketLengreEnn(self, minutter):
        return self.minutes > minutter

class Innstilling(Hendelse):
    def __init__(self):
        Hendelse.__init__(self)

    def forsinketLengreEnn(self, minutter):
        return True


class Buss(object):
    def __init__(self):
        self.rute = None
        self.hendelser = []
    def kanBrukeKjetting(self):
        # Annet alternativ til default-metode: returner noe som
        # gir mening for de fleste underklassene.
        return False

class LangBuss(Buss):
    def __init__(self):
        Buss.__init__(self)
    def kanBrukeKjetting(self):
        return True

class ElBuss(Buss):
    def __init__(self):
        Buss.__init__(self)
```

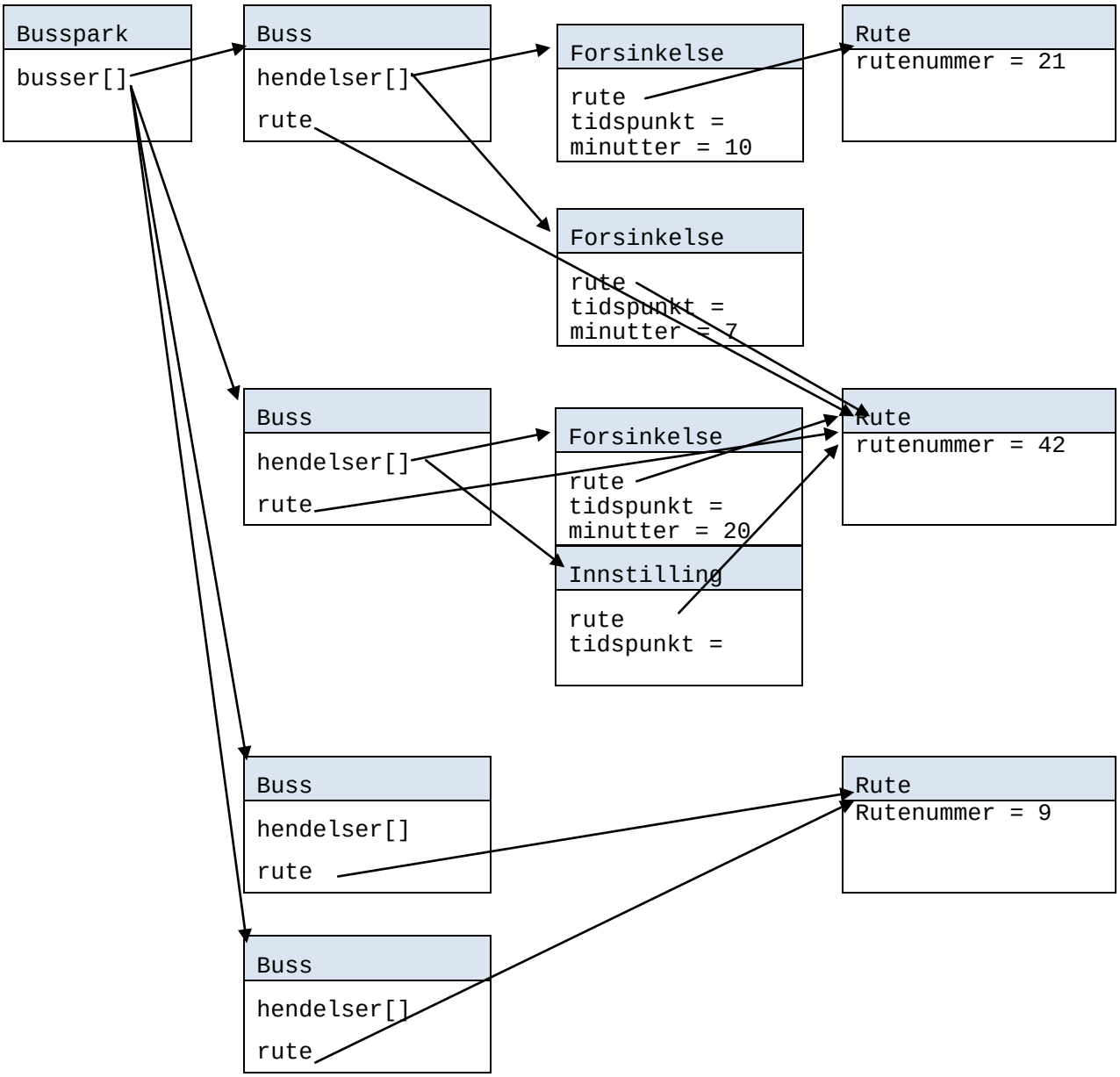
```
class Busspark(object):
    def __init__(self):
        self.busser = []
```

Oppgave 4 (10%)

Tegn opp en datastruktur med et lite antall busser, ruter, forsinkelser og innstillinger.
Du trenger ikke angi verdier for alle attributtene i alle objektene.

Her viser dere at dere forstår forskjellen på arv og på initialiserte objekter. Vi tar utgangspunkt i en Busspark og viser hvordan den inneholder flere busser som kanskje har hendelser. Videre viser vi hvordan objektene er koblet sammen i minnet når vi har satt opp et antall objekter (f.eks. hvordan busser kan være koblet opp mot ruter og forsinkelser mot ruter).

For tidspunkt (ikke angitt i figuren under) kan dere f.eks. skrive 20:35 og lignende, eller dato og tidspunkt.


Busspark
busser[]
Buss
hendelser[]
rute
Forsinkelse
rute
tidspunkt =
minutter = 10
Rute
rutenummer = 21
Forsinkelse
rute
tidspunkt =
minutter = 7
Buss
hendelser[]
rute
Forsinkelse
rute
tidspunkt =
minutter = 20
Rute
rutenummer = 42
Innstilling
rute
tidspunkt =
Buss
hendelser[]
rute
Rute
Rutenummer = 9
Buss
hendelser[]
rute

Oppgave 5 (15%)

Fylkeskommunen trenger en oversikt over hvor mange forsinkelser og innstillinger det har vært på en gitt dag. Som en forenkling antar vi at bussene kun har registrert hendelser fra siste døgn, slik at vi ikke trenger å sjekke datoene.

Metoden *finnHendelser(self, minForsinkelse)* skal lage en liste over alle hendelsene som har skjedd siste døgn som har ført til en forsinkelse over «minForsinkelse» minutter. Metoden skal returnere en tuple som har følgende informasjon:

(antall innstillinger, antall forsinkelser over «minForsinkelse» minutter,
liste over alle hendelser som førte til forsinkelser over «minForsinkelse»).

En Innstilling vil automatisk være en forsinkelse over N minutter for alle mulige valg av N.

Lag metoden finnHendelser. Angi hvilken klasse du vil legge den i.

Her hjelper det godt å se på forrige besvarelse og tenke hvordan man skal søke igjennom ting. Det naturlige er å legge metoden i Busspark siden den da har tilgang til listen over busser og dermed kan lete igjennom alle hendelsene for hver buss. En grei implementasjon er som følger:

```
class Busspark(object):
    def __init__(self):
        self.busser = []

    def finnHendelser(self, minForsinkelse):
        antFors = 0
        antInst = 0
        forsinkelser = []
        for buss in self.busser:
            for hendelse in buss.hendelser:
                if hendelse.forsinketLengreEnn(minForsinkelse):
                    if isinstance(hendelse, Forsinkelse):
                        antFors += 1
                    elif isinstance(hendelse, Innstilling):
                        antInst += 1
                    forsinkelser.append(hendelse)
        return (antInst, antFors, forsinkelser)
```

Del B

Oppgave 6 (15%)

Velg to av kulepunktene under og gi en kort beskrivelse av uttrykkene/konseptene som er angitt der.

1. object vs. class
2. parameter
3. instance
4. method
5. mutable vs. immutable

Enkelte leste tydeligvis ikke oppgaveteksten over godt nok og svarte like godt på alle punktene. Andre kom med lange utledninger når vi ba om en kort beskrivelse. Eksempler på besvarelser som kan fungere er:

1) En klasse er en template som beskriver hvordan en type implementeres. Den vil angi attributter og implementasjonen av metoder. Objekter er instanser av klasser, det vil si når man har allokert et objekt av en gitt klasse/type.
2) Et parameter er input-verdier til en funksjon eller metode. De er angitt i definisjonen av funksjonen eller metoden. For eksempel er bar en parameter til foo under:
   ```
   def foo(bar):
       print bar
   ```
3) En instance er et allokert objekt av en gitt type/klasse.
   Alternativt kan dere også si at instanse og objekt er synonymt.
4) En metode er en funksjon som er assosiert med en klasse eller et objekt. Man kan typisk kalle metoden til et objekt via objektet, f.eks: object.method()
5) En av de korteste besvarelsene som fikk full score var noe sånt som:
   Et objekt som er mutable kan endres. Eksempel er lister.
   Et objekt som er immutable kan ikke endres. Eksempel er tuple i Python.

Det er alltids lov å utdype litt, men det er sjelden noen klarer å score mer poeng etter å ha fylt en halv side per deloppgave.
Det vi hovedsakelig er ute etter når vi spør slike spørsmål er

a) Vet dere forskjellen på disse to begrepene (f.eks. i 1 og 5)
b) Vet dere det viktigste som kjennetegner eller definerer det vi spør om (f.eks. 2, 3, 4)

Oppgave 7 (10%)

```
class DefaultVar(object):
    def __init__(self, var1, var2 = 42, var3 = 22):
        self.var1 = var1
        self.var2 = var2
        self.var3 = var3
        print "Init with v1", self.var1, "v2", self.var2, \
              "v3", self.var3
```

Gitt koden over, hva vil programmet skrive ut for hvert av uttrykkene under?

```
a) dv1 = DefaultVar(100)
b) dv2 = DefaultVar()
c) dv3 = DefaultVar(100, var3 = 900)
```

Denne oppgave tester forståelse av default-variabler og initialisering og bruk av attributter. Dere trenger bare veldig korte svar her. F.eks:

a) `Init with v1 100 v2 v2 42 v3 22`
b) Denne vil kaste en exception (dere kan også si feile) siden var1 ikke er angitt, og den har ingen default-verdi.
c) `Init with v1 100 v2 42 v3 900`

Oppgave 8 (10%)

```
class Vector(object):
    def __init__(self, u = 0, v = 0):
        self.u = u
        self.v = v

    def getU(self):
        return self.u

    def setU(self, u):
        self.u = u

    def normalize(self):
        factor = math.sqrt(self.u ** 2 + self.v ** 2)
        self.u /= factor
        self.v /= factor

    def __str__(self):
        return "<%f, %f>" % (self.u, self.v)
```

a) Hva er forskjellen på en accessor og en mutator?
b) Gitt koden over, angi hvilke metoder som er accessor og mutator.

Igjen trenger vi bare korte svar, så lenge de berører de rette punktene. Eksempel kan være:

a) En accessor er en metode som leser verdier fra et objekt uten å endre på objektets tilstand mens en mutator er en metode som endrer på objektets tilstand.
b) Accessor: getU, __str__
Mutator: __init__, setU, normalize()

Jeg tror ikke vi trekte noen for å ha glemt __init__.