Forelesning IN1900 – 12 September 2023

Ole Christian Lingjærde
Institutt for Informatikk, Universitetet i Oslo

Uke: 11 September – 17 September, 2023

Forrige forelesning på en slide

Tupler:

```
(0, 2, 3, 6)
```

Løpe gjennom en liste:

```
for i in range(len(a))
for e in a
```

Løpe gjennom flere lister i parallell:

```
for i in range(len(a))
for e,f in zip(a,b)
```

If-tester:

```
if      if-else      if-elif-else
```

Funksjoner:

```
def f(x, y, z)
def f(x, y, z=3)
f(0, y=3, z=6)
```

Denne ukens agenda

- Litt mer om funksjoner (lambda, testing)
- Input fra bruker
- Lese fra fil
- Skrive til fil
- eval og exec
- Feilhåndtering

Denne ukens agenda

- **Litt mer om funksjoner (lambda, testing)**
- **Input fra bruker**
- **Lese fra fil**
- Skrive til fil
- eval og exec
- Feilhåndtering

I DAG

Denne ukens agenda

- Litt mer om funksjoner (lambda, testing)
- Input fra bruker
- Lese fra fil
- **Skrive til fil**
- **eval og exec**
- **Feilhåndtering**

TORSDAG

Funksjoner med funksjoner som argumenter

Funksjon med tall som argument:

```
def poly(x):
    return (x-1)**2

y = poly(0)  # Vi setter y = 1
```

Funksjon med funksjon som argument:

```
def evaluate(f):
    return f(0)

y = evaluate(poly) # Vi setter y = 1
```

Eksempel

Vi kan estimere den annenderiverte til en funksjon f i et gitt punkt x med denne formelen:

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

hvor $h > 0$ er et lite tall (f.eks. h=0.000001).

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

Lambda-funksjoner

Definere funksjon med def:

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

Definere funksjon med lambda:

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

Docstrings

En standardisert måte å kommentere funksjoner på (og senere: klasser) er å bruke en docstring:

```
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).
    """
    a = (y1 - y0)/(x1 - x0)
    b = y0 - a*x0
    return a, b
```

Docstrings

Vi kan bruke help(...) for å få se docstring'en:

```
> help(line)

Help on function line in module __main__:

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).
```

Quiz

Hva gjør denne funksjonen?

```
def findmax(f):
    x = [i/100 for i in range(101)]
    y = [f(e) for e in x]
    return max(y)
```

Quiz

Hva gjør denne funksjonen?

```
def findmax(f):
    x = [i/100 for i in range(101)]
    y = [f(e) for e in x]
    return max(y)
```

SVAR:

Estimerer maksimumsverdien til funksjonen f på intervallet [0,1] ved å se på f(0.00), f(0.01), f(0.02), ..., f(1.00)

Quiz

Hva skrives ut her?

```
def findmax(f):
    x = [i/100 for i in range(101)]
    y = [f(e) for e in x]
    return max(y)

y = findmax(lambda x: (x-0.5)**2)
print(y)
```

Quiz

Hva skrives ut her?

```
def findmax(f):
    x = [i/100 for i in range(101)]
    y = [f(e) for e in x]
    return max(y)

y = findmax(lambda x: (x-0.5)**2)
print(y)
```

SVAR:

0.25 (= 0.5**2)

Å sjekke at et program fungerer

Hvordan vite at et program fungerer?

- Må strengt tatt vise at alle input gir korrekt output (vanskelig!)
- Alternativ: vise at iallfall noen input gir korrekt output (lett!)

Å sjekke at et program fungerer

Hvordan vite at et program fungerer?

- Må strengt tatt vise at alle input gir korrekt output (vanskelig!)
- Alternativ: vise at iallfall noen input gir korrekt output (lett!)

Vanlig praksis:

- Teste én funksjon av gangen
- Sjekke at svaret blir riktig for noen testeksempler
- Enhetstesting

Eksempel A: finne maksverdi

Filnavn: maksimum.py

```
def maxval(x):
    m = x[0]
    for e in x:
        if e > m:
            m = e
    return m

### Testfunksjon:
def test_maxval():
    input = [6,2,8,3,16,-1]
    expected = 16
    computed = maxval(input)
    result = (expected == computed)
    assert result
```

Filnavn: maksimum.py

```
def maxval(x):
    m = x[0]
    for e in x:
        if e > m:
            m = e
    return m
```

Funksjonen vi ønsker å teste

```
### Testfunksjon:

def test_maxval():
    input = [6,2,8,3,16,-1]
    expected = 16
    computed = maxval(input)
    result = (expected == computed)
    assert result
```

Filnavn: maksimum.py

```
def maxval(x):
    m = x[0]
    for e in x:
        if e > m:
            m = e
    return m

### Testfunksjon:
def test_maxval():
    input = [6,2,8,3,16,-1]
    expected = 16
    computed = maxval(input)
    result = (expected == computed)
    assert result
```

Navnet på testfunksjonen

Filnavn: maksimum.py

```
def maxval(x):
    m = x[0]
    for e in x:
        if e > m:
            m = e
    return m

### Testfunksjon:

def test_maxval():
    input = [6,2,8,3,16,-1]
    expected = 16
    computed = maxval(input)
    result = (expected == computed)
    assert result
```

Test-input og forventet output

Filnavn: maksimum.py

```
def maxval(x):
    m = x[0]
    for e in x:
        if e > m:
            m = e
    return m

### Testfunksjon:
def test_maxval():
    input = [6,2,8,3,16,-1]
    expected = 16
    computed = maxval(input)
    result = (expected == computed)
    assert result
```

Beregnet output

Filnavn: maksimum.py

```
def maxval(x):
    m = x[0]
    for e in x:
        if e > m:
            m = e
    return m

### Testfunksjon:
def test_maxval():
    input = [6,2,8,3,16,-1]
    expected = 16
    computed = maxval(input)
    result = (expected == computed)
    assert result
```

Selve testen

Filnavn: maksimum.py

```
def maxval(x):
    m = x[0]
    for e in x:
        if e > m:
            m = e
    return m

### Testfunksjon:
def test_maxval():
    input = [6,2,8,3,16,-1]
    expected = 16
    computed = maxval(input)
    result = (expected == computed)
    assert result
```

Gir feilmelding hvis result == F

Vi kjører testen:

```
(base) ole@oles-mbp Code % pytest maksimum.py

========== test session starts ==========
platform darwin -- Python 3.9.12, pytest-7.1.1, pluggy-1.0.0
rootdir: /Users/ole/Documents/IN1900/H23/Uke4/Code
plugins: anyio-3.5.0
collected 1 item


maksimum.py .                                                [100%]
========== 1 passed in 0.01s ==========
```

→ Testen er bestått

Eksempel B: summere annenhver verdi

Filnavn: addisjon.py

```
### Funksjon som skal finne x[0]+x[2]+x[4]+...
def sum2(x):
    s = 0
    for i in range(0, len(x), 2):
        s = s + x[i]
    return s

### Testfunksjon:
def test_sum2():
    input = [1, 3, 7, 9, 4]
    expected = 12     # 1+7+4
    computed = sum2(input)
    result = (expected == computed)
    assert result
```

Vi kjører testen:

```
(base) ole@oles-mbp Code % pytest addisjon.py

========== test session starts ==========
platform darwin -- Python 3.9.12, pytest-7.1.1, pluggy-1.0.0
rootdir: /Users/ole/Documents/IN1900/H23/Uke4/Code
plugins: anyio-3.5.0
collected 1 item


addisjon.py .                                                [100%]
========== 1 passed in 0.01s ==========
```

→ Testen er bestått

Eksempel C: funksjon som ikke virker

Filnavn: addisjon2.py

```
### Funksjon som skal finne x[0]+x[2]+x[4]+...
def sum2(x):
    s = 0
    for i in range(2, len(x), 2):
        s = s + x[i]
    return s

### Testfunksjon:
def test_sum2():
    input = [1, 3, 7, 9, 4]
    expected = 12     # 1+7+4
    computed = sum2(input)     # 7+4
    result = (expected == computed)
    assert result
```

```
(base) ole@oles-mbp Code % pytest addisjon2.py
========================= test session starts ==========================
platform darwin -- Python 3.9.12, pytest-7.1.1, pluggy-1.0.0
rootdir: /Users/ole/Documents/IN1900/H23/Uke4/Code
plugins: anyio-3.5.0
collected 1 item

addisjon2.py F                                                   [100%]
============================== FAILURES ===============================
_____________________ test_sum2 ______________________________

    def test_sum2():
        input = [1, 3, 7, 9, 4]
        expected = 12    # 1+7+4
        computed = sum2(input)
        result = (expected == computed)
>       assert result
E       assert False

addisjon2.py:16: AssertionError
======================== short test summary info ========================
FAILED addisjon2.py::test_sum2 - assert False
========================== 1 failed in 0.06s ===========================
```

```
(base) ole@oles-mbp Code % pytest addisjon2.py
========================= test session starts ==========================
platform darwin -- Python 3.9.12, pytest-7.1.1, pluggy-1.0.0
rootdir: /Users/ole/Documents/IN1900/H23/Uke4/Code
plugins: anyio-3.5.0
collected 1 item

addisjon2.py F                                                    [100%]
============================= FAILURES =================================
_____________________ test_sum2 ____________________________

    def test_sum2():
        input = [1, 3, 7, 9, 4]
        expected = 12    # 1+7+4
        computed = sum2(input)
        result = (expected == computed)
>       assert result
E       assert False

addisjon2.py:16: AssertionError
======================= short test summary info ========================
FAILED addisjon2.py::test_sum2 - assert False
========================= 1 failed in 0.06s ============================
```

Noen forbedringer

Vi kan forbedre testingen:

- Flere testeksempler
  - Øker sjansen for å oppdage feil
- Tillate små avvik (f.eks. < 1e-6) mellom expected og computed ved å bruke kriteriet result = abs(expected-computed) < 1e-6
  - Unngår at testen feiler pga avrundingsfeil
- Gi fornuftig feilmelding ved å bruke varianten assert result, 'Feilmelding'
  - Litt lettere å se hva som gikk galt

Eksempel C: beregne derivert

Filnavn: deriv.py

```
### Funksjon som skal beregne f'(1)

def diff1(f):
    h = 1e-6
    deriv = (f(1+h)-f(1))/h
    return deriv

### Testfunksjon:
def test_diff1():
    from math import sin, cos
    input = [lambda x: x**2, lambda x: sin(x)]
    expected = [2, cos(1)]
    for inp,exp in zip(input,expected):
        comp = diff1(inp)
        result = abs(exp-comp)<1e-6
        assert result, f'Got {comp}, expected {exp}'
```

Filnavn: deriv.py

```
### Funksjon som skal beregne f'(1)

def diff1(f):
    h = 1e-6
    deriv = (f(1+h)-f(1))/h
    return deriv

### Testfunksjon:
def test_diff1():
    from math import sin, cos
    input = [lambda x: x**2, lambda x: sin(x)]
    expected = [2, cos(1)]
    for inp,exp in zip(input,expected):
        comp = diff1(inp)
        result = abs(exp-comp)<1e-6
        assert result, f'Got {comp}, expected {exp}'
```

Funksjon som beregner den deriverte av f(x) i punktet x=1

Filnavn: deriv.py

```
### Funksjon som skal beregne f'(1)

def diff1(f):
    h = 1e-6
    deriv = (f(1+h)-f(1))/h
    return deriv

### Testfunksjon:
def test_diff1():
    from math import sin, cos
    input = [lambda x: x**2, lambda x: sin(x)]
    expected = [2, cos(1)]
    for inp,exp in zip(input,expected):
        comp = diff1(inp)
        result = abs(exp-comp)<1e-6
        assert result, f'Got {comp}, expected {exp}'
```

To test-eksempler

Filnavn: deriv.py

```
### Funksjon som skal beregne f'(1)

def diff1(f):
    h = 1e-6
    deriv = (f(1+h)-f(1))/h
    return deriv

### Testfunksjon:
def test_diff1():
    from math import sin, cos
    input = [lambda x: x**2, lambda x: sin(x)]
    expected = [2, cos(1)]
    for inp,exp in zip(input,expected):
        comp = diff1(inp)
        result = abs(exp-comp)<1e-6
        assert result, f'Got {comp}, expected {exp}'
```

Løp gjennom eksemplene med zip

Filnavn: deriv.py

```
### Funksjon som skal beregne f'(1)

def diff1(f):
    h = 1e-6
    deriv = (f(1+h)-f(1))/h
    return deriv

### Testfunksjon:
def test_diff1():
    from math import sin, cos
    input = [lambda x: x**2, lambda x: sin(x)]
    expected = [2, cos(1)]
    for inp,exp in zip(input,expected):
        comp = diff1(inp)
        result = abs(exp-comp)<1e-6
        assert result, f'Got {comp}, expected {exp}'
```

Tillat små avvik mellom exp og comp

Filnavn: deriv.py

```
### Funksjon som skal beregne f'(1)

def diff1(f):
    h = 1e-6
    deriv = (f(1+h)-f(1))/h
    return deriv

### Testfunksjon:
def test_diff1():
    from math import sin, cos
    input = [lambda x: x**2, lambda x: sin(x)]
    expected = [2, cos(1)]
    for inp,exp in zip(input,expected):
        comp = diff1(inp)
        result = abs(exp-comp)<1e-6
        assert result, f'Got {comp}, expected {exp}'
```

Fornuftig feilmelding

Quiz

Hva er feilen(e) her?

```
def mean(a):
  m = 0
  for e in a:
     m += e
  return m/len(a)

def test_mean(a):
  input = [[1,2], [3,5], [0,6]]
  expec = [1.5, 4, 3.5]
  for inp,exp in zip(input,expec):
    result = (mean(inp)-exp)<1e-6
    assist result, 'Feil!'
```

Quiz

Hva er feilen(e) her?

```
def mean(a):
    m = 0
    for e in a:
        m += e
    return m/len(a)

def test_mean(a):
    input = [[1,2], [3,5], [0,6]]
    expec = [1.5, 4, 3.5]
    for inp,exp in zip(input,expec):
        result = (mean(inp)-exp)<1e-6
        assist result, 'Feil!'
```

- Testfunksjoner har ikke parametre
- (0+6)/2 = 3.5
- Må ha med abs(...)
- assert, ikke assist

Input fra bruker

Hvordan får programmer tak i data?
Alternativer:

- Legge data direkte inn i programmet
  - Hardkodet input
- Få data når programmet startes
  - Kommandolinje input
- Få data når programmet kjører
  - Lese fra terminal
  - Lese fra fil
  - Lese fra database, internett, sensorer, ...

Input fra bruker

Hvordan får programmer tak i data?
Alternativer:

- Legge data direkte inn i programmet
  - Hardkodet input
- Få data når programmet startes
  - Kommandolinje input
- Få data når programmet kjører
  - Lese fra terminal
  - Lese fra fil
  - Lese fra database, internett, sensorer, ...

Hardkodet input

Eksempel:

```
# Inputverdier hardkodet som programsetninger:
a = 0.43
b = 6.233
x = 5

def p(x):
  res = a + b*x
  return res

print(f'p({x}) = {p(x)}')
```

Hardkodet input

Eksempel:

```
# Inputverdier hardkodet som programsetninger:
a = 0.43
b = 6.233
x = 5

def p(x):
  res = a + b*x
  return res

print(f'p({x}) = {p(x)}')
```

```
Terminal> python mittprogram.py
p(5.0) = 31.595
```

Hardkodet input

Eksempel:

```
# Inputverdier hardkodet som programsetninger:
a = 0.43
b = 6.233
x = 5

def p(x):
  res = a + b*x
  return res

print(f'p({x}) = {p(x)}')
```

Største ulempe med denne teknikken er mangel på fleksibilitet.
Hvis vi ønsker å endre input, må vi endre på selve programmet.

Kommandolinje input

Eksempel:

```
# Inputverdier leses fra terminal ved programstart:
import sys
a = float(sys.argv[1])
b = float(sys.argv[2])
x = float(sys.argv[3])

def p(x):
  res = a + b*x + c*x**2
  return res

print(f'p({x}) = {p(x)}')
```

Kommandolinje input

Eksempel:

```
# Inputverdier leses fra terminal ved programstart:
import sys
a = float(sys.argv[1])
b = float(sys.argv[2])
x = float(sys.argv[3])

def p(x):
  res = a + b*x
  return res

print(f'p({x}) = {p(x)}')
```

```
Terminal> python mittprogram.py 0.43 6.233 5
p(5.0) = 31.595
```

Mer om kommandolinje input

- Alle kommandolinje argumenter er tilgjengelige i programmet via listen sys.argv
- Alle er i utgangspunktet tekststrenger, og trenger du andre datatyper må du typekonvertere
- Flere ord atskilt av blanke kan gjøres til ett argument ved å omslutte ordet med doble anførselstegn
- Eksempler:

```
Terminal> python mittprogram.py a b c
--> sys.argv: ['mittprogram.py', 'a', 'b', 'c']

Terminal> python mittprogram.py "Per Olsen" 43
--> sys.argv: ['mittprogram.py', 'Per Olsen', '43']
```

Lese fra terminal

Eksempel:

```
# Inputverdier lest fra bruker under programkjøring:
a = float(input('a = ? '))
b = float(input('b = ? '))
x = float(input('x = ? '))

def p(x):
  res = a + b*x
  return res

print(f'p({x}) = {p(x)}')
```

Lese fra terminal

Eksempel:

```
# Inputverdier lest fra bruker under programkjøring:
a = float(input('a = ? '))
b = float(input('b = ? '))
x = float(input('x = ? '))

def p(x):
  res = a + b*x
  return res

print(f'p({x}) = {p(x)}')
```

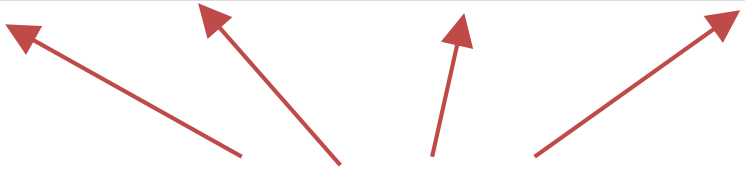
```
Terminal> python mittprogram.py
a = ? 0.43
b = ? 6.233
x = ? 5
p(5.0) = 31.595
```

Lese tekstfil

Navn	Postnummer	Telefon	Epost
Ole Olsen	0316	62336316	ole.olsen@gmail.com
Anne Arnesen	0778	92630023	anne33@live.no
Per Arne Jensen	4513	67239235	peraj@uio.no

Overskrift

Linjer med samme format



Kolonner separert av blanke tegn

Må løse to oppgaver:

- Få tak i hvert enkelt element i filen
- Legge elementene inn i en datastruktur

Vi leser tekstfiler slik:

- Vi åpner filen for lesing
- Vi leser filen linje for linje*
- Vi lukker filen

* Det finnes andre måter å lese på, men disse er mindre viktige i IN1900

Vi leser tekstfiler slik:

- Vi åpner filen for lesing
- Vi leser filen linje for linje*
- Vi lukker filen

* Det finnes andre måter å lese på, men disse er mindre viktige i IN1900

```
infile = open('filename.txt', 'r')   # Åpne filen

overskrift = infile.readline()       # Les overskrift

for line in infile:                  # Les resten en
   ...gjør noe med line...           # linje av gangen

infile.close()                       # Lukk filen
```

Splitte i enkeltord

For å ekstrahere hvert enkelt ord i en linje kan vi bruke split():

```
infile = open('filename.txt', 'r')

overskrift = infile.readline()

for line in infile:
    words = line.split() # Splitt linjen i enkeltord
    # Ordene er nå i words[0], words[1], words[2], ...

infile.close()
```

Eksempel

Vi ønsker å lese denne filen og beregne hvor mye nedbør det kom totalt i Bergen basert på disse målingene:

Nedbor.txt:

```
Rainfall in mm in Bergen:
Jan  254
Feb  216
Mar  205
Apr  141
Mai  118
Jun  129
Jul  153
Aug  196
Sep  249
Okt  267
Nov  252
Des  281
```

Eksempel

```
infile = open('Nedbor.txt', 'r')

infile.readline()

rain = []
for line in infile:
    words = line.split() # Splitt linjen i enkeltord
    rain.append(float(words[1]))

total = sum(rain)
print(f'Total nedbørsmengde: {total}')

infile.close()
```