Kap 3: Funksjoner og if-tester

Ole Christian Lingjærde, Institutt for Informatikk, UiO

31 August - 4 September, 2020 (Del 2 av 2)

Forrige forelesning på en foil

- **Lister:** kan brukes til å lage 1D-, 2D- og 3D-tabeller
- **Funksjoner:** brukes til å splitte programmer i mindre biter
- **Kall:** kalle på funksjon = anvende funksjonen
- **Metoder:** funksjoner som kalles med `.`-notasjon
- **Argumenter og returverdier:** input og output
- **Navngitte argumenter:** i `f(x=0.5)` er `x` navngitt
- **Lokale variabler:** variabler definert inni funksjoner er lokale

Tema i dag

- Mer om funksjoner
- If-tester
- Assert-tester
- Testfunksjoner

Kort repetisjon av funksjoner

Funksjon uten input:

```
# Vi definerer en funksjon:
def skrivut():
    print("Pi is approximately 3.1415926")

# Funksjonen utføres først nå vi kaller på den:
skrivut()
```

Funksjon med input:

```
# Vi definerer en funksjon:
def f(x):
    return x**2

# Vi kaller på funksjonen:
y = f(x)
print(y)
```

Eksempel 1

Funksjon som skriver ut gjennomsnittsverdi:

```
def printsnitt(a):
    # Vi regner først ut summen av alle elementene i a:
    total = 0
    for e in a:
        total = total + e

    # Vi deler på antall elementer:
    snittverdi = total/len(a)

    # Vi skriver ut svaret:
    print(f"Gjennomsnittsverdien er {snittverdi:g}")

# Vi prøver funksjonen på en liste:
x = [3.6, 7.2, 9.3, -5.6, 0.1]
printsnitt(x)
```

Vi prøver programmet:

```
> python eksempel1.py
Gjennomsnittsverdien er 2.92
Gjennomsnittsverdien er 2
```

Eksempel 2

Funksjon som skriver ut en melding:

```
def skrivMelding(navn, melding):
    s = f"Gratulerer {navn}, du har {melding}!"
    print(s)

# Bruk funksjonen:
skrivMelding("Mia", "vunnet 100 kroner")
skrivMelding("Jon", "bestått eksamen")
```

Vi prøver programmet:

```
> python eksempel2.py
Gratulerer Mia, du har vunnet 100 kroner!
Gratulerer Jon, du har bestått eksamen!
```

Eksempel 3

Funksjon som regner ut indreprodukt:

```
def innerprod(x, y):
    verdi = 0
    for i in range(len(x)):
        verdi += x[i]*y[i]
    return verdi

# Prøv funksjonen:
x = [1, 2, 3, 4]
y = [0, 1, 4, 2]
svar = innerprod(x,y)
print(f"Indreproduktet av x og y er {svar:5.2f}")
```

Kjøreresultat:

```
> python eksempel3.py
Indreproduktet av x og y er 22.00
```

Eksempel 4

Funksjon som beregner $n! = 1 \cdot 2 \cdot 3 \cdots n$:

```
def factorial(n):
    v = 1
    for i in range(1,n+1):
        v = v * i
    return v

# Prøv funksjonen:
for n in range(1,8):
    print(f"Fakultet av {n} er {factorial(n)}")
```

Vi kjører programmet:

```
> python eksempel4.py
Fakultet av 1 er 1
Fakultet av 2 er 2
Fakultet av 3 er 6
Fakultet av 4 er 24
Fakultet av 5 er 120
Fakultet av 6 er 720
Fakultet av 7 er 5040
```

Globale variabler

- Variabler definert utenfor funksjoner kalles globale.
- Globale variabler er også synlige inni funksjoner.
- Eksempel:

```
def skrivut():
    print(f"alpha = {alpha:g}")

alpha = 10        # Global variabel
skrivut()         # Utskrift: alpha = 10
```

Lokale variabler

- Variabler definert inni funksjoner kalles lokale.
- De er kun synlige inni funksjonen.
- De eksisterer bare når funksjonen utføres!
- Eksempel:

```
def skrivut():
    alpha = 0.6323                  # LOKAL variabel
    print(alpha)                    # Skriver ut alpha

skrivut()                           # Utskrift: 0.6323
print(alpha)                        # FEILMELDING
```

Lokale variabler kan skjule globale variabler

- Vær varsom hvis en lokal og global variabel har samme navn!
- Bare den lokale variabelen er synlig inni funksjonen
- Bare den globale variabelen er synlig utenfor funksjonen
- Eksempel:

```
def g(t):
    alpha = 1.0                 # LOKAL variabel
    return alpha * t

alpha = 100000                  # GLOBAL variabel
print(g(2))                     # Utskrift: 2.0
```

Et litt større eksempel

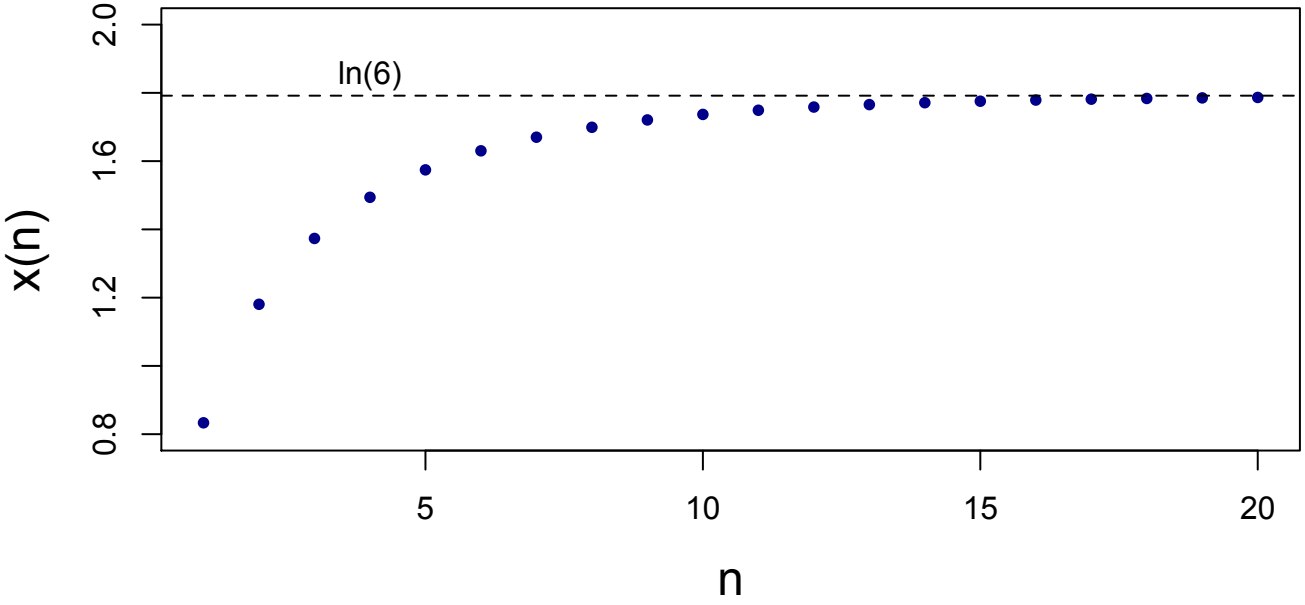
Denne følgen approksimerer den naturlige logaritmen til 6 ($\ln 6$):

$$x(n) = \sum_{i=1}^{n} \frac{1}{i} \left(\frac{5}{6}\right)^{i}$$

Eksempel: $\ln 6 = 1.791759...$, mens

$$\begin{aligned}
x(1) &= 0.8333333 \\
x(2) &= 1.180556 \\
x(3) &= 1.373457 \\
x(4) &= 1.49402 \\
x(5) &= 1.574396 \\
&\cdots \\
x(20) &= 1.786568 \\
&\cdots \\
x(100) &= 1.791759
\end{aligned}$$

Visuelt: x(n) for n=1,2,...,20



Implementasjon i Python

Python-funksjon som implementerer $x(n)$:

```
def x(n):
    s = 0
    for i in range(1, n+1):
        s += (1/i) * (5/6)**i
    return s
```

Hvor forskjellig er $x(n)$ fra $\ln 6$?

- Anta at vi ønsker å finne $\ln 6$ med stor nøyaktighet
- Hvor stor må vi velge n?
- Visuelt ser vi at $n > 20$ gir ganske nøyaktig svar
- Kan vi si noe mer presis om estimeringsfeilen?

To forslag:

- Regne ut feilen $\ln(6) - x(n)$.
- Regne ut første ledd i summen som *ikke* er tatt med

Løsning 1 er best, men da må vi regne ut $\ln(..)$. Hvis vi skal unngå å bruke fasitsvaret, er løsning 2 et alternativ.

Utvidelse av x(n)

```python
from math import log

def x(n):
    s = 0
    for i in range(1, n+1):
        s += (1/i) * (5/6)**i

    feil1 = log(6) - s
    feil2 = (1/(n+1)) * (5/6)**(n+1)

    return s, feil1, feil2

# Typisk kall:
value, feil1, feil2 = x(100)

# Resultat av kjøring:
#   value: 1.7917594686571028
#   feil1: 5.7095217442793e-10
#   feil2: 9.962601874928813e-11
```

Default-verdier for argumenter

Når vi definerer funksjoner kan vi oppgi default-verdier for argumentene. Eksempel:

```
def skrivut(x, y, z=1, w=2.5):
    print(x, y, z, w)
```

Kall på funksjoner hvor argumenter har default-verdier

```
def skrivut(x, y, z=0, t=99):
   print (x, y, z, t)

# Benytt default-verdiene for z og t:
skrivut("Hei", 3)                  # Utskrift: Hei 3 0 99

# Benytt default-verdien for t:
skrivut("Hei", 3, z="XX")          # Utskrift: Hei 3 XX 99

# Benytt default-verdien for z:
skrivut("Hei", 3, t="YY")          # Utskrift: Hei 3 0 YY

# Ikke benytt noen default-verdier:
skrivut("Hei", 3, z="XX", t="YY") # Utskrift: Hei 3 XX YY
```

Quiz 1

Tenk deg at vi har laget funksjonen

```
def h(x, y, z=0):
    import math
    res = x * math.sin(y) + z
    return res
```

Hvilke av disse funksjonskallene er da lovlige?

```
r = h(0)
r = h(0, 1)
r = h(0, 1, 2)
r = h(x=0, 1, 2)
r = h(0, y=1)
r = h(0, 1, z=3)
r = h(0, 0, x=0)
r = h(z=0, x=1)
r = h(z=0, x=1, y=2)
```

Quiz 1

Tenk deg at vi har laget funksjonen

```
def h(x, y, z=0):
    import math
    res = x * math.sin(y) + z
    return res
```

Hvilke av disse funksjonskallene er da lovlige?

```
r = h(0)                    # Ikke lovlig (mangler y)
r = h(0, 1)
r = h(0, 1, 2)
r = h(x=0, 1, 2)
r = h(0, y=1)
r = h(0, 1, z=3)
r = h(0, 0, x=0)
r = h(z=0, x=1)
r = h(z=0, x=1, y=2)
```

Quiz 1

Tenk deg at vi har laget funksjonen

```
def h(x, y, z=0):
    import math
    res = x * math.sin(y) + z
    return res
```

Hvilke av disse funksjonskallene er da lovlige?

```
r = h(0)                    # Ikke lovlig
r = h(0, 1)                 # Lovlig
r = h(0, 1, 2)
r = h(x=0, 1, 2)
r = h(0, y=1)
r = h(0, 1, z=3)
r = h(0, 0, x=0)
r = h(z=0, x=1)
r = h(z=0, x=1, y=2)
```

Quiz 1

Tenk deg at vi har laget funksjonen

```
def h(x, y, z=0):
    import math
    res = x * math.sin(y) + z
    return res
```

Hvilke av disse funksjonskallene er da lovlige?

```
r = h(0)                # Ikke lovlig
r = h(0, 1)             # Lovlig
r = h(0, 1, 2)          # Lovlig
r = h(x=0, 1, 2)
r = h(0, y=1)
r = h(0, 1, z=3)
r = h(0, 0, x=0)
r = h(z=0, x=1)
r = h(z=0, x=1, y=2)
```

Quiz 1

Tenk deg at vi har laget funksjonen

```
def h(x, y, z=0):
    import math
    res = x * math.sin(y) + z
    return res
```

Hvilke av disse funksjonskallene er da lovlige?

```
r = h(0)                # Ikke lovlig
r = h(0, 1)             # Lovlig
r = h(0, 1, 2)          # Lovlig
r = h(x=0, 1, 2)        # Ikke lovlig (navngitte må komme sist)
r = h(0, y=1)
r = h(0, 1, z=3)
r = h(0, 0, x=0)
r = h(z=0, x=1)
r = h(z=0, x=1, y=2)
```

Quiz 1

Tenk deg at vi har laget funksjonen

```
def h(x, y, z=0):
    import math
    res = x * math.sin(y) + z
    return res
```

Hvilke av disse funksjonskallene er da lovlige?

```
r = h(0)                # Ikke lovlig
r = h(0, 1)             # Lovlig
r = h(0, 1, 2)          # Lovlig
r = h(x=0, 1, 2)        # Ikke lovlig
r = h(0, y=1)           # Lovlig
r = h(0, 1, z=3)
r = h(0, 0, x=0)
r = h(z=0, x=1)
r = h(z=0, x=1, y=2)
```

Quiz 1

Tenk deg at vi har laget funksjonen

```
def h(x, y, z=0):
    import math
    res = x * math.sin(y) + z
    return res
```

Hvilke av disse funksjonskallene er da lovlige?

```
r = h(0)                # Ikke lovlig
r = h(0, 1)             # Lovlig
r = h(0, 1, 2)          # Lovlig
r = h(x=0, 1, 2)        # Ikke lovlig
r = h(0, y=1)           # Lovlig
r = h(0, 1, z=3)        # Lovlig
r = h(0, 0, x=0)
r = h(z=0, x=1)
r = h(z=0, x=1, y=2)
```

Quiz 1

Tenk deg at vi har laget funksjonen

```
def h(x, y, z=0):
    import math
    res = x * math.sin(y) + z
    return res
```

Hvilke av disse funksjonskallene er da lovlige?

```
r = h(0)                # Ikke lovlig
r = h(0, 1)             # Lovlig
r = h(0, 1, 2)          # Lovlig
r = h(x=0, 1, 2)        # Ikke lovlig
r = h(0, y=1)           # Lovlig
r = h(0, 1, z=3)        # Lovlig
r = h(0, 0, x=0)        # Ikke lovlig (x angis to ganger)
r = h(z=0, x=1)
r = h(z=0, x=1, y=2)
```

Quiz 1

Tenk deg at vi har laget funksjonen

```
def h(x, y, z=0):
    import math
    res = x * math.sin(y) + z
    return res
```

Hvilke av disse funksjonskallene er da lovlige?

```
r = h(0)                # Ikke lovlig
r = h(0, 1)             # Lovlig
r = h(0, 1, 2)          # Lovlig
r = h(x=0, 1, 2)        # Ikke lovlig
r = h(0, y=1)           # Lovlig
r = h(0, 1, z=3)        # Lovlig
r = h(0, 0, x=0)        # Ikke lovlig
r = h(z=0, x=1)         # Ikke lovlig (y mangler)
r = h(z=0, x=1, y=2)
```

Quiz 1

Tenk deg at vi har laget funksjonen

```
def h(x, y, z=0):
    import math
    res = x * math.sin(y) + z
    return res
```

Hvilke av disse funksjonskallene er da lovlige?

```
r = h(0)                # Ikke lovlig
r = h(0, 1)             # Lovlig
r = h(0, 1, 2)          # Lovlig
r = h(x=0, 1, 2)        # Ikke lovlig
r = h(0, y=1)           # Lovlig
r = h(0, 1, z=3)        # Lovlig
r = h(0, 0, x=0)        # Ikke lovlig
r = h(z=0, x=1)         # Ikke lovlig
r = h(z=0, x=1, y=2)    # Lovlig
```

Quiz 2

Hva skrives ut her?

```
def skrivut(k):
    x = k * 2
    print(f"x = {x:g}")

x = 5
print(f"x = {x:g}")
skrivut(5)
print(f"x = {x:g}")
```

Quiz 2

Hva skrives ut her?

```
def skrivut(k):
    x = k * 2
    print(f"x = {x:g}")

x = 5
print(f"x = {x:g}")      # x = 5
skrivut(5)               # x = 10
print(f"x = {x:g}")      # x = 5
```

Mer om funksjoner - et eksempel

Anta at vi har en funksjon av t, med parametre A, a og ω:

$$f(t; A, a, \omega) = Ae^{-at}\sin(\omega t)$$

Mulige implementasjoner i Python:

```
from math import pi, exp, sin

def f(t, A=1, a=1, omega=2*pi):
    return A*exp(-a*t)*sin(omega*t)
```

Her har A, a og ω default-verdier. Det gir oss mange muligheter når vi kaller på funksjonen:

```
v1 = f(0.2)                          # Bare oppgi t
v2 = f(0.2, omega=1)                 # Endre omega
v3 = f(0.2, omega=1, A=2.5)          # Endre omega og A
v4 = f(A=5, a=0.1, omega=1, t=1.3)   # Endre alle parametre
v5 = f(0.2, 1, 2.5)                  # Endre A og a
```

Funksjoner kan ha funksjoner som argumenter

I Python kan argumenter til en funksjon selv være funksjoner.

Eksempel: Hvis vi vet hvordan vi skal beregne $f(x)$, kan vi estimere den annenderiverte i et gitt punkt x med formelen:

$$f''(x) \approx \frac{f(x-h) - 2f(x) + f(x+h)}{h^2}$$

Python-implementasjon:

```
def diff2(f, x, h=1E-6):
    r = (f(x-h) - 2*f(x) + f(x+h))/float(h*h)
    return r
```

Første argument til `diff2(.)` er her den funksjonen vi skal finne den annenderiverte til.

En liten digresjon

Funksjonen vi nettopp definerte, har nøkkelord-argumentet `h=1E-6`. Er det en grunn til å velge $h = 0.000001$ i stedet for en lavere eller høyere verdi?

- Matematisk forventer vi at approksimasjonen blir bedre når h blir mindre.
- Når vi løser problemer numerisk, må vi også ta hensyn til avrundingsfeil.
- Noen numeriske problemer er mer følsomme for avrundingsfeil enn andre, så i praksis blir det ofte litt prøving og feiling for å finne en god h-verdi.

Effekten av å endre h

For å studere effekten av å endre h skriver vi et lite program:

```
def diff2(f, x, h=1E-6):
    r = (f(x-h) - 2*f(x) + f(x+h))/float(h*h)
    return r

def g(t):
    return t**(-6)

# Beregn g''(t) for mindre og mindre verdier av h:
for k in range(1,14):
    h = 10**(-k)
    print (f"h = {h:.0e}: {diff2(g, 1, h):.5f}")
```

Output ($g''(1) = 42$)

```
h = 1e-01: 44.61504
h = 1e-02: 42.02521
h = 1e-03: 42.00025
h = 1e-04: 42.00000
h = 1e-05: 41.99999
h = 1e-06: 42.00074
h = 1e-07: 41.94423
h = 1e-08: 47.73959
h = 1e-09: -666.13381
h = 1e-10: 0.00000
h = 1e-11: 0.00000
h = 1e-12: -666133814.77509
```

For små h-verdier dominerer avrundingsfeil

For $h < 10^{-8}$ er resultatene helt feil!

- **Problem 1:** for små h subtraherer vi tall som er nesten like and dette gir opphav til avrundingsfeil.
- **Problem 2:** for små h deler vi avrundingsfeilen på et veldig lite tall (h^2), og dette forsterker feilen.

Mulig løsning: bruke desimaltall med flere sifre

- Python har en (langsom) flyttalls datatype (`decimal.Decimal`) hvor antall sifre er vilkårlig stort.
- Bruk av 25 sifre gir nøyaktige resultater for $h \leq 10^{-13}$

I praksis er det sjelden behov for høyere nøyaktighet.

Funksjoner vs hovedprogram

Hovedprogrammet er den delen av programmet som ikke ligger inni noen funksjon. Generelt:

- Programeksekveringen starter med første programsetning i hovedprogrammet og fortsetter linje for linje, fra topp til bunn.
- Funksjoner utføres bare når man kaller på dem.

Merk: funksjoner kan kalles fra hovedprogrammet *eller* fra en funksjon. Dette kan noen ganger føre til lange "kjeder" av funksjonskall.

Lambda-funksjoner

Noen ganger trenger vi å lage funksjoner som bare beregner enkle uttrykk. Da er *lambda-funksjoner* ofte et nyttig alternativ til vanlige funksjonsdefinisjoner.

Eksempel: funksjonen

```
def f(x,y):
    return x**2 - y**2
```

kan defineres på én linje med lambda-konstruksjonen:

```
f = lambda x, y: x**2 - y**2
```

Lambdafunksjoner kan brukes direkte som argumenter i funksjoner:

```
z = g(lambda x, y: x**2 - y**2, 4)
```

Funksjoner bør dokumenteres

For å legge til en kort beskrivelse av en funksjon lager vi en *doc string* som plasseres rett etter funksjons-headeren og inni triple anførselstegn.

Eksempler:

```
def C2F(C):
    """Konverter Celsiusgrader (C) til Fahrenheit."""
    return (9.0/5)*C + 32

def line(x0, y0, x1, y1):
    """
    Beregn koeffisientene a og b i uttrykket for en rett
    linje y = a*x + b som passerer gjennom (x0,y0) og (x1,y1)

    x0, y0: første punkt (floats).
    x1, y1: andre punkt (floats).
    return: a, b (floats) for linjen (y=a*x+b).
    """
    a = (y1 - y0)/(x1 - x0)
    b = y0 - a*x0
    return a, b
```

If-tester

If-tester gjør det mulig å lage forgreninger i programkjøringen, dvs at ulike handlinger utføres under ulike betingelser. Eksempel:

$$f(x) = \begin{cases} \sin x, & 0 \leq x \leq \pi \\ 0, & \text{otherwise} \end{cases}$$

For å implementere f i Python, trenger vi å teste hvilken verdi x har for å vite hvilken handling som skal utføres:

```
from math import sin, pi

def f(x):
    if 0 <= x <= pi:
        return sin(x)
    else:
        return 0
```

Generell form på if-testen

Type 1 (if)

```
if betingelse:
    <setninger som eksekveres hvis betingelse==True>
```

Type 2 (if-else)

```
if betingelse:
    <setninger som eksekveres hvis betingelse==True>
else:
    <setninger som eksekveres hvis betingelse==False>
```

Type 3 (if-elif-else)

```
if betingelse1:
    <setninger>
elif betingelse2:
    <setninger>
elif betingelse3:
    <setninger>
else:
    <setninger>
```

Eksempel 1

Stykkevis definert funksjon:

$$N(x) = \begin{cases} 0, & x < 0 \\ x, & 0 \leq x < 1 \\ 2 - x, & 1 \leq x < 2 \\ 0, & x \geq 2 \end{cases}$$

Implementasjon med if-elif-else:

```
def N(x):
    if x < 0:
        return 0
    elif 0 <= x < 1:
        return x
    elif 1 <= x < 2:
        return 2 - x
    elif x >= 2:
        return 0
```

Eksempel 2

Funksjon som teller hvor mange ganger s forekommer i a:

```
def count(s, a):
    cnt = 0
    for e in a:
        if e == s:
            cnt += 1
    return cnt
```

Eksempel på bruk:

```
>>> count(5.3, [2.2, 6.6, 2.5, 5.3, 8.9, 5.3])
>>> 2
>>>
>>> count('Anna', ['Ola', 'Karianne', 'Anna', 'Jens'])
>>> 1
>>>
>>> count([1,2], [1, 5, [1,2], [1,2], 3])
>>> 2
```

Inline if-tester

Vanlig konstruksjon:

```
if betingelse:
    variable = verdi1
else:
    variable = verdi2
```

Mer kompakt syntaks med samme effekt:

```
variable = (verdi1 if betingelse else verdi2)
```

Eksempel:

```
def f(x):
    return (sin(x) if 0 <= x <= 2*pi else 0)
```

Oppgave 3.20 i Langtangen

Write functions

Three functions `hw1`, `hw2`, and `hw3` work as follows:

```
>>> print(hw1())
>>> Hello, World
>>>
>>> hw2()
>>> Hello, World
>>>
>>> print(hw3('Hello, ', 'World'))
>>> Hello, World
>>>
>>> print(hw3('Python ', 'function'))
>>> Python function
```

Write the three functions.

Filename: hw_func.

Oppgave 3.23 i Langtangen

Wrap a formula in a function

Implement the formula (1.9) from Exercise 1.12 in a Python function with three arguments: `egg(M, To=20, Ty=70)`.

$$t = \frac{M^{2/3} c \rho^{1/3}}{K \pi^2 (4\pi/3)^{2/3}} \ln \left[0.76 \frac{T_0 - T_w}{T_y - T_w} \right].$$

The parameters ρ, K, c, and Tw can be set as local (constant) variables inside the function. Let t be returned from the function. Compute t for these conditions:

- Soft (Ty $<$ 70) and hard boiled (Ty $>$ 70)
- Small (M = 47g) and large (M = 67g) egg
- Fridge (T0 = 4C) and hot room (T0 = 25C).

Filename: egg_func.

Oppgave 3.28 i Langtangen

Find the max and min elements in a list

Given a list a, the `max` function in Python's standard library computes the largest element in a: max(a). Similarly, min(a) returns the smallest element in a.

Write your own max and min functions.

Hint: Initialize a variable `max_elem` by the first element in the list, then visit all the remaining elements (`a[1:]`), compare each element to `max_elem`, and if greater, set `max_elem` equal to that element. Use a similar technique to compute the minimum element.

Filename: maxmin_list.

assert: en spesiell form for test

Noen ganger ønsker man å stoppe programkjøringen og gi en feilmelding dersom en bestemt betingelse ikke er oppfylt. For dette formål har vi `assert`. Generell form:

```
assert betingelse, melding
```

Eksempel:

```
>>> x = 5
>>> assert x > 0, "x maa vaere positiv"  # Ingenting skjer
>>> x = -5
>>> assert x > 0, "x maa vaere positiv"  # Gir feilmelding
Traceback (most recent call last):

  File "<ipython-input-30-c680011d20e2>", line 1, in <module>
    assert x > 0, "x should be positive"

AssertionError: x maa vaere positiv
```

Testfunksjoner

Anta at vi har laget en funksjon i Python som returnerer en verdi. Hvordan kan vi vite om funksjonen fungerer som den skal?

Enkel strategi: sjekk at funksjonen iallfall gir riktig svar på noen utvalgte eksempler!

Teststrategi

- Anta at vi har laget en funksjon `f(x)` og at vi ønsker å vite at kallet `y = f(x)` gir riktig svar.
- Vi lager en *testfunksjon* i Python som kaller på `f(x)` med noen utvalgte verdier for `x` der vi vet hva output `y` skal være (vi har fasiten).
- Hvis output ikke stemmer med fasiten, skal testfunksjonen gi en feilmelding.

Eksempel

```
# Funksjon som skal finne a[0]+a[k]+a[2k]+...
def finn_sum(a,k):
    res = sum([a[i] for i in range(0,len(a),k)])
    return res

# Vi lager en testfunksjon:
def test_finn_sum():
    """Testfunksjon for finn_sum."""
    a = [0,1,2,3,4,5]        # Eksempel på inputverdi for a
    k = 3                    # Eksempel på inputverdi for k
    expected = 3             # Hva output forventes å bli
    computed = sum3(a,k)     # Hva output faktisk er
    success = (computed == expected) # Fikk vi riktig svar?
    message = 'computed %s, expected %s' % (computed, expected)
    assert success, message

# Vi kaller på testfunksjonen:
test_finn_sum()
```

Utvidelse til flere tester

```
# Funksjon som skal finne a[0]+a[k]+a[2k]+...
def finn_sum(a,k):
    res = sum([a[i] for i in range(0,len(a),3)])
    return res

# Vi lager en testfunksjon:
def test_finn_sum():
    """Testfunksjon for finn_sum."""
    tol = 1E-14
    k = 3
    inputs = [[6], [6,1], [6,1,2], [6,1,2,3]]
    answers = [6, 6, 6, 9]
    for a, expected in zip(inputs, answers):
        computed = finn_sum(a,k)
        message = '%s != %s' % (computed, expected)
        assert abs(expected - computed) < tol, message

# Vi kaller på testfunksjonen:
test_finn_sum()
```

Husk at zip(a, b) lager tupler (a[i],b[i]):

```
>>> zip(inputs, answers)
>>> [([6], 6), ([6, 1], 6), ([6, 1, 2], 6), ([6, 1, 2, 3], 9)]
```

Krav til testfunksjoner

En testfunksjon vil *ikke* gi noe utskrift på skjermen hvis funksjonen som testes består alle testene. Hvis en av testene feiler, vil det komme en feilmelding (`AssertionError`).

Regler for testfunksjoner:

- Hvis navnet på funksjonen som skal testes er `XXX` så skal navnet på testfunksjonen være `test_XXX`.
- En testfunksjon har ingen argumenter
- En testfunksjon må ha en `assert success, message` setning, hvor `success` er `True` hvis testen består og `False` ellers. Siste argument `message` kan droppes.

Hvordan kjøres testfunksjonene?

- **Fra programmet:** Vi kan legge inn et kall på testfunksjonen i samme program som funksjonen er definert.
- **Fra ipython:** Hvis vi kjører Python interaktivt via ipython kan vi kalle på testfunksjonen derfra.
- **Fra kommandolinjen:** Hvis vi står på samme filkatalog (directory) som programfilen, kan vi gi kommandoen `pytest prog.py` for å kjøre alle testfunksjoner i programfilen `prog.py`.
- **Forenklet testing:** Vi kan gi kommandoen `pytest` (uten argumenter) for å kjøre alle testfunksjoner i alle Python-programmer med navn som starter på `test_`.

Enhetstester Testfunksjoner som tester en liten bit av et program (en funksjon) kalles ofte *enhetstester*. For å teste et helt program, må programmet passere alle enhetstestene.

Sluttkommentar om testfunksjoner

- Å bevise at et program oppfører seg korrekt for alle tenkelige input er generelt svært vanskelig.
- Å vise at programmet oppfører seg korrekt for *noen* inputverdier er et skritt på veien og kan ofte være tilstrekkelig.
- Det at en suksessfull test ikke gir noe output kan være irriterende - men bruker du pytest så får du beskjed om hvilke testfunksjoner som er kjørt.

Oppsummering

If-tester:

```
if x < 0:
    value = -1
elif x >= 0 and x <= 1:
    value = x
else:
    value = 1
```

Egendefinerte funksjoner:

```
def quadratic_polynomial(x, a, b, c):
    value = a*x*x + b*x + c
    derivative = 2*a*x + b
    return value, derivative

p, dp = quadratic_polynomial(1, 2, 0.5, 1)
p, dp = quadratic_polynomial(x=x, a=-4, b=0.5, c=0)
```

Argumenter med default-verdi må komme sist:

```
def f(x, A=1, a=1, w=pi):
    return A*exp(-a*x)*sin(w*x)
```