Forelesning IN1900 – 7 September 2023

Ole Christian Lingjærde
Institutt for Informatikk, Universitetet i Oslo

Uke: 4 September – 10 September, 2023

Lister og tupler

Liste:

```
a = [0, 2, 4, 8, 10]
```

Tuppel:

```
a = (0, 2, 4, 8, 10)
```

Hva er forskjellen?

Lister kan modifiseres, tupler kan ikke:

```
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

Konstante og beskyttet mot feilaktige endringer

Raskere beregninger

Brukes mye i Python-programmer

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

Løpe gjennom en liste

Vi repeterer hvordan dette gjøres:

METODE A:

```
a = [1, 5, 7, 8, 2, 4, 8]
for i in range(len(a)):
    print(a[i])
```

METODE B:

```
a = [1, 5, 7, 8, 2, 4, 8]
for e in a:
    print(e)
```

Hvilken metode skal jeg bruke?

METODE A:

```
for i in range(len(a)):
    ... gjør noe med a[i] ...
```

METODE B:

```
for e in a:
    ... gjør noe med e ...
```

Svakheter med metode A?

- litt mer å skrive

Svakheter med metode B?

- ikke egnet hvis vi ønsker å forandre innholdet i listen når vi løper gjennom

Løpe gjennom flere lister samtidig

Vi kan løpe gjennom flere lister samtidig:

METODE A:

```
a = [7, 8, 2, 4, 8]
b = [8, 4, 2, 8, 7]
for i in range(len(a)):
   print(a[i],b[i])
```

METODE B:

```
a = [7, 8, 2, 4, 8]
b = [8, 4, 2, 8, 7]
for e,f in zip(a,b):
  print(e,f)
```

Løpe gjennom flere lister samtidig

Vi kan løpe gjennom flere lister samtidig:

METODE A:

```
a = [7, 8, 2, 4, 8]
b = [8, 4, 2, 8, 7]
for i in range(len(a)):
   print(a[i],b[i])
```

```
7 8
8 4
2 2
4 8
8 7
```

METODE B:

```
a = [7, 8, 2, 4, 8]
b = [8, 4, 2, 8, 7]
for e,f in zip(a,b):
  print(e,f)
```

```
7 8
8 4
2 2
4 8
8 7
```

Hva gjør `zip`?

Anta at vi har disse listene:

```
a = [0,1,2,3]
b = [4,5,6,7]
c = [7,8,9,10]
```

zip(a,b,c) lager tupler av elementene med samme posisjon i de tre listene:

zip(a,b,c) → (0,4,7)
(1,5,8)
(2,6,9)
(3,7,10)

Tuplene blir levert samlet som en "iterator". Den fungerer omtrent som en Pez-dispenser: du kan få ut ett og ett tuppel f.eks. med en for-løkke.

Du kan få tuplene levert som en liste

Funksjonen zip(a,b,c) leverer tuplene i form av en iterator, men du kan få de levert som en liste også:

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

z1 = zip(a,b,c)
print(z1)                    <zip object at 0x7f85518c7d80>

z2 = list(zip(a,b,c))
print(z2)                    [(0,4,7), (1,5,8), (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
```

Nærliggende å tolke som en matrise/tabell:

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

Kort oppsummering av lister

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

If-tester

Vi skal lese et tall fra terminal og skrive ut logaritmen til tallet. Vi lager et program logx.py:

```
import math
print("x = ?")
x = eval(input()) # Leser x fra terminal
logx = math.log(x)
print(f"log({x}) = {logx}")
```

Gir dette programmet alltid fornuftig output?

If-tester (forts.)

Vi prøver programmet med en positiv x:

```
> python logx.py
x = ?
2.7
log(2.7) = 0.9932517730102834
```

Vi prøver igjen, nå med en negativ x:

```
> python logx.py
x = ?
-11
Traceback (most recent call last):
File "<ipython-input-63-b644e331e7a0>", line 4, in <module>
logx = math.log(x)
ValueError: math domain error
```

If-tester (forts.)

Vi utvider programmet med en **if-test** slik at det bare regner ut logaritmen når x > 0 og gir feilmelding ellers:

```
import math
print("x = ?")
x = eval(input())
if x > 0:
    logx = math.log(x)
    print(f"log({x})={logx}")
else:
    print("log(x) er bare definert for x > 0")
```

If-tester (forts.)

Vi prøver programmet med en positiv x:

```
> python logx.py  x = ?
2.7
log(2.7) = 0.9932517730102834
```

Vi prøver igjen, nå med en negativ x:

```
> python logx.py  x = ?
-11
log(x) er bare definert for x > 0
```

→ Fornuftig utskrift uansett fortegn av x.

If-tester (forts.)

Programmet vil fortsatt feile dersom brukeren skriver inn noe annet enn et tall:

```
> python logx.py
x = ?
femti
Traceback (most recent call last):

File "<ipython-input-5-b6b12d30e678>", line 3, in <module>
    x = eval(input())

  File "<string>", line 1, in <module>

NameError: name 'femti' is not defined
```

If-tester (forts.)

Vi utvider programmet slik at det først tester om input er numerisk og deretter om x > 0:

```
import math
print("x = ?")
answer = input()
if answer.isnumeric():
    x = eval(answer)
    if x > 0:
        logx = math.log(x)
        print(f"log({x}) = {logx}")
    else:
        print("x kan ikke være 0")
else:
    print("x må være et positivt tall")
```

If-tester (forts.)

```
import math
print("x = ?")
answer = input()
if answer.isnumeric():
    x = eval(answer)
    if x > 0:
        logx = math.log(x)
        print(f"log({x}) = {logx}")
    else:
        print("x kan ikke være 0")
else:
    print("x må være et positivt tall")
```

Merk: isnumeric returnerer True når alle tegn er sifre, så -2 er ikke numerisk ifølge denne funksjonen.

If-tester (forts.)

```
import math
print("x = ?")
answer = input()
if answer.isnumeric():
    x = eval(answer)
    if x > 0:
        logx = math.log(x)
        print(f"log({x}) = {logx}")
    else:
        print("x kan ikke være 0")
else:
    print("x må være et positivt tall")
```

If-tester (forts.)

Nå fungerer programmet for alle typer input:

```
> python logx.py
x = ?
50
log(50) = 3.912023005428146

> python logx.py
x = ?
0
x kan ikke være 0

> python logx.py
x = ?
-50
x må være et positivt tall

> python logx.py
x = ?
femti
x må være et positivt tall
```

If-tester: de tre variantene

If:

```
if x < 0:
  setninger
```

If-else:

```
if x < 0:
  setninger
else:
  setninger
```

If-elif-else:

```
if x < 0:
  setninger
elif x < 5:
  setninger
else:
  setninger
```

Quiz

Hva skrives ut her?

```
x = 3.14

if x < x**2:
   print("A", end="")

if x < 0:
   print("B", end="")
else:
   x = -x
   print("C", end="")

if x < x**2:
   print("D", end="")
else:
   print("E", end="")
```

Hindrer ny linje etter utskrift

Quiz

Hva skrives ut her?

```
x = 3.14

if x < x**2:
   print("A", end="")

if x < 0:
   print("B", end="")
else:
   x = -x
   print("C", end="")

if x < x**2:
   print("D", end="")
else:
   print("E", end="")
```

Svar: ACD

Quiz

Vi kjører dette programmet:

```
x = 0
if x >= 0:
    x = x + 2
if x-2 <= 0:
     x = x * 2
     if x%2 != 0:
          x = x - 1
     else:
          x = x + 1
else:
    x = x * 6
```

Hvilken verdi har nå x?

Quiz

Vi kjører dette programmet:

```
x = 0
if x >= 0:
    x = x + 2
if x-2 <= 0:
     x = x * 2
     if x%2 != 0:
          x = x - 1
     else:
          x = x + 1
else:
    x = x * 6
```

Hvilken verdi har nå x?

Svar: 5

Quiz

Vi kjører dette programmet:

```
x = 0
for k in range(6):
   if k % 2 == 0:
      x = x + k
   else:
      x = x - k
```

Hvilken verdi har nå x?

Quiz

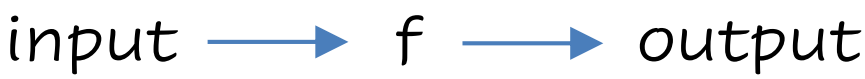
Vi kjører dette programmet:

```
x = 0
for k in range(6):
   if k % 2 == 0:
      x = x + k
   else:
      x = x - k
```

k	k%2==0	x = ...	x
0	True	x=x+0	0
1	False	x=x-1	-1
2	True	x=x+2	1
3	False	x=x-3	-2
4	True	x=x+4	2
5	False	x=x-5	**-3**

Funksjoner i Python

Funksjoner spiller en viktig rolle i Python. De spiller samme rolle som funksjoner i matematikken:



Mens vi i matematikken definerer f med en formel, definerer vi i Python f med programkode.

Eksempel

Matematisk funksjon:

$$f(x) = 2x^2 - 3x$$

Python-funksjon:

```
def f(x):
    return 2*x**2 - 3*x
```

Innebygde funksjoner

Python har en lang rekke innebygde funksjoner:

```
import math
x = math.sin(0.24)      # Sinus til 0.24
y = math.atanh(0.2)     # Invers hyberbolsk tangens til 0.2
z = sorted([6,2,3,7]) # Lager en sortert liste
a = list(range(50))     # Lager liste med verdiene 0,1,...,49
```

En viktig del av dette kurset er å lære å lage slike funksjoner selv.

Eksempel: en formel for sin(x)

- Vi kan regne ut sin(x) med formelen

$$sin(x) = \sum_{k=0}^{\infty} (-1)^k \frac{x^{2k+1}}{(2k+1)!}$$

- Vi får god tilnærming til sin(x) selv med noen få ledd.

Eksempel

Program for å finne sinus til x=0.15 med formelen over:

```
y = x
kfac = 1
sign = 1
for k in range(3, 50, 2):
  kfac = kfac * (k-1) * k
  sign = -sign
  y = y + sign * x**k / kfac
```

Med bruk av innebygd funksjon:

```
import math
y = math.sin(x)
```

Selvdefinerte funksjoner

Generell oppskrift for å lage funksjoner:

```
def fnk(x,y,z):
  programsetninger
```

Her er:

- `fnk`: navnet vi har gitt funksjonen
- `x,y,z`: variablene til funksjonen (null, en eller flere)
- `programsetninger`: det vi ønsker skal utføres

Variablene i en Python-funksjon kalles også argumenter, input-variabler og formelle parametre.

Eksempel 1: Regne ut annengradspolynom

Funksjon som regner ut $x^2 + 3x - 5$ for en gitt x :

```
def f(x):
    value = x**2 + 3*x - 5
    return value
```

Vi har gitt funksjonen navnet `f`, men den kunne også vært kalt `g` eller `funksjon_som_beregner_verdi_av_et_polynom`.

Eksempel på bruk:

```
print(0.5, f(0.5))
```

```
0.5 -3.25
```

Komplett program

```
def f(x):
   value = x**2 + 3*x - 5
   return value

print("x = ?")
x = eval(input())
print(f"Hvis x = {x} så er x^2+3x-5 = {f(x)}")
```

Kjøreeksempel:

```
x = ?
6
Hvis x = 6 så er x^2+3x-5 = 49
```

Eksempel 2: Løse annengradslikning

Løsningene av annengradslikningen $ax^2 + bx + c = 0$:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Python-funksjon som finner løsningene:

```
def solve(a, b, c):
  from math import sqrt
  x1 = (-b - sqrt(b**2 - 4*a*c)) / (2*a)
  x2 = (-b + sqrt(b**2 - 4*a*c)) / (2*a)
  return x1, x2
```

Eksempel: vi løser $3x^2 + 2x - 1 = 0$ med

```
print(solve(3, 2, -1))
```

Komplett program

Komplett program for å løse $ax^2 + bx + c = 0$:

```
def solve(a, b, c):
    from math import sqrt
    x1 = (-b - sqrt(b**2 - 4*a*c)) / (2*a)
    x2 = (-b + sqrt(b**2 - 4*a*c)) / (2*a)
    return x1, x2

print("a, b, c = ?")
a, b, c = eval(input())
x1, x2 = solve(a, b, c)
print(f"Løsningene er x1={x1} og x2={x2}")
```

Komplett program

Komplett program for å løse $ax^2 + bx + c = 0$:

```
def solve(a, b, c):
    from math import sqrt
    x1 = (-b - sqrt(b**2 - 4*a*c)) / (2*a)
    x2 = (-b + sqrt(b**2 - 4*a*c)) / (2*a)
    return x1, x2

print("a, b, c = ?")
a, b, c = eval(input())
x1, x2 = solve(a, b, c)
print(f"Løsningene er x1={x1} og x2={x2}")
```

Kjøreeksempel:

```
a, b, c = ?
4, 2, -2
Løsningene er x1=-1.0 og x2=0.5
```

Eksempel 3: Funksjon som leter i en liste

Kan vi lage en funksjon som finner ut om en liste `a` inneholder en bestemt verdi `x`?

Eksempel 3: Funksjon som leter i en liste

Kan vi lage en funksjon som finner ut om en liste `a` inneholder en bestemt verdi `x`?

Idé: løp gjennom alle elementene i listen med løkke, test om noen av elementene er lik x.

Eksempel 3: Funksjon som leter i en liste

Kan vi lage en funksjon som finner ut om en liste a inneholder en bestemt verdi x?

Idé: løp gjennom alle elementene i listen med løkke, test om noen av elementene er lik x.

```
def findvalue(a, x):
    found = False
    for i in range(len(a)):
        if a[i] == x:
            found = True
    return found
```

Komplett program

```
# Vi definerer letefunksjonen:
def findvalue(a,x):
   found = False
   for i in range(len(a)):
      if a[i] == x:
         found = True
   return found


# Sjekk om listen L=[11, 2, 56, 3] inneholder x=56:

L = [11, 2, 56, 3]
found = findvalue(L,56)
if found:
   print(f"L inneholder verdien {x}")
else:
   print(f"L inneholder ikke verdien {x}")
```

Funksjoner uten argumenter

Vi kan lage funksjoner uten input-variabler, f.eks:

```
def printstars():
    for i in range(50):
        print("*", end="")
```

Funksjoner uten argumenter

Vi kan lage funksjoner uten input-variabler, f.eks:

```
def printstars():
    for i in range(50):
        print("*", end="")
```

Må ha med parentesene når vi bruker funksjonen:

```
printstars()
```

Eksempel: kalender

Funksjon som finner tidspunktet programmet kjøres:

```
def day():
  import time
  day = time.gmtime().tm_yday
  year = time.gmtime().tm_year
  text = f"Det er nå dag nummer {day:g} i år {year:g}"
  return text
```

Eksempel: kalender

Funksjon som finner tidspunktet programmet kjøres:

```
def day():
  import time
  day = time.gmtime().tm_yday
  year = time.gmtime().tm_year
  text = f"Det er nå dag nummer {day:g} i år {year:g}"
  return text
```

Bruk av funksjonen:

```
print(day())
```

```
Det er nå dag nummer 249 i år 2021
```

Returverdier

Funksjoner kan ha null, en eller flere returverdier:

```
def gratulasjon(navn):
    print(f"Gratulerer {navn}, du har vunnet 1 million kroner!")

def kvadrat(x):
    return x**2

def potenser(x):
    return x, x**2, x**3, x**4
```

Quiz 1

Virker dette programmet, og hva skriver det isåfall ut?

```
def skriv_ut:
    print("Hello world")

skriv_ut
```

Det virker ikke:

- mangler parenteser i første linje
- mangler parenteser i siste linje

Quiz 2

Virker dette programmet, og hva skriver det isåfall ut?

```
def skriv_ut():
    print("Hello world")

skriv_ut()
```

Det virker og skriver ut:
Hello world

Quiz 3

Virker dette programmet, og hva skriver det isåfall ut?

```
def skriv_ut(tekst):
    print(tekst)

skriv_ut()
```

Det virker ikke:

- når metoden kalles i siste linje må vi oppgi input-verdi

Quiz 4

Virker dette programmet, og hva skriver det isåfall ut?

```
def skriv_ut(tekst):
    print(tekst)

skriv_ut("Hello world")
```

Det virker og skriver ut:
Hello world

Quiz 5

Virker dette programmet, og hva skriver det isåfall ut?

```
def regn_ut(x):
    y = x**2 + 2*x + 1

print(regn_ut(10))
```

Det virker ikke, fordi metoden ikke returnerer noen verdi.

Quiz 6

Virker dette programmet, og hva skriver det isåfall ut?

```
def regn_ut(x):
    y = x**2 + 2*x + 1
    return y

print(regn_ut(10))
```

Det virker og skriver ut:
121

Quiz 7

Virker dette programmet, og hva skriver det isåfall ut?

```
def f(x):
    y = x**2 + 2*x + 1
    return x,y

x0, y0 = f(10)
print(x0,y0)
```

Det virker og skriver ut:

10 121

Quiz 8

Virker dette programmet, og hva skriver det isåfall ut?

```
def f(x):
    return x**2

def g(x):
    return f(x)**3

print(g(2))
```

Det virker og skriver ut:

64 (g(2) = f(2)**3 = (2**2)**3) = 4**3)

Quiz 9

Virker dette programmet, og hva skriver det isåfall ut?

```
def f(n):
    v = 1
    for i in range(2,n+1):
        v = v*i
    return v

print(f(4))
```

Det virker og skriver ut:

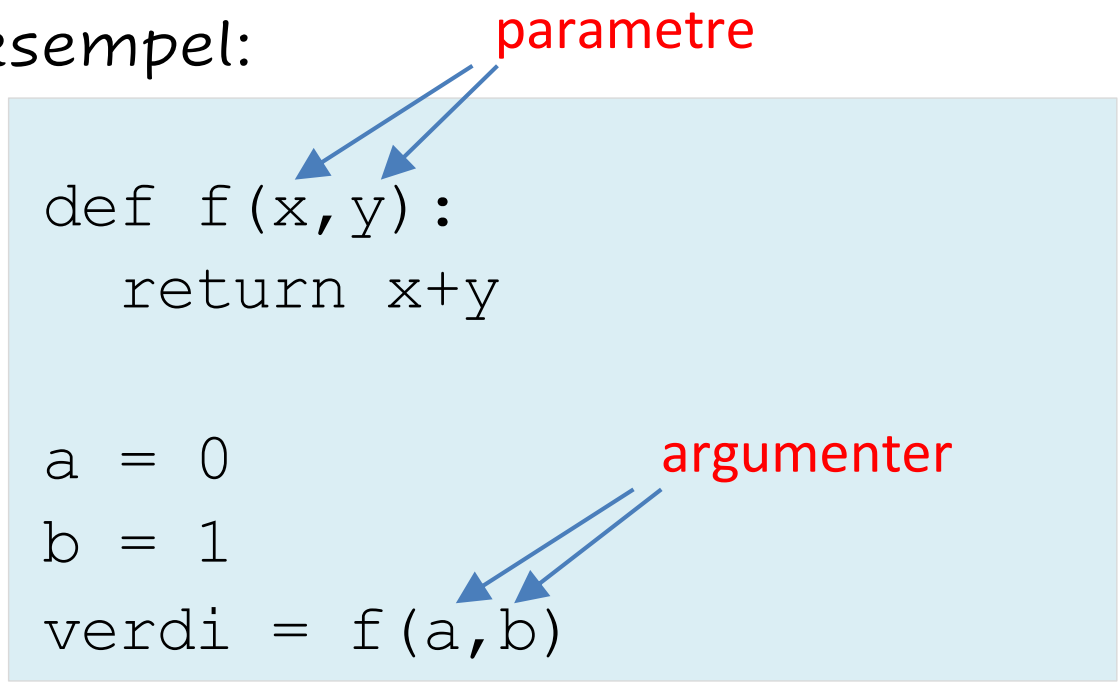
24 (= 2 * 3 * 4)

Overføring av verdier til funksjoner

Regel 1:

- Inputs i en funksjonsdefinisjon kalles parametre
- Inputs i et funksjonskall kalles argumenter

Eksempel:



Overføring av verdier til funksjoner

Regel 2:

> Antall argumenter i et funksjonskall må være det samme som antall parametre i funksjonen.

Eksempel:

```
def f(x,y):
   return x+y

verdi = f(0)        # Ikke lovlig
verdi = f(0,1)      # Lovlig
verdi = f(0,1,2)    # Ikke lovlig
```

Overføring av verdier til funksjoner

Regel 3:

> Vi kan gi default-verdier til parametre når vi definerer en funksjon. Da trenger vi ikke å gi verdier til dem i et kall.

Eksempel:

```
def f(x,y,z=2):
   return x+y+z

verdi = f(0)       # Ikke lovlig
verdi = f(0,1)     # Lovlig
verdi = f(0,1,2)   # Lovlig
```

Overføring av verdier til funksjoner

MERK:

Parametre med default-verdier må stå bakerst i listen av parametre!

EKSEMPEL:

```
def f(x, y=0, z=0):       # Lovlig
   print(x+y+z)

def f(x, y=0, z):         # Ikke lovlig
   print(x+y+z)

def f(x=0, y, z):         # Ikke lovlig
   print(x+y+z)
```

Overføring av verdier til funksjoner

Regel 4:

> Vi kan navngi argumenter når vi kaller på en funksjon. Dette kalles *keyword arguments*. Slike argumenter må komme sist i argumentlisten.

Eksempel:

```
def f(x,y,z):
  return x+y+z

verdi = f(x=0,y=1,z=2)   # Lovlig
verdi = f(0,y=1,z=2)     # Lovlig
verdi = f(0,z=2,y=1)     # Lovlig
verdi = f(z=2,0,1)       # Ikke lovlig
```

Overføring av verdier til funksjoner

Regel 5:

> Vi kan bruke *args hvis vi ønsker å tillate et variabelt antall argumenter i et kall.

Eksempel:

```
def f(*args):
  # Nå er args et tuppel
  for k in args:
    print(k)

f()       # Lovlig
f(0)      # Lovlig
f(0,1)    # Lovlig
```

Variabler og parametre

I matematiske funksjoner skiller vi ofte mellom variabler og parametre/koeffisienter.

Eksempler:

$$f(x) = ax^2 + bx + c$$

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

Når vi programmerer slike funksjoner, må vi også ha med parametrene som input-variabler.

Ofte nyttig å gi dem default-verdier.

Eksempel

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

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

Mulig implementasjon 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)
```

Mange muligheter når vi kaller på funksjonen, f.eks:

```
f(0.5)
f(0.5, A=2)
f(0.5, a=4)
f(0.5, A=3, a=1, omega=4*pi)
```

Quiz 1

Vi har funksjonen

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

Hvilke av disse funksjonskallene er lovlige?

```
r = h(0)                # IKKE lovlig (y mangler)
r = h(0, 1)             # Lovlig
r = h(0, 1, 2)          # Lovlig
r = h(x=0, 1, 2)        # IKKE lovlig (navngitt arg først)
r = h(0, y=1)           # Lovlig
r = h(0, 1, z=3)        # Lovlig
r = h(0, 0, x=0)        # IKKE lovlig (x får verdi 2 ganger)
r = h(z=0, x=1)         # IKKE lovlig (y mangler)
r = h(z=0, x=1, y=2)    # Lovlig
```