Forelesning IN1900 – 14 September 2023

Ole Christian Lingjærde
Institutt for Informatikk, Universitetet i Oslo

Uke: 11 September – 17 September, 2023

Dagens agenda

- Mer om lesing fra fil
- Funksjonene eval og exec
- Skriving til fil
- Unntakshåndtering
- Moduler

Quiz 1

Hva skrives ut i hvert tilfelle?

```
x = [i-1 for i in range(10)]
print(x[1])

x = [i+1 for i in range(-2,10)]
print(x[1])

x = [0, 23, 63]; y = x[-2]
print(y)

x = [0, 23, 63] + [0, 33]
print(x[3])

x = range(2, 201, 2)
print(f'{x[-2]}')

s = '3.0423'
print(s[2:2])
```

Quiz 1

Hva skrives ut i hvert tilfelle?

```
x = [i-1 for i in range(10)]
print(x[1])                                  # 0

x = [i+1 for i in range(-2,10)]
print(x[1])

x = [0, 23, 63]; y = x[-2]
print(y)

x = [0, 23, 63] + [0, 33]
print(x[3])

x = range(2, 201, 2)
print(f'{x[-2]}')

s = '3.0423'
print(s[2:2])
```

Quiz 1

Hva skrives ut i hvert tilfelle?

```
x = [i-1 for i in range(10)]
print(x[1])                                  # 0

x = [i+1 for i in range(-2,10)]
print(x[1])                                  # 0

x = [0, 23, 63]; y = x[-2]
print(y)

x = [0, 23, 63] + [0, 33]
print(x[3])

x = range(2, 201, 2)
print(f'{x[-2]}')

s = '3.0423'
print(s[2:2])
```

Quiz 1

Hva skrives ut i hvert tilfelle?

```
x = [i-1 for i in range(10)]
print(x[1])                                  # 0

x = [i+1 for i in range(-2,10)]
print(x[1])                                  # 0

x = [0, 23, 63]; y = x[-2]
print(y)                                     # 23

x = [0, 23, 63] + [0, 33]
print(x[3])

x = range(2, 201, 2)
print(f'{x[-2]}')

s = '3.0423'
print(s[2:2])
```

Quiz 1

Hva skrives ut i hvert tilfelle?

```
x = [i-1 for i in range(10)]
print(x[1])                                  # 0

x = [i+1 for i in range(-2,10)]
print(x[1])                                  # 0

x = [0, 23, 63]; y = x[-2]
print(y)                                     # 23

x = [0, 23, 63] + [0, 33]
print(x[3])                                  # 0

x = range(2, 201, 2)
print(f'{x[-2]}')

s = '3.0423'
print(s[2:2])
```

Quiz 1

Hva skrives ut i hvert tilfelle?

```
x = [i-1 for i in range(10)]
print(x[1])                                  # 0

x = [i+1 for i in range(-2,10)]
print(x[1])                                  # 0

x = [0, 23, 63]; y = x[-2]
print(y)                                     # 23

x = [0, 23, 63] + [0, 33]
print(x[3])                                  # 0

x = range(2, 201, 2)
print(f'{x[-2]}')                            # 198

s = '3.0423'
print(s[2:2])
```

Quiz 1

Hva skrives ut i hvert tilfelle?

```
x = [i-1 for i in range(10)]
print(x[1])                                  # 0

x = [i+1 for i in range(-2,10)]
print(x[1])                                  # 0

x = [0, 23, 63]; y = x[-2]
print(y)                                     # 23

x = [0, 23, 63] + [0, 33]
print(x[3])                                  # 0

x = range(2, 201, 2)
print(f'{x[-2]}')                            # 198

s = '3.0423'
print(s[2:2])                                # Ingenting
```

Kort oppsummering fra sist om testing

mittprogram.py:

```
def f(x):
  y = 2*x**2 + 3*x + 1
  return y
```

LEGG TIL TESTFUNKSJONER

mittprogram.py:

```
def f(x):
  y = 2*x**2 + 3*x + 1
  return y

def test_f():
  input = [0, 1, 2]
  expec = [1, 6, 15]
  for inp,exp in zip(input,expec):
     assert f(inp) == exp, "Feil!"
```

KJØR TESTING

Terminalvindu:

```
Terminal> pytest mittprogram.py
Terminal>
```

Mer om lesing fra fil

Tre "standard-måter" å lese tekstfiler på:

Lese en linje av gangen:	`infile.readline()`
Lese alt inn i en liste:	`infile.readlines()`
Lese alt inn i en string:	`infile.read()`

Metode A: lese en linje av gangen

Vi kan bruke `for line in infile` for å lese en linje av gangen. Alternativt brukes `infile.readline()`.

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

# Metode A.1
for line in infile:
    ... gjør noe med line ...
infile.close()

# Metode A.2
line = infile.readline()
while line:
    ... gjør noe med line ...
    line = infile.readline()
infile.close()
```

Eksempel

Filen AI.txt:

Artificial intelligence (AI) is the intelligence of machines or software, as opposed to the intelligence of humans or animals. It is also the field of study in computer science that develops and studies intelligent machines. "AI" may also refer to the machines themselves.

AI technology is widely used throughout industry, government and science. Some high-profile applications are: advanced web search engines (e.g., Google Search), recommendation systems (used by YouTube, Amazon, and Netflix), understanding human speech (such as Siri and Alexa), self-driving cars (e.g., Waymo), generative or creative tools (ChatGPT and AI art), and competing at the highest level in strategic games (such as chess and Go).

Kode for å lese:

```
infile = open('AI.txt', 'r')
for line in infile:
    print(line)
infile.close()
```

Eksempel

Utskriften:

Artificial intelligence (AI) is the intelligence of machines or software, as opposed to the intelligence of humans or animals. It is also the field of study in computer science that develops and studies intelligent machines. "AI" may also refer to the machines themselves.

AI technology is widely used throughout industry, government and science. Some high-profile applications are: advanced web search engines (e.g., Google Search), recommendation systems (used by YouTube, Amazon, and Netflix), understanding human speech (such as Siri and Alexa), self-driving cars (e.g., Waymo), generative .....

Eksempel

Utskriften:

Artificial intelligence (AI) is the intelligence of machines or software, as opposed to the intelligence of humans or animals. It is also the field of study in computer science that develops and studies intelligent machines. "AI" may also refer to the machines themselves.

AI technology is widely used throughout industry, government and science. Some high-profile applications are: advanced web search engines (e.g., Google Search), recommendation systems (used by YouTube, Amazon, and Netflix), understanding human speech (such as Siri and Alexa), self-driving cars (e.g., Waymo), generative .....

Hva gikk feil?

Eksempel

Når vi leser filen, leser vi også alle linjeskiftene. Disse blir liggende som kontrolltegn på slutten av hver linje og må eventuelt fjernes hvis vi ikke ønsker dem.

```
infile = open('AI.txt', 'r')
for line in infile:
   print(line, end="")
infile.close()
```

Ikke fjern linjeskift, men endre utskrift

```
infile = open('AI.txt', 'r')
for line in infile:
   line = line.replace('\n', '')
   print(line)
infile.close()
```

Fjern linjeskift, behold utskrift

Metode B: lese alt inn i en liste

Vi kan bruke `readlines()` til å lese filen inn i en liste. Da blir hver linje et eget element i listen.

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

lines = infile.readlines()
for line in lines:
   print(line)
infile.close()
```

Eksempel

Filen AI.txt:

Artificial intelligence (AI) is the intelligence of machines or software, as opposed to the intelligence of humans or animals. It is also the field of study in computer science that develops and studies intelligent machines. "AI" may also refer to the machines themselves.

AI technology is widely used throughout industry, government and science. Some high-profile applications are: advanced web search engines (e.g., Google Search), recommendation systems (used by YouTube, Amazon, and Netflix), understanding human speech (such as Siri and Alexa), self-driving cars (e.g., Waymo), generative or creative tools (ChatGPT and AI art), and competing at the highest level in strategic games (such as chess and Go).

Kode for å lese:

```
infile = open('AI.txt', 'r')
lines = infile.readlines()
infile.close()
```

Eksempel

Innholdet i listen `lines`:

```
['Artificial intelligence (AI) is the intelligence of \n',
 'machines or software, as opposed to the intelligence \n',
 'of humans or animals. It is also the field of study \n',
  .... osv ...
 ]
```

Metode C: lese alt inn i en string

Vi kan bruke `read()` for å lese hele filen inn i en string. For å splitte stringen inn i enkeltord kan vi etterpå bruke `split()`.

```
infile = open('data.txt', 'r')
text = infile.read()
words = text.split()
infile.close()
```

Funksjonen `eval`

Når et Python-program kjører, blir uttrykkene i programmet *evaluert* for å finne verdien. Eksempel:

```
x = 3 * math.sin(0.5)
a = [3,4,6] + [2,3]
s = 'Result: 5.0'
```

er ekvivalent med:

```
x = eval("3 * math.sin(0.5)")
a = eval("[3,4,6] + [2,3]")
s = eval("'Result: 5.0'")
```

Resultatet av `eval(expr)` er kontekstavhengig

Anta at exprer et uttrykk som inneholder variabler, f.eks. `x**2+5`. Da vil verdien til `eval(expr)` avhenge av verdien til disse variablene.

```
expr = 'f(x-1) + f(x)'

f = lambda x: x**2
x = 10
res = eval(expr)
print(f'Result = {res}')

f = lambda x: x
x = 5
res = eval(expr)
print(f'Result = {res}')
```

Resultatet av `eval(expr)` er kontekstavhengig

Anta at exprer et uttrykk som inneholder variabler, f.eks. `x**2+5`. Da vil verdien til `eval(expr)` avhenge av verdien til disse variablene.

```
expr = 'f(x-1) + f(x)'

f = lambda x: x**2
x = 10
res = eval(expr)
print(f'Result = {res}')  # UTSKRIFT: Result = 181

f = lambda x: x
x = 5
res = eval(expr)
print(f'Result = {res}')
```

Resultatet av `eval(expr)` er kontekstavhengig

Anta at exprer et uttrykk som inneholder variabler, f.eks. `x**2+5`. Da vil verdien til `eval(expr)` avhenge av verdien til disse variablene.

```
expr = 'f(x-1) + f(x)'

f = lambda x: x**2
x = 10
res = eval(expr)
print(f'Result = {res}')  # UTSKRIFT: Result: 181

f = lambda x: x
x = 5
res = eval(expr)
print(f'Result = {res}')   # UTSKRIFT: Result = 9
```

Å konvertere et uttrykk til en funksjon

Vi kan konvertere et uttrykk til en funksjon. Eksempel:

```
# Definer et uttrykk:
expr = 'x + 3*x - 5*x**2'   # Variabelnavnet er x

# Konverter uttrykket til en funksjon:
def f(x):
    res = eval(expr)
    return res

# Bruk funksjonen:
print(f'f(0.5) = {f(0.5)}')   # f(0.5) = 0.75
```

Spørsmål:

Hva skjer ovenfor hvis variabelen i `expr` heter `y` i stedet for `x`?

Hvis vi bruker feil variabelnavn

```
# Vi definerer et uttrykk
expr = 'y + 3*y - 5*y**2'    # Variabelnavn: y

# Vi prøver å konvertere det til en funksjon av x
def f(x):
    res = eval(expr) return res

# Vi prøver å bruke funksjonen
print(f'f(0.5) = {f(0.5)}')
```

```
Terminal> python myprog.py
NameError                                 Traceback (most recent call
<ipython-input-9-303f7906c038> in <module>() 8
      9 # Use the function
---> 10 print('f(0.5) = {f(0.5)}')
<ipython-input-9-303f7906c038> in f(x)
      4 # We try to convert the expression into a function of x
      5 def f(x):


----> 6      res = eval(expr)
      7      return res
      8
<string> in <module>()
NameError: name 'y' is not defined
```

Hva kan `eval` brukes til?

- Vi kan konstruere nye uttrykk under programkjøring, og få dem evaluert.
- Vi kan lese uttrykk fra fil, eller fra bruker, og evaluere dem.

Eksempel A: finne maksimumsverdien til et uttrykk

Oppgave: skriv et program `maxval.py` som leser inn et uttrykk som et kommandolinje-argument og som finner maksimalverdien til uttrykket for x i intervallet [0,1].

maxval.py

```
import sys
from math import *
expr = sys.argv[1]
xval = [float(i)/1000 for i in range(0,1001)]
yval = [eval(expr) for x in xval]
print(f'Maximal value on [0,1]: {max(yval):5.2f}')
```

Kjøring av programmet

```
Terminal> python maxval.py "log(x+1)*cos(x)"
Maximum value on [0,1]:  0.41
```

Eksempel B: skriv ut verdien til uttrykket for ulike x

Oppgave: skriv et program `ftable.py` som leser inn et uttrykk og et positivt heltall fra kommandolinjen og skriver ut verdien til uttrykket for x = 1/n, 2/n, ..., n/n.

ftable.py

```
import sys
from math import *
expr = sys.argv[1]
n = int(sys.argv[2])
xval = [float(i)/n for i in range(1,n+1)]
yval = [eval(expr) for x in xval]
for x,y in zip(xval,yval):
    print(f'{x:5.2f} {y:5.2f}')
```

Kjøring av programming

```
Terminal> python ftable.py "log(x)*cos(x)" 5
 0.20 -1.58
 0.40 -0.84
 0.60 -0.42
 0.80 -0.16
 1.00  0.00
```

Eksempel C: utfør numerisk integrasjon

Oppgave: skriv et program `diff.py` som spør brukeren om et uttrykk f(x) og en verdi x0, leser inn disse, og deretter beregner verdien f'(x0) numerisk fra følgende formel:

$$f'(x_0) \approx \frac{f(x_0 + h) - f(x_0 - h)}{2h} \qquad (h \text{ small})$$

diff.py

```
# from math import *
expr = input("Expression = ? ")
x0 = float(input("x0 = ? "))

def f(x):
    return eval(expr)

def der(f, x, h=1E-5):
    return (f(x+h) - f(x-h))/(2*h)

print(f'f(x)={expr} ==> f\'({x0})={der(f,x0):5.2f}')
```

Vi kjører programmet

```
Terminal> python diff.py
Expression = ? x**2 + 3*x
x0 = ? 2
f(x)=x**2 + 3*x ==> f'(2)= 7.00
```

Funksjonen `exec`

Funksjonen `exec(s)` er analog til `eval(s)`, bortsett fra at den tolker `s` som en programsetning (eller flere) og utfører denne. Eksempel: anta at vi har

```
x = 1.5 * math.sin(0.5)
x = x**2 - 3
a = [3,4,6] + [2,3]
s = 'Result: 5.0' % 5.0
```

Dette er ekvivalent med:

```
code = """
x = 1.5 * math.sin(0.5)
x = x**2 - 3
a = [3,4,6] + [2,3]
s = 'Result: 5.0'
"""
exec(code)
```

Legg merke til bruken av triple anførselstegn (""") som brukes for å definere en string som strekker seg over flere linjer.

Å skrive data til fil

Grunnleggende idé:

```
outfile = open(filename, 'w')

for data in somelist:
    outfile.write(sometext + '\n')

outfile.close()
```

Vi kan legge tekst til en eksisterende fil med `open(filename, 'a')`.

Eksempel: skrive en tabell til fil

Problem:

Vi har en nestet liste (rader og kolonner):

```
data = \
[[ 0.75,        0.29619813, -0.29619813, -0.75       ],
 [ 0.29619813,  0.11697778, -0.11697778, -0.29619813],
 [-0.29619813, -0.11697778,  0.11697778,  0.29619813],
 [-0.75,       -0.29619813,  0.29619813,  0.75       ]]
```

Skriv disse dataene til fil som en tabell

Løsning:

```
outfile = open('tmp_table.dat', 'w')
for row in data:
for column in row:
    outfile.write(f'{column:14.8f}')
    outfile.write('\n')
outfile.close()
```

Resultatfil etter å ha kjørt programmet:

```
    0.75000000    0.29619813   -0.29619813   -0.75000000
    0.29619813    0.11697778   -0.11697778   -0.29619813
   -0.29619813   -0.11697778    0.11697778    0.29619813
   -0.75000000   -0.29619813    0.29619813    0.75000000
```

Oppsummering av fil-lesing og fil-skriving

```
infile  = open(filename, 'r')   # r = read
outfile = open(filename, 'w')   # w = write
outfile = open(filename, 'a')   # a = append

# Lese fra fil
line    = infile.readline() # les neste linje
filestr = infile.read()     # les resten av filen inn i en string
lines   = infile.readlines()  # les resten av filen inn i en liste
for line in infile:           # les resten av filen linje-for-linje

# Skrive til fil
outfile.write(s) # legg til \n hvis du trenger linjeskift

# Lukke fil
infile.close()
outfile.close()
```

Feilhåndtering

Feilhåndtering er et viktig tema i programmering. Det handler om hvordan programmet skal reagere når noe går galt under programeksekveringen. Ting som kan gå galt inkluderer f.eks.:

- Brukeren gir feil input, f.eks. ikke mange nok kommandolinje-argumenter, eller kommandolinje-argumenter av feil datatype.
- Programmet klarer ikke å åpne en fil fordi filen mangler eller er lesebeskyttet.
- Programmet prøver å lese en fil som er feilformattert.

Tre strategier

- **Proaktiv handling:** Legg inn sjekkpunkter i programmet som identifiserer feilsituasjoner før de faktisk hender. Eksempel: sjekk om `x` er et tall før man utfører `res = x**2`.
- **Rydd opp etterpå:** Vent til feilen oppstår og rydd deretter opp. For at dette skal fungere, må vi "fange" feilen før den fører til programstopp.
- **Ikke gjør noe:** Når en feil forekommer lar vi kjøretidssystemet håndtere det. Dette betyr essensielt at programeksekveringen stopper og det gis en standard feilmelding generert av systemet.

Hvorfor gjøre noe i det hele tatt?

Det er fristende å ikke gjøre noe (alternativ 3 over), fordi feilhåndtering betyr mer programkode og mer komplekse programmer. MEN det er gode grunner til å håndtere feil eksplisitt:

- Programmer som skal brukes av andre blir sjelden populære hvis de kræsjer med uforståelige feilmeldinger.
- Selv om du bare skal bruke programmet selv, kan det være at du finner ut at en lett forståelig feilmelding er nyttig og hjelper deg å finne feilen raskt.
- En bråstopp i programeksekveringen kan resultere i tap av data eller ødelagte filer. Å fange en feil tillater deg å rydde opp før programmet stopper (= terminerer).

Eksempel: en feilsituasjon

```
import sys
C = float(sys.argv[1])
F = 5./9*C + 32
print(F)
```

En bruker kan lett bruke programmet på feil måte:

```
Terminal> python celsius2fahrenheit.py
Traceback (most recent call last):
  File "c2f_cml.py", line 2, in ?
    C = float(sys.argv[1])
IndexError: list index out of range
```

Hva skjedde?

- Brukeren glemte å gi et kommandolinje-argument
- Listen `sys.argv` har dermed bare ett element, `sys.argv[0]`, som er programnavnet (`celsius2fahrenheit.py`)
- `sys.argv[1]` eksisterer ikke, så vi får `IndexError`

Eksempel (forts): proaktiv handling

For å unngå kjøretidsfeil med systemgenerert feilmelding, legger vi inn en test på om brukeren har gitt det nødvendige antall argumenter:

```
# Program celsius2fahrenheit.py

import sys
if len(sys.argv) < 2:
    print('Missing argument: degrees Celcius')
    sys.exit(1)    # Abort program execution
F = 9.0*C/5 + 32
print(f'{C}C is {F}F')
```

Denne gangen vil det bli generert en lett forståelig feilmelding dersom brukeren ikke gir kommandolinje-argumentet:

```
Terminal> python celsius2fahrenheit.py
Missing argument: degrees Celcius
```

Håndtering av feil ved å rydde opp etterpå

Normalt vil en kjøretidsfeil medføre at programkjøringen stopper umiddelbart. Vi kan unngå dette ved å legge den "utrygge" koden inn i en *try-except blokk*. Dette gjør det mulig for oss å selv avgjøre hva som skal skje når det oppstår en feil (= unntak).

Prototype for å teste unntak

```
try:
    <her er koden som kan gi feil>
except:
    <vi kommer hit hvis det oppstår en feil>
```

Hvis noe går galt i try-delen, vil Python generere ett unntak (= raise an exception) og programeksekveringen hopper direkte til except-delen.

Eksempel (forts.): rydde opp etterpå

Denne gangen prøver vi å lese inn verdien til `C` fra kommandolinjen. Hvis det skjer noe galt, forteller vi brukeren om det og stopper kjøringen:

```
# Program celsius2fahrenheit.py

import sys
try:
    C = float(sys.argv[1])
except:
    print('Missing argument: degrees Celcius')
    sys.exit(1)  # Abort the program
F = 9.0*C/5 + 32
print(f'{C}C is {F}F')
```

Kjøring av programmet:

```
Terminal> python celsius2fahrenheit.py
Missing argument: degrees Celcius

Terminal> python celsius2fahrenheit.py 21C
Missing argument: degrees Celcius
```

Testing av spesifikke typer unntak

Det er god programmeringsskikk å teste på spesifikke typer av unntak.

```
try:
    C = float(sys.argv[1])
except IndexError:
    print('Missing argument: degrees Celcius')
```

Hvis vi har en indeks som er utenfor det lovlige området i `sys.argv`, vil Python generere en `IndexError` exception, og kjøringen hopper til `except` blokken. Hvis det oppstår en annen type feil under kjøring, vil Python i dette tilfellet bare stoppe (abortere) programkjøringen.

Prototype for å teste spesifikke typer av unntak

```
try:
    <here we put the code that may fail>
except <exceptiontype 1>:
    <handle exception of type 1>
except <exceptiontype 2>:
    <handle exception of type 2>
...
```

Eksempel (forts.): test for spesifikke unntakstyper

```
# Program celsius2fahrenheit.py

import sys
try:
    C = float(sys.argv[1])
except IndexError:
    print('Missing argument: degrees Celcius')
    sys.exit(1)  # Abort execution
except ValueError:
    print('Argument is not a number')
    sys.exit(1)
F = 9.0*C/5 + 32
print('{C}C is {F}F')
```

Vi kjører programmet:

```
Terminal> python celsius2fahrenheit.py
Missing argument: degrees Celcius

Terminal> python celsius2fahrenheit.py 21C
Argument is not a number
```

Å generere våre egne unntak

Vi har sett at Python kan generere unntak (exceptions) dersom det oppstår feil under kjøring. Vi kan også generere våre egne unntak med `raise ExceptionType(message)`.

Eksempel:

```
import sys
def read_C():
    try:
        C = float(sys.argv[1])
    except IndexError:
        raise IndexError('Celsius degrees must be supplied')
    except ValueError:
        raise ValueError('Degrees must be a number')
    if C < -273.15:
        raise ValueError('Temperature is outside range')
    return C

try:
    C = read_C()
except (IndexError, ValueError) as e:
    print(e); sys.exit(1)
```

Vi kjører programmet

```
Terminal> python celsius2fahrenheit.py
Celsius degrees must be supplied

Terminal> python celsius2fahrenheit.py 21C
Degrees must be a number

Terminal> python celsius2fahrenheit.py -500
Temperature is outside range
```

Oppsummering av unntak (exceptions)

Handling exceptions:

```
try:
    <statements>
except ExceptionType1:
    <provide a remedy for ExceptionType1 errors>
except ExceptionType2, ExceptionType3, ExceptionType4:
    <provide a remedy for three other types of errors>
except:
    <provide a remedy for any other errors>
...
```

Raising exceptions:

```
if z < 0:
    raise ValueError(
        f'z={z} is negative - cannot do log(z)')
```

Moduler i Python

Hvis vi samler mange funksjoner i en enkelt programfil, har vi en modul. Moduler er nyttige for å samle relaterte funksjoner og tilhørende data på ett sted. Moduler kan også lett gjenbrukes av andre programmer.

Vi har ofte brukt moduler, f.eks. `math` og `sys`.

Eksempel: lage vår egen modul

Betrakt disse formlene for renteberegninger:

$$A = A_0 \left(1 + \frac{p}{360 \cdot 100}\right)^{n}, \tag{1}$$

$$A_0 = A \left(1 + \frac{p}{360 \cdot 100}\right)^{-n}, \tag{2}$$

$$n = \frac{\ln \frac{A}{A_0}}{\ln\left(1 + \frac{p}{360 \cdot 100}\right)}, \tag{3}$$

$$p = 360 \cdot 100 \left(\frac{A}{A_0}\right)^{1/n} - 1 \tag{4}$$

$$! \tag{5}$$

A_0: initial amount, p: percentage, n: days, A: final amount

Vi ønsker å lage en modul med disse fire funksjonene.

Først lager vi funksjoner for formlene

```
from math import log as ln

def present_amount(A0, p, n):
    return A0*(1 + p/(360.0*100))**n

def initial_amount(A, p, n):
    return A*(1 + p/(360.0*100))**(-n)

def days(A0, A, p):
    return ln(A/A0)/ln(1 + p/(360.0*100))

def annual_rate(A0, A, n):
    return 360*100*((A/A0)**(1.0/n) - 1)
```

Så lager vi modulfilen

- Samle de fire funksjonene i en fil `interest.py`
- Nå er `interest.py` en modul `interest`

Eksempel på bruk:

```
# Hvor lang tid tar det å doble mengden penger?

from interest import days
A0 = 1; A = 2; p = 5
n = days(A0, 2, p)
years = n/365.0
print 'Money has doubled after %.1f years' % years
```

Å legge til en testblokk i en modulfil

- Modul-filer kan ha en if-test på slutten som inneholder en *test blokk* for å teste eller demonstrere modulen
- Testblokken eksekverer ikke når modulen importeres inn i et annet program.
- Testblokken eksekverer *kun* når filen kjøres som et program

```
if __name__ == '__main__': # this test defines the test block
    <block of statements>
```

I vårt tilfelle:

```
if __name__ == '__main__':
    A = 2.2133983053266699
    A0 = 2.0
    p = 5
    n = 730
    print 'A=%g (%g) A0=%g (%.1f) n=%d (%d) p=%g (%.1f)' % \
          (present_amount(A0, p, n), A,
           initial_amount(A, p, n), A0,
           days(A0, A, p), n,
           annual_rate(A0, A, n), p)
```

Testblokker samles ofte i funksjoner

La oss lage en *testfunksjon* for det vi hadde i testblokken:

```
def test_all_functions():
    # Define compatible values
    A = 2.2133983053266699; A0 = 2.0; p = 5; n = 730
    # Given three of these, compute the remaining one
    # and compare with the correct value (in parenthesis)
    A_computed = present_amount(A0, p, n)
    A0_computed = initial_amount(A, p, n)
    n_computed = days(A0, A, p)
    p_computed = annual_rate(A0, A, n)
    def float_eq(a, b, tolerance=1E-12):
        """Return True if a == b within the tolerance."""
        return abs(a - b) < tolerance

    success = float_eq(A_computed,  A)  and \
              float_eq(A0_computed, A0) and \
              float_eq(p_computed,  p)  and \
              float_eq(n_computed,  n)
    assert success  # could add message here if desired

if __name__ == '__main__':
    test_all_functions()
```

Hvordan kan Python finne vår nye modul?

- Hvis modulen er i samme mappe som hovedprogrammet er alt enkelt og ok.
- Selvlagde moduler samles normalt i en bestemt mappe, f.eks. `/Users/ole/lib/python/mymods`
- Python må da fortelles at modulen ligger i denne mappen

Metode A: legg mappen til `PYTHONPATH` i `.bashrc`:

```
export PYTHONPATH=$PYTHONPATH:/Users/ole/lib/python/mymods
```

Metode B: legg mappen til `sys.path` i programmet:

```
sys.path.insert(0, '/Users/hpl/lib/python/mymods')
```

Metode C: flytt modulfilen inn i en mappe hvor Python allerede leter etter biblioteker.

Exercise 3.7

Evaluate a sum and write a test function

- Write a Python function sum_1k(M) that returns the sum $s = \sum_{k=1}^{M} \frac{1}{k}$.
- Compute *s* for the case *M* = 3 by hand and write another function test_sum_1k() that calls sum_1k(3) and checks that the answer is correct.

Filename: sum_func.

Exercise 4.1

Make an interactive program

Make a program that asks the user for a temperature in Fahrenheit degrees and reads the number; computes the corresponding temperature in Celsius degrees; and prints out the temperature in the Celsius scale.

Filename: f2c_qa.

Exercise 4.2

Read a number from the command line

Modify the program from Exercise 4.1 such that the Fahrenheit temperature is read from the command line.

Filename: f2c_cml.

Exercise 4.3

Read a number from a file

Modify the program from Exercise 4.1 such that the Fahrenheit temperature is read from a file with the following content:

```
4.12 Exercises 217
Temperature data
-----------------
Fahrenheit degrees: 67.2
```

Hint: Create a sample file manually. In the program, skip the first three lines, split the fourth line into words and grab the third word.

Filename: f2c_file_read.