

# **Forelesning IN1900 – 31 August 2022**

**Ole Christian Lingjærde  
Institutt for informatikk, Universitetet i Oslo**

**Uke: 29 August – 4 September, 2022**

## Forrige forelesning på en foil

**Formatert utskrift:** %-operator og f-strings

**Lister:** kan brukes til å lagre mange verdier

**To typer løkker:** while-løkker og for-løkker

**For-løkke:** nyttig for å løpe gjennom verdiene i en liste

**While-løkke:** mest generelle løkketyper; løkketest

**Ikke noe absolutt skille:** kan ofte bruke begge løkketyper

- Mer om lister
- Tupler (= lister som ikke kan endres)
- Funksjonen zip
- Lister av lister (tabeller)
- Mange eksempler på bruk av løkker og lister

# Konstruksjon av lister

Lister brukes ofte i Python.

Vi brukler god tid både på forelesninger og øvingsgrupper på å lære hvordan de brukes.

Vi må lære å:

- Lage lister
- Hente ut verdier fra lister (alt eller deler)
- Forandre lister
- Bruke lister på en fornuftig måte

[Live programmering: lage lister](#)

# Oversikt over metoder for å lage lister

## Metode A: Angi listeelementer eksplisitt

```
a = [2, 3, 5, 7, 11, 13]
minliste = ['Kurs', 'i', 'programmering']
```

## Metode B: Angi et intervall

```
# Angi stopp:
a = list(range(10))           # [0, 1, 2, ..., 9]

# Angi start og stopp:
a = list(range(3, 10))       # [3, 4, 5, ..., 9]

# Angi start, stopp og steglengde:
a = list(range(3, 10, 2))    # [3, 5, 7, 9]

# Kan droppe list(..) hvis eneste bruk er for-løkke:
for i in range(10):
    print(i)                  # Skriver ut tallene 0-9
```

## Metode C: Angi startverdi og antall repetisjoner

```
a = [0]*10           # [0, 0, ..., 0] (10 elementer)
b = [1]*25          # [1, 1, ..., 1] (25 elementer)
c = ['Hei']*3       # ['Hei', 'Hei', 'Hei']
d = [0,1,2]*2       # [0, 1, 2, 0, 1, 2]
```

## Metode D: Angi en regel for å generere verdier

```
a = [i*i for i in range(5)]           # [0, 1, 4, 9, 16]
b = [10-2*j for j in range(5)]        # [10, 8, 6, 4, 2]
c = [e**2 for e in a]                  # [0, 1, 16, 81, 256]
d = [e**2 for e in a if e%2==0]        # [0, 16, 256]
e = [s[0] for s in ['Hei', 'paa', 'deg']] # ['H', 'p', 'd']
```

(Merk:  $e \% 2 == 0$  tester om  $e$  er et partall)

Foreslå en måte å lage denne listen på i Python:

```
a = [1,2,3,4]
```

Foreslå en måte å lage denne listen på i Python:

```
a = [1,2,3,4]
```

Svar:

```
a = [1, 2, 3, 4]
```



Foreslå en måte å lage denne listen på i Python:

```
a = [-2, -2, -2, . . . . , -2]
```

(listen skal ha lengde 100)

Foreslå en måte å lage denne listen på i Python:

```
a = [-2, -2, -2, . . . . , -2]
```

(listen skal ha lengde 100)

**Svar:**

```
a = [-2] * 100
```

Foreslå en måte å lage denne listen på i Python:

```
a = [1, 2, 3, ..., 9999]
```

Foreslå en måte å lage denne listen på i Python:

```
a = [1, 2, 3, ..., 9999]
```

Svar:

```
a = list(range(1,10000))
```

Foreslå en måte å lage denne listen på i Python:

```
a = [2, 4, 6, 8, ....., 2000]
```

Foreslå en måte å lage denne listen på i Python:

```
a = [2, 4, 6, 8, ....., 2000]
```

Svar:

```
a = list(range(2,2001,2))
```

Foreslå en måte å lage denne listen på i Python:

```
a = [999, 998, 997, ..., 1]
```

Foreslå en måte å lage denne listen på i Python:

```
a = [999, 998, 997, ..., 1]
```

Svar:

```
a = list(range(999, 0, -1))
```



Foreslå en måte å lage denne listen på i Python:

```
a = [0, 1, 0, 1, ....., 0, 1]
```

(listen skal ha lengde 100)

Foreslå en måte å lage denne listen på i Python:

```
a = [0, 1, 0, 1, ....., 0, 1]
```

(listen skal ha lengde 100)

**Svar:**

```
a = [0,1] * 50
```

Foreslå en måte å lage denne listen på i Python:

```
a = [1*2, 2*3, 3*4, 4*5, ..., 999*1000]
```

Foreslå en måte å lage denne listen på i Python:

```
a = [1*2, 2*3, 3*4, 4*5, ..., 999*1000]
```

Svar:

```
a = [i*(i+1) for i in range(1,1000)]
```

Hva blir resultatet?

$$a = [-1]*3 + [0]*3$$

Hva blir resultatet?

$$a = [-1]*3 + [0]*3$$

Svar:

$$a : [-1, -1, -1, 0, 0, 0]$$

Hva blir resultatet?

```
b = list(range(-1,4))
```

Hva blir resultatet?

```
b = list(range(-1,4))
```

Svar:

```
b : [-1, 0, 1, 2, 3]
```



Hva blir resultatet?

```
c = [k-1 for k in [1,2,3]]
```

Hva blir resultatet?

```
c = [k-1 for k in [1,2,3]]
```

Svar:

```
c : [0, 1, 2]
```

Hva blir resultatet?

```
d = [k**3 for k in range(1,4)]
```

Hva blir resultatet?

```
d = [k**3 for k in range(1,4)]
```

Svar:

```
d : [1, 8, 27]
```

Hva blir resultatet?

```
e = [1, 3, 5]
```

```
f = [e[i-1]+e[i] for i in range(0,3)]
```

Hva blir resultatet?

```
e = [1, 3, 5]  
f = [e[i-1]+e[i] for i in range(0,3)]
```

Svar:

f : [6, 4, 8]

# Beregninger av følger

Løkker kan brukes til å regne ut følger, f.eks. Fibonacci-følgen:

$$F_0 = 1$$

$$F_1 = 1$$

$$F_k = F_{k-2} + F_{k-1} \quad k = 2, 3, \dots$$

Vi regner ut de første leddene:

$$F_0 = 1$$

$$F_1 = 1$$

$$F_2 = 1 + 1 = 2$$

$$F_3 = 1 + 2 = 3$$

$$F_4 = 2 + 3 = 5$$

$$F_5 = 3 + 5 = 8$$

...

[Live programmering: Fibonacci-følgen](#)

# Bruk av løkke for å beregne Fibonacci-rekken

## Løsning A:

```
F = [1, 1]
for k in range(2, 100):
    F.append(F[k-1] + F[k-2])
print(F)
```

## Løsning B:

```
F = [1]*100
for k in range(2, 100):
    F[k] = F[k-2] + F[k-1]
print(F)
```

Spørsmål: Løsning A endrer størrelsen på listen F underveis. Kan du se noen potensielle ulemper med det?



## Beregning av summer (rekker)

Den harmoniske rekken (med 100 ledd) er:

$$S = \sum_{i=1}^{100} \frac{1}{i} = \frac{1}{1} + \frac{1}{2} + \dots + \frac{1}{100}$$

Denne kan vi lett beregne med en løkke.

[Live programmering: harmonisk rekke](#)

# Flere alternative løsninger

## Alternativ A: while-løkke

```
s = 0    # Startverdi
i = 1    # Teller
while i <= 100:
    s += 1/i
    i = i + 1
print(f"s = {s:5.2f}")
```

## Alternativ B: for-løkke

```
s = 0
for i in range(1, 101):
    s += 1/i
print(f"s = {s:5.2f}")
```

## Alternativ C: implisitt løkke

```
s = sum(1/i for i in range(1,101))
print(f"s = {s:5.2f}")
```

## Å hente ut verdier fra en liste

Vi kan plukke ut en enkelt verdi eller mange verdier samtidig fra en liste.

```
# Lag en liste  
a = [5, 10, 15, 20, 25, 30]  
  
# Plukke ut første del av listen:  
b = a[:4]      # [5, 10, 15, 20] (indeks 0-3)  
  
# Plukke ut siste del av listen:  
c = a[3:]      # [20, 25, 30] (indeks 3-5)  
  
# Plukke ut midtparti av listen:  
d = a[3:5]     # [20, 25] (indeks 3-4)  
  
# Alle elementer unntatt første og siste:  
e = a[1:-1]    # (indeks 1-4)  
  
# Alle elementer unntatt tre siste:  
f = a[:-3]     # (indeks 0-2)  
  
# Lage en kopi av hele listen:  
g = a[:]       # (indeks 0-5)
```

Hva blir resultatet i hvert av disse tilfellene?

```
a = [0, 2, 4, 6, 8, 10, 12, 14]
```

```
x = a[3]
```

```
# Svar:
```

```
x = a[3:4]
```

```
# Svar:
```

```
x = a[3:3]
```

```
# Svar:
```

```
x = a[0:2] + a[3:5]
```

```
# Svar:
```

```
x = [2*e for e in a[0:2]]
```

```
# Svar:
```

```
x = a[-2:2]
```

```
# Svar:
```

Hva blir resultatet i hvert av disse tilfellene?

```
a = [0, 2, 4, 6, 8, 10, 12, 14]
```

```
x = a[3]
```

```
# Svar: x = 6
```

```
x = a[3:4]
```

```
# Svar:
```

```
x = a[3:3]
```

```
# Svar:
```

```
x = a[0:2] + a[3:5]
```

```
# Svar:
```

```
x = [2*e for e in a[0:2]]
```

```
# Svar:
```

```
x = a[-2:2]
```

```
# Svar:
```

Hva blir resultatet i hvert av disse tilfellene?

```
a = [0, 2, 4, 6, 8, 10, 12, 14]
```

```
x = a[3]
```

```
# Svar: x = 6
```

```
x = a[3:4]
```

```
# Svar: x = [6]
```

```
x = a[3:3]
```

```
# Svar:
```

```
x = a[0:2] + a[3:5]
```

```
# Svar:
```

```
x = [2*e for e in a[0:2]]
```

```
# Svar:
```

```
x = a[-2:2]
```

```
# Svar:
```

Hva blir resultatet i hvert av disse tilfellene?

```
a = [0, 2, 4, 6, 8, 10, 12, 14]
```

```
x = a[3]
```

```
# Svar: x = 6
```

```
x = a[3:4]
```

```
# Svar: x = [6]
```

```
x = a[3:3]
```

```
# Svar: x = []
```

```
x = a[0:2] + a[3:5]
```

```
# Svar:
```

```
x = [2*e for e in a[0:2]]
```

```
# Svar:
```

```
x = a[-2:2]
```

```
# Svar:
```

Hva blir resultatet i hvert av disse tilfellene?

```
a = [0, 2, 4, 6, 8, 10, 12, 14]
```

```
x = a[3]
```

```
# Svar: x = 6
```

```
x = a[3:4]
```

```
# Svar: x = [6]
```

```
x = a[3:3]
```

```
# Svar: x = []
```

```
x = a[0:2] + a[3:5]
```

```
# Svar: x = [0, 2, 6, 8]
```

```
x = [2*e for e in a[0:2]]
```

```
# Svar:
```

```
x = a[-2:2]
```

```
# Svar:
```



Hva blir resultatet i hvert av disse tilfellene?

```
a = [0, 2, 4, 6, 8, 10, 12, 14]
```

```
x = a[3]
```

```
# Svar: x = 6
```

```
x = a[3:4]
```

```
# Svar: x = [6]
```

```
x = a[3:3]
```

```
# Svar: x = []
```

```
x = a[0:2] + a[3:5]
```

```
# Svar: x = [0, 2, 6, 8]
```

```
x = [2*e for e in a[0:2]]
```

```
# Svar: x = [0, 4]
```

```
x = a[-2:2]
```

```
# Svar:
```

Hva blir resultatet i hvert av disse tilfellene?

```
a = [0, 2, 4, 6, 8, 10, 12, 14]
```

```
x = a[3]
```

```
# Svar: x = 6
```

```
x = a[3:4]
```

```
# Svar: x = [6]
```

```
x = a[3:3]
```

```
# Svar: x = []
```

```
x = a[0:2] + a[3:5]
```

```
# Svar: x = [0, 2, 6, 8]
```

```
x = [2*e for e in a[0:2]]
```

```
# Svar: x = [0, 4]
```

```
x = a[-2:2]
```

```
# Svar: x = []
```

# En liste kan ha flere navn

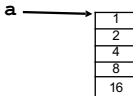
Det neste vi skal se på er litt vanskelig, og ikke regn med at du skjønner det 100% nå i starten!

Når vi lager en liste må vi skille mellom **selve listen** (ofte kalt liste-objektet) og **navnet** vi bruker for å referere til listen. En liste kan faktisk ha flere navn samtidig!

Etter at vi har utført.....

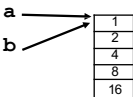
så er situasjonen slik:

`a = [1, 2, 4, 8, 16]`



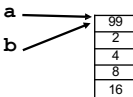
Ett dataobjekt,  
og ett navn a

`b = a`



Ett dataobjekt,  
og to navn a og b

`a[0] = 99`



Dataobjektet endres  
og både a og b «ser»  
denne endringen

## Hva betyr dette i praksis?

Når en liste i Python har flere navn, må vi huske på at disse refererer til den *samme listen*. Eksempel:

```
# Vi lager en liste med tre verdier
a = [1,2,4]

# Vi gir listen et ekstra navn:
b = a

# Vi endrer første verdien i b:
b[0] = 99

# Vi skriver ut innholdet av a og b:
print(f"a={a} og b={b}")
```

a=[99, 2, 4] og b=[99, 2, 4]

**Både a og b har blitt endret!**

## Å lage kopi av en liste

Det går an å lage en kopi av en liste - altså av selve listeobjektet. Da kan vi gjøre en endring i én av listene, uten at det forandrer den andre.

```
a = [1, 2, 4]
b = a[:]      ## Lag kopi av a
b[0] = 99     ## Nå endres b[0], men ikke a[0]
print(a)
print(b)

[1, 2, 4]
[99, 2, 4]
```

Vi ønsker å skrive ut kvadratroten av alle heltall mellom 0 og 99. Vi lager et program **kvadratrot.py**:

```
from math import sqrt

x = [0]*100           #x er liste av lengde 100
y = x                 #y er liste av lengde 100

for i in range(100):
    x[i] = i
    y[i] = sqrt(i)

# Skriv ut resultatet
for i in range(100):
    print(f"x = {x[i]:5.2f} --> sqrt(x) = {y[i]:5.2f}")
```

Vi kjører programmet på forrige slide:

```
> python kvadratrot.py  
  
x = 0.00 --> sqrt(x) = 0.00  
x = 1.00 --> sqrt(x) = 1.00  
x = 1.41 --> sqrt(x) = 1.41  
x = 1.73 --> sqrt(x) = 1.73  
x = 2.00 --> sqrt(x) = 2.00  
.....  
.....
```

*Dette er ikke det vi ønsket! Hva gikk galt??*

## Quiz (svar)

```
from math import sqrt
x = [0]*100
y = x          ## Nå er x og y to navn på samme liste!

for i in range(100):
    x[i] = i
    y[i] = sqrt(i)  ## Nå overskriver vi x[i]

# Skriv ut resultatet
for i in range(100):
    print(f"x = {x[i]:5.2f} --> sqrt(x) = {y[i]:5.2f}")
```



# Vi retter opp programmet

```
from math import sqrt

x = [0]*100
y = [0]*100          ## Nå er x og y uavhengige lister

for i in range(100):
    x[i] = i
    y[i] = sqrt(i)   ## Nå overskriver vi ikke x[i]

# Skriv ut resultatet
for i in range(100):
    print(f"x = {x[i]:5.2f} --> sqrt(x) = {y[i]:5.2f}")
```

Hvis vi er sikre på at vi ikke kommer til å endre innholdet kan tupler være alternativ til liste:

```
a = (0, 2, 4, 6)           # Tuppel med fire verdier
a = 0, 2, 4, 6           # Kan droppe parenteser
a[1:3]                    # Indeksering
(0,1)+(2,4,5)             # Slå sammen tupler
5 in a                    # Teste medlemskap
a[1] = 3                  # FEILMELDING
a.append(18)              # FEILMELDING
del a[0]                  # FEILMELDING
```

# Hva du må vite om tupler

Tupler er konstante og beskyttet mot feilaktige endringer

Tupler gir raskere beregninger

Tupler brukes mye i Python-programmer

I oppslagstabeller (dictionaries) kan tupler, men ikke lister, brukes som nøkler. Mer om dette senere i kurset!

# Løpe gjennom to lister samtidig

Tenk deg at vi har to lister A og B av samme lengde.

Vi kan gå gjennom dem samtidig som i dette eksemplet:

```
A = [1, 3, 5]
B = [1, 9, 25]
for i in range(len(A)):
    print(A[i], B[i])
```

og svaret blir da:

```
1 1
3 9
5 25
```

# Funksjonen zip

En alternativ metode som ofte er nyttig for å løpe gjennom flere lister samtidig, er å bruke zip:

```
A = [1, 3, 5]
B = [1, 9, 25]

for a,b in zip(A,B):
    print(a,b)
```

og svaret blir som før:

```
1 1
3 9
5 25
```

# Hva gjør zip?

Anta at vi har to eller flere lister av samme lengde:

```
a = [0,1,2]
b = [4,5,6]
c = [7,8,9]
```

`zip(a,b,c)` samler alle elementer med en gitt indeks i et tuppel. Etterpå kan vi løpe gjennom disse tuplene i en løkke, eller konvertere til en liste:

```
x = list(zip([0,1,2], [4,5,6], [7,8,9]))
```

# Nå er :

```
# x[0] lik tuplet (0,4,7)
# x[1] lik tuplet (1,5,8)
# x[2] lik tuplet (2,6,9)
```

## Lister av lister (tabeller)

Elementene i en liste kan selv være lister:

```
A = [[1, 3, 5], [2, 4, 6]]
```

```
# A[0] er listen [1, 3, 5]
```

```
# A[1] er listen [2, 4, 6]
```

```
# A[0][0] er 1
```

```
# A[0][1] er 3
```

```
# A[0][2] er 5
```

```
# A[1][0] er 2
```

```
# A[1][1] er 4
```

```
# A[1][2] er 6
```

Naturlig å tolke A i eksemplet over som en matrise/tabell:

$$A = \begin{pmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{pmatrix}$$

# Kort oppsummering av lister

Kommando	Mening
<code>a = []</code>	Lag en tom liste
<code>a = [1, 4.4, 'run.py']</code>	Lag en liste med angitte verdier
<code>a.append(4.4)</code>	Utvid listen a med verdien 4.4
<code>a + [1,3]</code>	Slå sammen (konkatenér) to lister
<code>a.insert(i, 99)</code>	Sett inn 99 i listen a slik at den får indeks i
<code>a[0]</code>	Verdien med indeks 0 (førsteplassen)
<code>a[2]</code>	Verdien med indeks 2 (tredjeplassen)
<code>a[-1]</code>	Verdien som ligger sist i listen
<code>a[1:3]</code>	Ny liste med to elementer a[1], a[2]
<code>del a[3]</code>	Fjern elementet med indeks 3
<code>a.index(99)</code>	Finn første indeks som inneholder verdien 99
<code>99 in a</code>	Er verdien 99 i listen a? (True eller False)
<code>a.count(99)</code>	Hvor mange ganger forekommer 99 i listen?
<code>len(a)</code>	Hvor lang er listen a?
<code>min(a)</code>	Minste element i listen a
<code>max(a)</code>	Største element i listen a
<code>sum(a)</code>	Beregn summen av alle elementer i a
<code>sorted(a)</code>	Returmer en sortert versjon av a
<code>reversed(a)</code>	Returmer en reversert versjon av a
<code>isinstance(a, list)</code>	Er a en liste? (True eller False)
<code>type(a) is list</code>	Er a en liste? (True eller False)



## **Exercise 1.4: Convert from meters to British length units**

Make a program where you set a length given in meters and then compute and write out the corresponding length measured in inches, in feet, in yards, and in miles. Use that one inch is 2.54 cm, one foot is 12 inches, one yard is 3 feet, and one British mile is 1760 yards. For verification, a length of 640 meters corresponds to 25196.85 inches, 2099.74 feet, 699.91 yards, or 0.3977 miles.

Filename: `length_conversion`.

## Exercise 1.4: Convert from meters to British length units

Make a program where you set a length given in meters and then compute and write out the corresponding length measured in inches, in feet, in yards, and in miles. Use that one inch is 2.54 cm, one foot is 12 inches, one yard is 3 feet, and one British mile is 1760 yards. For verification, a length of 640 meters corresponds to 25196.85 inches, 2099.74 feet, 699.91 yards, or 0.3977 miles.

Filename: `length_conversion`.

For å konvertere fra  $L\_meter$  til  $L\_inch$ , kan vi bruke formelen

$$L\_inch = L\_meter * 100 / 2.54$$

For å konvertere fra  $L\_inch$  til  $L\_foot$  kan vi bruke

$$L\_foot = L\_inch / 12$$

OSV.

[Live programming](#)

# Plenumsoppgaver

## Exercise 1.12: How to cook the perfect egg

As an egg cooks, the proteins first denature and then coagulate. When the temperature exceeds a critical point, reactions begin and proceed faster as the temperature increases. In the egg white, the proteins start to coagulate for temperatures above  $63^\circ\text{C}$ , while in the yolk the proteins start to coagulate for temperatures above  $70^\circ\text{C}$ . For a soft boiled egg, the white needs to have been heated long enough to coagulate at a temperature above  $63^\circ\text{C}$ , but the yolk should not be heated above  $70^\circ\text{C}$ . For a hard boiled egg, the center of the yolk should be allowed to reach  $70^\circ\text{C}$ .

The following formula expresses the time  $t$  it takes (in seconds) for the center of the yolk to reach the temperature  $T_y$  (in Celsius degrees):

$$t = \frac{M^{2/3} c \rho^{1/3}}{K \pi^2 (4\pi/3)^{2/3}} \ln \left[ 0.76 \frac{T_o - T_w}{T_y - T_w} \right]. \quad (1.9)$$

Here,  $M$ ,  $\rho$ ,  $c$ , and  $K$  are properties of the egg:  $M$  is the mass,  $\rho$  is the density,  $c$  is the specific heat capacity, and  $K$  is thermal conductivity. Relevant values are  $M = 47$  g for a small egg and  $M = 67$  g for a large egg,  $\rho = 1.038$  g cm $^{-3}$ ,  $c = 3.7$  J g $^{-1}$  K $^{-1}$ , and  $K = 5.4 \cdot 10^{-3}$  W cm $^{-1}$  K $^{-1}$ . Furthermore,  $T_w$  is the temperature (in C degrees) of the boiling water, and  $T_o$  is the original temperature (in C degrees) of the egg before being put in the water. Implement the formula in a program, set  $T_w = 100^\circ\text{C}$  and  $T_y = 70^\circ\text{C}$ , and compute  $t$  for a large egg taken from the fridge ( $T_o = 4^\circ\text{C}$ ) and from room temperature ( $T_o = 20^\circ\text{C}$ ).

Filename: `egg`.

# Plenumsoppgaver

## Exercise 1.12: How to cook the perfect egg

As an egg cooks, the proteins first denature and then coagulate. When the temperature exceeds a critical point, reactions begin and proceed faster as the temperature increases. In the egg white, the proteins start to coagulate for temperatures above 63 °C, while in the yolk the proteins start to coagulate for temperatures above 70 °C. For a soft boiled egg, the white needs to have been heated long enough to coagulate at a temperature above 63 °C, but the yolk should not be heated above 70 °C. For a hard boiled egg, the center of the yolk should be allowed to reach 70 °C.

The following formula expresses the time  $t$  it takes (in seconds) for the center of the yolk to reach the temperature  $T_y$  (in Celsius degrees):

$$t = \frac{M^{2/3} c \rho^{1/3}}{K \pi^2 (4\pi/3)^{2/3}} \ln \left[ 0.76 \frac{T_o - T_w}{T_y - T_w} \right]. \quad (1.9)$$

Here,  $M$ ,  $\rho$ ,  $c$ , and  $K$  are properties of the egg:  $M$  is the mass,  $\rho$  is the density,  $c$  is the specific heat capacity, and  $K$  is thermal conductivity. Relevant values are

$M = 47$  g for a small egg and  $M = 67$  g for a large egg,  $\rho = 1.038$  g cm<sup>-3</sup>,  $c = 3.7$  J g<sup>-1</sup> K<sup>-1</sup>, and  $K = 5.4 \cdot 10^{-5}$  W cm<sup>-1</sup> K<sup>-1</sup>. Furthermore,  $T_w$  is the temperature

(in C degrees) of the boiling water, and  $T_o$  is the original temperature (in C degrees) of the egg before being put in the water. Implement the formula in a program, set

$T_w = 100$  °C and  $T_y = 70$  °C, and compute  $t$  for a large egg taken from the fridge ( $T_o = 4$  °C) and from room temperature ( $T_o = 20$  °C).

Filename: egg.

# Plenumsoppgaver

## Exercise 1.12: How to cook the perfect egg

As an egg cooks, the proteins first denature and then coagulate. When the temperature exceeds a critical point, reactions begin and proceed faster as the temperature

NB: sjekk alltid at enhetene er de samme overalt i formelen før du regner ut. Konverter om nødvendig!

F.eks. kan vi ikke ha

- $km$  ett sted og  $m^3$  et annet sted
- $\text{år}$  ett sted og  $s^{-1}$  et annet sted

## Live programmering

$c$  is the specific heat capacity, and  $\kappa$  is thermal conductivity. Relevant values are  $M = 47$  g for a small egg and  $M = 67$  g for a large egg  $\rho = 1.038 \text{ g cm}^{-3}$   $c = 3.7 \text{ J g}^{-1} \text{ K}^{-1}$ , and  $K = 5.4 \cdot 10^{-5} \text{ W cm}^{-1} \text{ K}^{-1}$ . Furthermore,  $T_w$  is the temperature (in C degrees) of the boiling water, and  $T_o$  is the original temperature (in C degrees) of the egg before being put in the water. Implement the formula in a program, set  $T_w = 100^\circ\text{C}$  and  $T_y = 70^\circ\text{C}$ , and compute  $t$  for a large egg taken from the fridge ( $T_o = 4^\circ\text{C}$ ) and from room temperature ( $T_o = 20^\circ\text{C}$ ).

Filename: egg.