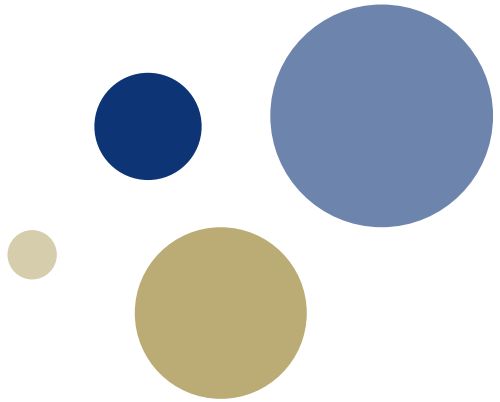
NTNU

Kunnskap for en bedre verden



Repetisjon, del 1

TDT 4110 IT Grunnkurs

Professor Guttorm Sindre

Resultat av Kahoot!

- Følgende temaer hadde størst behov
  - på en skala fra 1 – lite behov – til 3 – stort behov:
    - Binærfiler 2,5
    - Rekursjon 2,3
    - 2D-lister / tabeller 2,3
    - Tekstfiler 2,3
    - Unntaksbehandling 2,3
    - Dictionaries 2,25
    - Doble løkker 2,15
    - Lister og tupler 2,08
    - Strenger 2,08
    - Mengder 2,0
    - …

Plan for repetisjon

- Aritmetiske operatorer og syntaks: 3b aug.2016
- Funksjoner, parametre, returverdier
- Enkle løkker, lister, strenger: Oppg 4 a-e aug.2015
- Dictionary, filer, unntak: Oppg 4 f, ”g”, aug.2015
- 2D-lister, doble løkker: Aug. 2015: 2 e-f, Des. 2014: 3 a-b
  - Oppg 3e, Aug 2016

Aritmetiske operatorer og syntaks

- Ikke mye trekk for syntaksfeil, men litt
  - Hvis du trenger full pott: unngå syntaksfeil
  - Skriv kolon alle sted det skal være
  - Skriv operatorer på Python-måte, ikke matematisk
    - Slik: x >= 1 Ikke slik: x ≥ 1
    - Slik: x ** 3 Ikke slik: x^3
    - Slik: a * b Ikke slik: a·b
    - Slik: (a + 2) / b Ikke med horisontal brøkstrek eller kolon
- Vær nøye med innrykk
  - feil kan her også bli semantisk

Operatorer og syntaks (2)

- Husk heltallsdivisjon // og modulo % der det passer
  - Enklere kode, vanlig divisjon gir behov for avrunding / typekonvertering / andre ekstra beregninger
  - Eksempel: oppgave 3b, august 2016

Oppgave 3b (4%)

Når trenerne ser på klokka under kamper for å sjekke når neste innbytte skal skje, er det enklere å forholde seg til minutter og sekunder enn kun sekunder. Skriv derfor en funksjon `minutt_sekund(sekunder)` hvor innparameter er antall `sekunder` (heltall) og returverdi en streng på formen '`mm:ss`'. Hvis antall minutter er mindre enn 10, skal strengen ha formen '`m:ss`', mens sekunder alltid skal representeres som to tegn, også om det er mindre enn 10. Eksempel på kjøring

```
>>> minutt_sekund(120)
'2:00'
>>> minutt_sekund(206)
'3:26'
>>>
```

Enkle (1D) lister og løkker

- Viktig å vite hvordan man
  - Bygger opp lister
  - Aksesserer elementer i lister
    - Forskjell på å aksessere verdiene direkte og via indeks
    - Og når det passer å bruke hva
  - Plukke og fjerne elementer fra lister
- Når bruke while-løkke, når bruke for-løkke?

Oppgave 4, august 2015 (a)

NB!

Oppgave 4 Programmering vitneobservasjoner (45%)

I noen av oppgavene kan det være gunstig å kalle funksjoner som du har laget i tidligere deloppgaver. Selv om du ikke har fått til den tidligere oppgaven, kan du kalle funksjon derfra med antagelse om at den virker som spesifisert i oppgaveteksten.

Politiet trenger et system for å sjekke om vitneobservasjoner av kjøretøyer fra hendelser som etterforskes, stemmer med faktiske kjøretøyer i et register.

Oppgave 4 a) (5%)

Skriv en funksjon **les_inn_bilinfo( )** som leser inn fra tastatur vitnets observasjon av bilmerke, modell og farge for et kjøretøy. Funksjonen skal returnere disse tre opplysningene i en liste.

NB!

Eksempel på kjøring (det i fete typer tastes inn av brukeren):

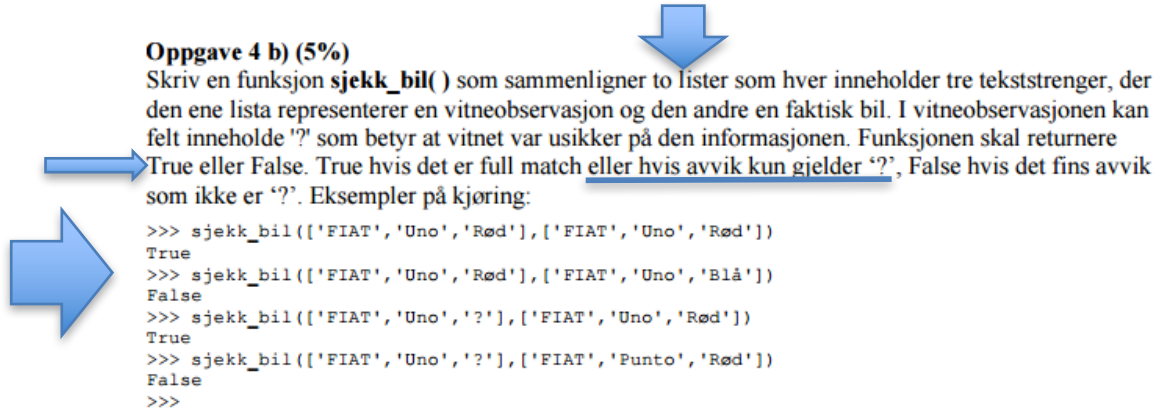
```
>>> les_inn_bilinfo()
Hvilket bilmerke var det? FIAT
Hvilken modell? Uno
Hvilken farge? Rød
['FIAT', 'Uno', 'Rød']
>>>
```

- Ser av ”Eksempel på kjøring at
  - Funksjonen ikke trenger noen parameter
  - Det er tre ulike input-setninger
- Dette er ei kort liste (kun 3 element)
  - Trenger ikke bruke løkke, kan bare opprette den direkte
  - Løkke blir bare vanskeligere, fordi ledeteksten er ulik

Løsning 4a:

```
def les_inn_bilinfo():
    merke=input('Hvilket bilmerke var det? ')
    modell=input('Hvilken modell? ')
    farge=input('Hvilken farge? ')
    return [merke,modell,farge]
```

Oppgave 4, august 2015 (b)



Oppgave 4 b) (5%)
Skriv en funksjon **sjekk_bil( )** som sammenligner to lister som hver inneholder tre tekststrenger, der den ene lista representerer en vitneobservasjon og den andre en faktisk bil. I vitneobservasjonen kan felt inneholde '?' som betyr at vitnet var usikker på den informasjonen. Funksjonen skal returnere True eller False. True hvis det er full match eller hvis avvik kun gjelder '?', False hvis det fins avvik som ikke er '?'. Eksempler på kjøring:

```
>>> sjekk_bil(['FIAT','Uno','Rød'],['FIAT','Uno','Rød'])
True
>>> sjekk_bil(['FIAT','Uno','Rød'],['FIAT','Uno','Blå'])
False
>>> sjekk_bil(['FIAT','Uno','?'],['FIAT','Uno','Rød'])
True
>>> sjekk_bil(['FIAT','Uno','?'],['FIAT','Punto','Rød'])
False
>>>
```

- Eksempel på kjøring viser to innparametre, begge lister
- Repetisjon: Hvordan aksessere enkeltelement i lister?
  - Hvis vi bare trenger verdiene kan for verdi in liste: funke
  - Men ikke her, må parallelt sammenligne element i to lister
  - Da må vi bruke indeks

Løsning 4b:
Flere greie måter å gjøre denne på:

```
def sjekk_bil(vitneobs,bildata):  # LØSNING MED LØKKE
    for i in range(3):
        if vitneobs[i] != '?' and vitneobs[i] != bildata[i]:
            return False
    return True
```

Oppgave 4, august 2015 (c)

Oppgave 4 c) (5%)

Gitt tuppelet `SKILTBOKSTAV = ('A','B','C','D','E','F','G','H','J','K','L', 'N','P','R','S','T','U','V','X','Y','Z','?')`

Dette inneholder bokstaver som er lov å bruke på moderne norske bilskilt, samt '?' helt bakerst (hvis vitnet ikke husker). Skriv en funksjon **les_gyldig_vitneskilt()** som leser inn fra tastatur en streng på nøyaktig 7 tegn, hvorav de 2 første tegnene skal være tegn fra tuppelet SKILTBOKSTAV, og de fem siste tegnene skal være tall eller ?. Ved feil input skal funksjonen be brukeren gjøre et nytt forsøk, inntil input er gyldig. Da skal funksjonen returnere strengen. Eksempel på kjøring:

Indikerer løkke

Ulik respons ved ulike feil Indikerer if-elif-…-else inni løkka

```
>>> les_gyldig_vitneskilt()
Skriv inn skilt, 2 bokst + 5 tall (?=usikker) VFC1111
Fem siste tegn må være tall eller ?
Skriv inn skilt, 2 bokst + 5 tall (?=usikker) 8V12345
To første tegn må være lovlig skiltbokstav eller ?
Skriv inn skilt, 2 bokst + 5 tall (?=usikker) VF123456
Skiltnummer må være 7 tegn langt
Skriv inn skilt, 2 bokst + 5 tall (?=usikker) V???888
'V???888'
>>>
```

- Repetisjon: hva slags løkke velger vi?
  - for : antall repetisjoner kjent når løkka starter
    - For eksempel gjennomløpe alle element i liste
  - while : antall repetisjoner ukjent
    - Passer her, uvisst hvor lenge brukeren feiler
    - Hva blir sluttbetingelsen? Gyldig input gitt
    - Initialisering: Anta feil input inntil det motsatte er bevist
- if-elif-…else: ofte lurt å starte med enkleste betingelse
  - Her kanskje at input har feil lengde (ikke 7 tegn)

4c, forts.

Antar feil inntil motsatte bevist

Enkleste betingelse

Nest enkleste

Vanskeligste

Passert alle tester, dermed gyldig

Løsning 4c:

```
def les_gyldig_vitneskilt():
    gyldig=False
    while not gyldig:
        streng=input('Skriv inn skilt, 2 bokst + 5 tall (?=usikker) ')
        if len(streng)!=7:
            print('Skiltnummer må være 7 tegn langt')
        elif (streng[0] not in SKILTBOKSTAV
              or streng[1] not in SKILTBOKSTAV):
            print('To første tegn må være skiltbokstav eller ?')
        elif (streng[2:7] != '?????'
              and not streng[2:7].replace('?','').isdigit()):
            print('Fem siste tegn må være tall eller ?')
        else:
            gyldig=True
    return streng
```

Koden fra og med `elif (streng[2:7]…)` i stedet løses med en løkke (og dette er ikke dårligere):

```
    else:
       gyldig = True # antar ok input inntil motsatte er bevist
       for tegn in streng[2:7]:
           if tegn not in '?0123456789':   # evt. mengde / liste / tuppel a la SKILTBOKSTAV
               print('Fem siste tegn må være tall eller ?')
               gyldig = False
return streng
```

Oppgave 4, august 2015 (d)

Oppgave 4d (5%)
Skriv en funksjon **match( )** som skal sjekke om et vitneobservert skilt kan stemme overens med et faktisk skiltnummer. Funksjonen må ta inn de to strengene som skal sammenlignes som parametere. Returner True hvis det er en hel match (strengene er identiske) eller mulig match (de eneste forskjellene skyldes ?), og False hvis de to ikke kan stemme overens. Eksempel på kjøring:

```
>>> match('VF12345','VF12355')
False
>>> match('V?1234?','VF12355')
False
>>> match('VF???55','VF12355')
True
>>> match('??12355','VF12355')
True
>>> match('???????','VF12355')
True
>>>
```

- Minner om 4b,
  - nå sammenligning av strenger, ikke lister
  - Naturlig med for-løkke (vet antall rep.)
  - Må aksessere med indeks fordi samme posisjon er viktig

Løsning 4d:

```
def match(vitneskilt,regnr):
    test=True
    for i in range(0,7):
        if vitneskilt[i] != '?' and vitneskilt[i] != regnr[i]:
            test=False
    return test
```

Oppgave 4, august 2015 (e)

Oppgave 4 e) (5%)
Skriv en funksjon **match_liste( )** som sammenligner ett vitneobservert skilt med en liste av faktiske skilt. Funksjonen skal returnere lista av alle skilt som *kan* stemme med det observerte skiltet.
Eksempel på kjøring:

```
>>> match_liste('VF???55',['VX33322','VF12355','VF77455','DA?????','VF10055'])
['VF12355', 'VF77455', 'VF10055']
>>>
```

- Her passer en løkke av typen for element in liste:
  - Posisjon i lista ikke viktig
  - NB: sjekken av ett og ett skilt inni løkka er det som ble gjort i 4d

Løsning 4e:

```
def match_liste(vitneskilt,skiltliste):
    resultatliste=[]
    for skilt in skiltliste:
        if match(vitneskilt,skilt):
            resultatliste.append(skilt)
    return resultatliste
```

4d

Oppgave 4, august 2015 (f)

Oppgave 4 f) (20%)
Anta at vi har en tekstfil `biler.txt` med format som vist i utdraget under, dvs. skiltnummer, bilmerke, modell, farge og navn på eier, hvor hvert element er adskilt med mellomrom.

```
DK21518 FIAT Panda Blå Os,Liss
GH70709 Ford Mondeo Blå Hansen,Jo
FB37769 FIAT Panda Brun Å,Ole
TD79641 Ford S-Max Grå Berg,Jo
PE66975 Toyota Avensis Gul Nes,Al
JV13133 VW Polo Brun Bø,Ole
CG74083 FIAT Panda Blå Hansen,Ann
ZG27056 Toyota Previa Grønn Berg,Bo
```

Skriv et skript eller en main()-funksjon som gjør følgende:

- Les inn data fra fila `biler.txt` og putter i en dictionary. Bruk unntaksbehandling for å unngå krasj hvis fila mangler.
- La brukeren sjekke den ene vitneobserverte bilen etter den andre opp mot det som fins i dictionary'en, inntil brukeren ønsker å slutte.
- For hver bil som sjekkes, skriv ut potensielle treff til skjerm, dvs. alle biler hvor de opplysningene som ikke var '?', matchet. Vis på skjerm skiltnummer og navn på eier.
- Hvis ingen kjøretøy matcher, skal programmet skrive ut 'Ingen match'

Du bestemmer selv om du vil skrive all koden for dette i skriptet / main( ), eller om du vil dele det opp i flere funksjoner, men god oppdeling vil telle positivt der det er naturlig. Likeledes vil det telle positivt om du klarer å bruke funksjoner fra tidligere deloppgaver der det passer. Eksempel på kjøring (hvis fila kun besto av de linjene som var vist i utdraget ovenfor):

```
>>> main()
Fil lest
Hvilket bilmerke var det? FIAT
Hvilken modell? Panda
Hvilken farge? ?
Skriv inn skilt, 2 bokst + 5 tall (?=usikker) ???????
Mulige kjøretøyer er:
FB37769 Eier: Å,Ole
DK21518 Eier: Os,Liss
CG74083 Eier: Hansen,Ann
Vil du sjekke flere kjøretøyer? (J/N) J
Hvilket bilmerke var det? VW
Hvilken modell? Polo
Hvilken farge? Brun
Skriv inn skiltnummer 2 bokstaver + 5 tall (? for usikker) JV33333
Ingen match
Vil du sjekke flere kjøretøyer? (J/N) N
>>>
```

- Større og mer ullen oppgave…
- Hvordan tenke?
  - Fins det noen tydelige etapper den kan deles opp i?
    1. Lese fil, legge data inn i dictionary
    2. Løkke (så lenge bruker vil fortsette):
       a) lese inn observasjoner fra bruker
       b) Sjekk opplysninger mot dictionary
       c) Skriv info om matchende biler til skjerm
  - Fins det muligheter til å bruke funksjoner 4a-e ?

Etappe 1: fra tekstfil til dictionary

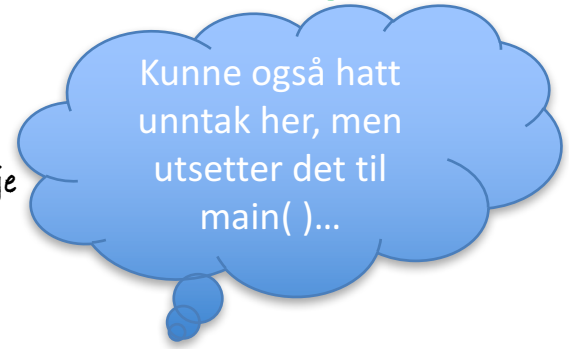
```
DK21518 FIAT Panda Blå Os,Liss
GH70709 Ford Mondeo Blå Hansen,Jo
FB37769 FIAT Panda Brun Å,Ole
TD79641 Ford S-Max Grå Berg,Jo
PE66975 Toyota Avensis Gul Nes,Al
JV13133 VW Polo Brun Bø,Ole
CG74083 FIAT Panda Blå Hansen,Ann
ZG27056 Toyota Previa Grønn Berg,Bo
```

{ nøkkel : verdi ,
nøkkel : verdi, …}

{ 'DK21518' : ['FIAT', 'Panda', 'Blå', 'Os,Liss'],
'GH70709' : ['Ford', 'Mondeo', 'Blå', 'Hansen,Jo',
…}

- Hva skal være nøkkel, og hva skal være verdi?
  - Bilmerke, modell eller farge passer ikke som nøkkel
    - Ikke unike, mange biler får samme nøkkel
  - Eier? Nei, den er ukjent, ikke det vi skal bruke til oppslag
  - Nøkkel: skiltnr. Verdi: liste med merke, modell, farge, eier
- Overordnet ide:

```
Åpne tekstfil
Opprett tom dictionary
For hver linje i fil:
        Finn enkeltdata fra linje
        Putt data i dictionary
Lukk tekstfil
Returner dictionary
```



Etappe 1: Løsning

Løsning 4f:
Bruker tre hjelpefunksjoner, en som leser fra fil og legger inn i dictionary, en som går gjennom dictionary og finner mulig matchende biler, og en som skriver ut resultat til skjerm. Andre oppdelinger kan også være mulig.

```
def les_biler_fra_fil(filnavn):
    fil=open(filnavn,'r')
    dic={}
    for linje in fil:
        linje=linje.strip('\n')
        liste=linje.split()
        dic[liste[0]] = liste[1:]
    fil.close()
    return dic
```

» Skilt blir nøkkel resten av lista blir verdi

Etappe 2: main( )

- Overordnet ide:

```
try:
        # Kalle funksjonen som leser tekstfila
        bil_db = les_biler_fra_fil('biler.txt')
except:
        skriv passende feilmelding
else:
        fortsett = True
        while fortsett:
                les_inn_bilinfo()              # 2a
                nr = les_gyldig_vitneskilt()  # 2b
                skiltliste  = matchende skilt i bil_db
                vis resultater på skjerm
                spør bruker om å fortsette, les inn
svar
                hvis svar er nei:
                        fortsett = False
```

Disse to linjene
er ikke-trivielle,
kan være lurt å gjøre
som hjelpefunksjoner

main() og skript - løsning

```
def main():
    try:
        bil_db = les_biler_fra_fil('biler.txt')
        print('Fil lest')
    except Exception as feil:
        print('Problemer med lesing av fil, feilmelding:')
        print(feil)
    else:
        fortsett=True
        while fortsett:
            bilinfo = les_inn_bilinfo()
            regnr = les_gyldig_vitneskilt()
            skiltliste = match_bildata(regnr,bilinfo,bil_db)
            vis_resultat(skiltliste)
            svar = input('Vil du sjekke flere kjøretøyer? (J/N) ')
            if svar.upper() == 'N':
                fortsett = False

main()
```

Hjelpefunksjoner, løsning

```
def match_bildata(vitneskilt,bilinfo,bil_db):
    kandidater=[]
    mulige_skilt = match_liste(vitneskilt,bil_db.keys())
    for skilt in mulige_skilt:
        if sjekk_bil(bilinfo,bil_db[skilt]):
            kandidater.append(skilt)
    return kandidater

def vis_resultat(skiltliste):
    if skiltliste == []:
        print('Ingen match')
    else:
        print('Mulige kjøretøyer er:')
        for element in skiltliste:
            print(element, 'Eier:', bil_db[element][3])
```

4e

4b

- Kunne hatt dette som en funksjon i stedet for to
  - Ikke avslutte match_bildata med return kandidater
  - I stedet fortsette med if kandidater == [ ]
  - skrive ut direkte i denne funksjonen

Binærfiler

- For å få repetert dette: ekstraoppgave (”4 g”)
- For å slippe tidkrevende lesing fra tekstfil og konvertering av data til dictionary hver gang, ønsker vi å lagre dictionary til binærfil, og lese den fra binærfil når programmet startes. Hvis binærfilen ikke fins, skal programmet i stedet prøve å hente fra tekstfil. Lag funksjoner for å lagre dictionary til fil og hente den fra fil, og vis hvordan du ellers ville forandre programmet for å inkludere denne funksjonaliteten.

Lagre og hente, løsning

```
import pickle

def lagre_data(data,filnavn):
    fil=open(filnavn,'wb')
    pickle.dump(data,fil)
    fil.close()

def hent_data(filnavn):
    try:
        fil = open(filnavn,'rb')
        data = pickle.load(fil)
        fil.close()
    except:
        return {}
    else:
        return data
```

Endret main() - løsning

```
# i main()
        bil_db = hent_data('biler.dat')
        if bil_db != {}:
            print('Binærfil lest.')
            fortsett = True
        else:
            # try: (lese tekstfil...)
        #  kode vi hadde fra før...
        # (nederst, etter while-løkke)
        # Lagrer dictionary i binærfil:
        lagre_data(bil_db, 'biler.dat')
        print('Binærfil lagret, takk for nå!')
```

2-D lister, eks. des. 2014 oppg 3a

```
>>> for row in weatherData:
      print(row)

[12.0, 2.4, 8.2]
[6.1, 0.6, 11.9]
[8.3, -3.5, 0.0]
[11.6, -5.2, 0.0]
[15.3, 2.8, 14.3]

>>> weatherData[0]
[12.0, 2.4, 8.2]
>>> weatherData[1]
[6.1, 0.6, 11.9]
>>>
```

```
>>> weatherStats(weatherData)
There are 5 days in the period.
The highest temperature was 15.3 C on day number 5
The lowest temperature was -5.2 C on day number 4
There was a total of 34.4 mm rain in the period
>>>
```

- Gitt: 2D-liste `weatherData`
- hver indre liste er værdata for ett døgn
  - Høyeste temperatur
  - Laveste temperatur
  - Nedbør
- Ønsker statistikk for perioden (nede venstre):
  - Høyeste temperatur
    - Og hvilken dag den inntraff
  - Laveste temperatur
    - Og hvilken dag…
  - Nedbør
- Problem: hvordan komme fra A til B?

Hvordan tenke her?

- Overordnet plan:
  - Gi startverdier til høyeste temp., høyeste dag, laveste temp., laveste dag og total nedbør
    - Men hva er fornuftige startverdier for hver av disse?
  - Gå i løkke dag for dag:
    - Total nedbør = total nedbør + dagens nedbør
    - Hvis dagens høy.temp. > høyeste temp.
      - Høyeste temp. = dagens høy.temp
      - Høyeste dag = dagens nummer
    - Hvis dagens lav.temp < laveste temp.
      - Laveste temp = dagens lav.temp
      - Laveste dag = dagens nummer
  - Etter løkka: print resultatene

- Viktig å skjønne for 2D-lister: Hvilken indeks er til hva?

Løsningsforslaget

Mulig løsning 3a:

```
def weatherStats(weatherData):# Problem 3a
  days=len(weatherData)
  htemp= weatherData [0][0]
  ltemp= weatherData [0][1]
  hday=lday=1
  rain=0
  daycount=0
  # Find lowest and highest temp and date
  for daydata in weatherData:
    rain=rain+daydata[2]
    daycount+=1
    if daydata[0]>htemp:
      hday=daycount
      htemp=daydata[0]
    if daydata[1]<ltemp:
      lday=daycount
      ltemp=daydata[1]

  print('There are',days,'days in the period.')
  print('The highest temperature was',htemp,'C on day number',hday)
  print('The lowest temperature was',ltemp,'C on day number',lday)
  print('There was a total of',rain,'mm rain in the period')
```

```
def weatherStats(weatherData):  # alternativ løsning (bare marginalt ulik, to indekser)
  htemp = weatherData[0][0] # antar første dag varmest...
  ltemp = weatherData[0][1] # og kaldest inntil noe annet viser seg varmere / kaldere
  hday = lday = 1           # dagnummeret for antatt varmest/kaldest hittil
  rain = 0                  # total nedbør, hittil lik 0
  for i in range(len(weatherData)):     # antall runder for løkka = antall rader i tabell
    rain += weatherData[i][2]           # legger til dagens regn
    if weatherData[i][0] > htemp:
       htemp = weatherData[i][0]        # dagens temperatur er nå høyeste hittil
       hday = i + 1                     # NB: +1 for dagnummer! Indeks 0,… mens dagnummer 1, …
    if weatherData[i][1] < ltemp:
       ltemp = weatherData[i][1]        # dagens temperatur er nå laveste hittil
       lday = i + 1
# print-setningene uforandret...
```

3b: lignende problem…

- Kaldeste tredagersperiode
  - NB: løkka må bare gå til `len(weatherData)` **-2**
    - Ellers fins det ikke to neste dager å regne inn
  - ”flere perioder… like kalde… returnere kun den siste”
    - Må ha `averagetemp` **<=** `ctemp`

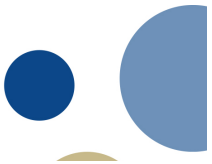
Oppgave 3b (10%)
Skriv en funksjon `coldestThreeDays` som tar inn parameteren `weatherData` (som definert over). Funksjonen skal finne den perioden av tre sammenhengende dager som hadde den laveste gjennomsnittlige minimumstemperaturen. Den skal returnere nummeret på første dagen i denne tredagersperioden. Dersom det er flere perioder som er like kalde, så skal den returnere kun den siste av disse periodene. Et eksempel på en kjøring av denne funksjonen for `weatherData` som definert tidligere i oppgaven gir:

```
>>> coldestThreeDays(weatherData)
2
>>>
```

Mulig løsning 3b:

```
def coldestThreeDays(weatherData): # Problem 3b
  ctemp=(weatherData[0][1]+weatherData[1][1]+weatherData[2][1])/3
  cday=1
  for i in range(1,len(weatherData)-2):
    averagetemp=(weatherData[i][1]+weatherData[i+1][1]+weatherData[i+2][1])/3
    if averagetemp<=ctemp:
      cday=i+1
      ctemp=averagetemp
  return cday
```

3e, aug. 2015



Oppgave 3e (6%)
Enkelte barn (og dels også foreldre) har sterke meninger om hvem som helst vil være på lag sammen, og dette kan være en kilde til konflikter. For å bli minst mulig påvirket av dette ønsker trenerne at programmet skal sette opp forslag til laginndeling automatisk, basert på en helt tilfeldig fordeling av spillere. Skriv en funksjon `laginndeling(spillere, sp_per_lag)` hvor innparameteren `spillere` er listen av de som er tilgjengelige for den aktuelle cupen, og `sp_per_lag` er antall spillere som man skal ha på banen. Funksjonen skal returnere en liste av lister, hvor den ytre lista inneholder de lagene man skal stille med i denne cupen, og de indre listene inneholder navn på spillerne på hvert av disse lagene. Alle lag må ha minst det antall spillere som innparameteren `sp_per_lag` angir, og ingen person kan være satt opp for å spille på mer enn ett lag. For øvrig skal funksjonen sette opp det maksimale antall lag man kan spille med (dvs. færrest mulig innbyttere), så det blir mest mulig spilletid på barna.

Eksempel på kjøring: 14 barn skal delta i en cup hvor hvert lag skal ha 4 på banen, funksjonen har tilfeldig fordelt barna i 3 lag, hvor 2 har 5 spillere (dvs. 1 innbytter) og ett har 4 spillere (ingen innbytter). Dette lagoppsettet returneres som en liste av lister.

```
>>> sp = ['Ada', 'Bo', 'Eli', 'Isa', 'Cindy', 'Henrik',  'Ine', 'Jo', 'Kim',
'Lucas', 'My', 'Noor',  'Ola', 'Pia']
>>> laginndeling(sp, 4)
[['Eli', 'Henrik', 'Pia', 'Ada', 'Ine'], ['Kim', 'My', 'Ola', 'Lucas', 'Noor'],
['Isa', 'Bo', 'Cindy', 'Jo']]
>>>
```