FY1008_M01

September 10, 2025

1 Modul 01 : Introduksjon

Dette materialet tilsvarer i all hovedsak Kap. 2 fra læreboken.

1.1 Første standard eksempel "Hello World!"

[ ]:
```
print("Hello World")
```

For å kjøre cellen, trykk Shift+Enter.....

Men kommandoen kan man også lagre i en fil:

[ ]:
```
%run /home/ingves/Teaching/Courses/FY1008/Code/hello.py
```

1.2 Beregninger med formler

Standard regne operasjoner

[ ]:
```
# Dette er en kommentar....
# Det er LURT å bruke kommentarer for å beskrive hva koden gjør!
4+2, 4-2, 4*2, 4/2
```

[ ]:
```
# Potenser
2**3    # + 2 * 2 * 2 = 8
```

[ ]:
```
# En formel
5*( 1 + 2**3 )
```

1.3 Variabler og tilordninger

[41]:
```
# tilordning av variable
n = 2
a = 10
b = 5.0
c = n * b
d = a * c
tall = d**n
```

[ ]:
```
# Hva er verdien til variabelen "tall"?
print(tall)
```

VIKTIG: Alle variable har en type!!!

[ ]:
```
type(n)
```

[ ]:
```
# Hva er outputen fra denne kommandoen?
type(a), type(b), type(c)
```

[ ]:
```
# verdiene til variable blir lagret i minne
# Hva betyr dette?
id(a), id(b)
```

[ ]:
```
# help(id) eller id()
help(id)
```

Funksjonen "hjelp" fungere for alle python funksjoner

Assigment operator

[ ]:
```
t = 0.6
t = t + 0.1     # Dette gjør ikke mening matematisk!!!!
print(t)
```

Sammenlikning av variable

[ ]:
```
tall = 5
print(tall==5)
print(tall==3)
```

[ ]:
```
# hva med
print(a==10)
print(a==c)   # Hva skjer her.....?
```

1.4 Strings (tekst strenger)

[ ]:
```
# En streng
tekst = "Dette er en streng"
print(tekst)
```

[ ]:
```
# Dette er også en streng
tekst2 = tekst + " med tillegg"
print(tekst)
#print(tekst2)
```

[ ]:
```
# Ikke definert
"test" - "a"
```

[ ]:
```
text1 = "Dette er et tall"
text2 = "og verdien er 100!"
print(text1,text2)
```

[ ]:
```
tall = 100
# Alternative 1
print("Dette er et tall og verdien er", tall)
#Alternativ 2
print("Dette er et tall og verdien er {}!".format(tall))
# Alternativ 3
print("Dette er et tall og verdien er %d!" % (tall) )
```

[ ]:
```
# Format statement med flere input verdier
a = 2
b = 3
print(f"Tallene har verdiene a={a} og b={b}!")
#print("Tallene har verdiene a={} og b={}!".format(a,b))
#print("Tallene har verdiene a={A} og b={B}!".format(B=b,A=a))
#print("Tallene har verdiene a={A} og b={B}!".format(A=b,B=a))
```

[ ]:
```
# Format specifier: Mange alternative metoder for å oppnå det samme
pi =3.1415926
tall = 1000
print( "Pi har verdien {pi:8.4f} og tall={tall:e}")
print( f"Pi har verdien {pi:8.4f}  og tall={tall:e}")
print( "Pi har verdien %8.4f og tall=%4.2e" % (pi,tall) )
```

[ ]:
```
# Det finnes også mange innebygde funksjoner som kan brukes på strenger
text1 = "Dette er en streng!"
text1_split = text1.split()
print( text1_split )
print( type(text1_split))
```

1.5 Importere moduler

Mye av funksjonaliteten til Python kommer fra de mange module man kan importere for å lett få tilgang til nye funksjoner. Det er også via moduler at man mange biblioteker blir gjort tilgjengelig til brukerne av python.

[ ]:
```
# Hva er kvadratroten av et tall?
# La oss prøve funksjonen sqrt()
sqrt(10.)
```

[ ]:
```
# sqrt er ikke en kjent funksjon......
# Dette er rart, eller hva?

# Man må importere en modul for å få tilgang til sqrt.....
import math
```

```
x=math.sqrt(10.)
#
x, x*x
#print(x,x*x)
```

Det finnes mange måter å importere en modul og dens innhold på:

- from math import sqrt
- import math
- from math import *

[ ]:
```
# bare importer sqrt fra math
from  math import sqrt
#sqrt(10.)              # Fungerer
#math.sqrt(10.)         # Fungerer ikke
#sin(0.5)               # ikke definert
#math.sin(0.5)          # ikke definert
```

[ ]:
```
# importer ALT fra math modulen; Funksjonene heter math.function
import math
#sqrt(10.)              # Fungere ikke
#math.sqrt(10.)         # Fungerer
#sin(0.5)               # ikke definert
#math.sin(0.5)           # Fungerer
```

[ ]:
```
# Importer ALT fra math;  Funksjonene heter function UTEN math.
from math import *
#sqrt(10.)              # Fungerer
#math.sqrt(10.)         # ikke
#sin(0.5)               # Fungerer
#math.sin(0.5)          # ikke definert
```

ANDEFALING Alltid bruk importering ala "import math".

Da vet du alltid hvor de ulike funksjonen kommer fra!

[ ]:
```
import math
import numpy
print( math.sqrt(10.), numpy.sqrt(10) )
```

[ ]:
```
# Hva med dette?
from math import sqrt
from numpy import sqrt
sqrt(10.)
#
# Hvilken sqrt funksjon bruker vi her?
#help( sqrt )
```

Du vil ofte se importeringer av moduler hvor man gir dem kortnavn, f.eks.

- import numpy as np

Dette vil vi bruke ofte.

[ ]:
```
import numpy as np
np.sqrt(10.)      # Fungerer
#numpy.sqrt(10)    # Fungere ikke (dersom vi ikke allerede har importert numpy)
```

Eksempel med den **Gaussiske funskjonen** (se side 14 i læreboken) som har formen

$$f(x) = \frac{1}{\sqrt{2\pi}s} \exp\left[-\frac{1}{2}\left(\frac{x-m}{s}\right)^2\right]$$

[ ]:
```
m=0; s=2; x = 1.0;

# Ved bruk av modulen math
import math
f_math = 1/(math.sqrt(2*math.pi)*s) * math.exp(-((x-m/s)**2)/2)
print(f_math)

# via numpy
import numpy as np
f_np = 1/(np.sqrt(2*np.pi)*s) * np.exp(-((x-m/s)**2)/2)
print(f_np)
```

Merk deg prioriteten til de ulike aritmetiske operasjonene i eksempelet overfor.

[ ]:
```
# Hvordan slette en importert modul?
#
import numpy
print("Before :", numpy.sqrt(10.) )
#
#import sys
del numpy        # Sletter numpy
#
print("After :", numpy.sqrt(10.) )
```

Sentrale moduler som vi vil bruke ofte i dette emnet er som følger:

[ ]:
```
# Sentrale python moduler
import numpy as np                 # Numerical Python
import scipy as sp                 # Scientific Python
import matplotlib.pyplot as plt    # matplotlib (for plotting)
```

Hvordan finne informasjon om en modul?

- Googleing
- "hoover" muspekeren over modul-navnet i vs code
- help(math)
- print(dir(math))
- "pydoc math" i en terminal

- I vs code bruk autocomplete: math.<press tab)>

[27]:
```
import numpy
#help(numpy)
#print( dir(numpy) )
```

1.6 Fallgruver i programmering av matematikk!

Det er mange ulike utfordringer man må ta stilling til når man skal gjøre beregninger. Noen av de sentrale temaene er: * Anrundingesfeil * Divisjon på null

Avrundingsfeil

Matematisk har man at for alle verdier av variablen $x \neq 0$

$$\frac{1}{x} \cdot x = 1.$$

Om man gjør denne beregningen numerisk, er det på langt nær sikkert at man får det forventede svaret. Dette skyldes at i en datamaskin vil ikke alle tall kunne representeres eksakt, og derfor vil man få avrundingsfeil. Man kan øke presisjonen i en variable for delvis å bøte på dette, og man snakker ofte om *single*, *double* og noen ganger også *quadruple* presisjon. Vi vil nå illustrere dette.

[ ]:
```
# Man kan også forsøke med verdiene x1=0.499 og x2=0.501
x1 = 49.
x2 = 51.
#
v1 =  (1/x1) * x1
v2 =  (1/x2) * x2
#
print(f"{v1:.16f} {v2:.16f}")
```

[ ]:
```
# er verdien lik en?
print( v1 == 1 )
print( v2 == 1 )
```

WARNING: IKKE test flyttall (reelle tall) for likhet!

Istedet test om differensen er tilstrekkelig liten (relativt maskin presisjon : single 10^{-8}, double 10^{-16}).

[ ]:
```
exact = 1.
tol = 1e-14
#
print( abs(v1-exact) < tol )
print( abs(v2-exact) < tol )
```

1.7 Frivillige øvinger

Relevante øvinger for materialet vi har godt gjennom finner du i kap. 1 av boken til Hans Petter Landtangen (som er støttelitteratur for emnet). Det kan være verdt å gjøre noen av disse for å få litt ekstra øvelse.

Merk deg at denne boken bruker Python 2 slik at det vil være noen mindre justeringer for å bruke den under Python 3. For øvingene, bør ikke dette være et problem.