

IN1900 torsdag 20/8: Variabler og matematiske formler (kapittel 2)

Joakim Sundnes^{1,2}

Simula Research Laboratory¹

Universitetet i Oslo, Institutt for informatikk²

Aug 20, 2026

Hva vil du lære i IN1900?

- Generell programmering:
 - Å tenke som en programmerer
 - Å oversette matematikk til kode
 - Generelle konsepter som er felles for alle språk
 - Debugging, testing osv.
- Python (syntaks)
- Verktøy for programmering (tekstbehandler, terminalvindu)

- Forelesning (lysbilder), fra kapittel 2 i "Programmering for matematikk og naturvitenskap" (Haraldrud & Sundnes):
 - Variabler, tilordning (2.1)
 - Utskrift og formatering (2.2)
 - Datatyper (2.4, 2.6)
 - Regneregler (2.5)
 - Kommentarer (2.7)
 - Brukerinput (2.8)
 - Import av biblioteker (2.9)
- "Live programmering" av oppgave 1.3, 1.4 (og muligens 1.12) fra "A primer on..." av H.P. Langtangen

Viktige begreper

- Generelle:
 - *Variabler*
 - Datatyper
 - Tilordning
 - Definisjon/deklarasjon av variabler
 - Setninger
 - Uttrykk
 - (Objekter, funksjoner)
- Python-spesifikke:
 - Import
 - Bibliotek, modul
 - f-streng
 - Formatering av utskrift

Eksempel 1: En enkel matematisk formel

Høyde for en ball i vertikal bevegelse er gitt av formelen

$$y(t) = v_0 t - \frac{1}{2} g t^2$$

hvor

- y er høyden (posisjonen) som funksjon av tiden t
- v_0 er starthastigheten ved $t = 0$
- g er tyngdeakselerasjonen

Oppgave:

Gitt $v_0 = 5$, $g = 9.81$ og $t = 0.6$, beregn y og skriv ut verdien.

Hvordan skrive og kjøre programmet

- Et program er ren tekst, skrevet i en *editor* for ren tekst
- Bruk VSCode, Atom, Emacs, Vim eller Spyder (*ikke* MS Word!)

Steg 1. Skriv programmet i en tekstbehandler, her den enkle linjen

```
print(5*0.6 - 0.5*9.81*0.6**2)
```

Steg 2. Lagre programmet i en fil, for eksempel `ball.py`. (`.py` angir Python.)

Steg 3. Gå til et *terminalvindu* og naviger til mappen som inneholder programfilen.

Steg 4. Kjør programmet:

```
Terminal> python ball.py
```

Programmet skriver ut `1.2342` i terminalvinduet.

Aritmetiske uttrykk blir evaluert som i matematikk

- Eksempel: $\frac{5}{9} + 2a^4/2$, skrevet i Python som `5/9 + 2*a**4/2`
- Samme regler som i matematikk: arbeid deg term for term (addisjoner/subtraksjoner) fra venstre, beregn potenser først, deretter multiplikasjon og divisjon i hver term
- Bruk parenteser for å overstyre standardreglene - eller bruk parenteser for å gjøre det tydelig hvordan reglene virker:
 $(5/9) + (2*(a**4))/2$
- *Advarsel:* Feil i parenteser er en av de vanligste feilene i Python-programmer!

Lagre tall i variabler for å gjøre programmet mer lesbart

Eksempelprogrammet vårt så slik ut:

```
print(5 * 0.6 - 0.5 * 9.81 * 0.6**2)
```

Men fra matematikk er du vant til variabler, for eksempel,

$$v - 0 = 5, \quad g = 9.81, \quad t = 0.6, \quad y = v - 0t - \frac{1}{2}gt^2$$

Vi kan bruke variabler i et program også, og dette gjør det siste programmet enklere å lese og forstå:

```
v0 = 5  
g = 9.81  
t = 0.6  
y = v0 * t - 0.5 * g * t**2  
print(y)
```

Dette programmet har over flere linjer med tekst og bruker variabler, ellers gjør programmet de samme beregningene og gir samme output som det forrige programmet

Definere variabler i Python

- En variabel er en navngitt enhet for et dataelement i programmet vårt
- Variabler kan ha forskjellige *typer*, f.eks. heltall, flyttall (desimaltall), tekststreng osv.
- Teknisk sett er en variabel et navn for et sted i datamaskinens minne, der dataene er lagret
- I Python blir variabler definert ganske enkelt ved å skrive navnet deres og gi en verdi:

```
v0 = 5  
g = 9.81
```

- Denne måten å gi verdier til variabler på kalles *tilordning* (engelsk: *assignment*)

Variabler refererer til objekter, som har en spesifikk type:

```
a = 5           # a refererer til et heltall (int) objekt
b = 9           # b refererer til et heltall (int) objekt
c = 9.0          # c refererer til et flyttall (float) objekt
d = b * a        # d refererer til int * int => int objekt
e = b / a        # e refererer til int / int => float objekt
```

I Python bestemmes typen til en variabel automatisk, basert på innholdet på høyre side av tilordningen

Ulike former for "typing"

- Statisk typing: Typen til en variabel bestemmes på forhånd, kan ikke endres. Eksempler: Java, C++
- Dynamisk typing: Typen bestemmes av hva den inneholder, og kan byttes mens koden kjører. Eksempler: Javascript, Python
- Sterk typing: Konvertering mellom variabeltyper skjer ikke automatisk. Du får en feilmelding hvis du blander inkompatible typer, f.eks. prøver å plusse sammen en tekst med et tall. Eksempel: Python, Java.
- Svak typing: Konvertering skjer automatisk. Språket prøver å gjette hvilken type som ønskes, og konverterer automatisk. Eksempler: Javascript

Python er et *dynamisk og sterkt typet* språk.

Konvertering mellom datatyper

Vi kan konvertere mellom typer:

```
a = 3                # a er int
b = float(a)         # b er float 3.0
c = 3.9              # c er float
d = int(c)            # d er int 3
d = round(c)          # d er float 4.0
d = int(round(c))      # d er int 4
d = str(c)            # d er str '3.9'
e = '-4.2'           # e er str
f = float(e)          # f er float -4.2
```

Vi kan sjekke typen til objekter

Funksjonen `type` sjekker typen til et objekt:

```
a = 3                # a er heltall (int)
c = 3.9              # c er et flyttall (float)
h = 'Hello!'         # h er en streng (str)
print(type(a))       # Utskrift: <class 'int'>
print(type(c))       # Utskrift: <class 'float'>
print(type(h))       # Utskrift: <class 'str'>
```

Flere eksempler på variabler og typer

Strenger over flere linjer defineres med triple anførselstegn:

```
jack = """All work and no play makes jack a dull boy.  
All work and no play makes jack a dull boy.  
All work and no play makes jack a dull boy.  
"""  
print(type(jack)) # Utskrift: <class 'str'>
```

Såkalte *boolske* variabler har verdi True eller False:

```
a = True          #NB ikke "True"!  
print(a)          #Utskrift: True  
print(type(a))    #Utskrift: <class 'bool'>
```

Hva skjer i denne interaktive Python-sesjonen?

```
>>> a = '10'  
>>> b = 10  
>>> print(a*10)  
>>> 10101010101010101010  
>>> print(b*10)  
>>> 100
```

Det er stor fleksibilitet i valg av variabelnavn

- I matematikk bruker vi vanligvis én bokstav for en variabel
- Navnet på en variabel i et program kan inneholde bokstavene a-z, A-Z, understrek _ og sifrene 0-9, men kan ikke starte med et siffer
- Variabelnavn er store/små-bokstav-sensitive (f.eks. er a forskjellig fra A)

```
initial_velocity = 5
accel_of_gravity = 9.81
TIME = 0.6
VerticalPositionOfBall = initial_velocity*TIME - \
                          0.5*accel_of_gravity*TIME**2
print(VerticalPositionOfBall)
```

(Merk: skråstreken bakover tillater en instruksjon å bli fortsatt på neste linje)

Gode variabelnavn gjør et program enklere å forstå!

Noen ord er reservert i Python

Enkelte ord har en spesiell betydning i Python og kan ikke brukes som variabelnavn:

`and, as, assert, break, class, continue, def, del, elif, else, except, exec, finally, for, from, global, if, import, in, is, lambda, not, or, pass, print, raise, return, try, with, while, og yield.`

NB: Tilordning er ikke *helt* det samme som vanlig likhetstegn i matematikk

Tilordningssetninger evaluerer først uttrykket på høyre side av likhetstegnet, og tilordner deretter resultatet til variabelen på venstre side.

```
myvar = 10  
myvar = 3 * myvar    # = 30
```

Kommentarer kan være nyttige for å forklare programkoden

Program med kommentarer:

```
# program for å beregne høyden av en ball
# i vertikal bevegelse:

v0 = 5      # starthastighet
g = 9.81    # tyngdeakselerasjon
t = 0.6     # tid
y = v0*t - 0.5*g*t**2 # vertikal posisjon
print(y)

"""
Kommentarer kan også skrives i en streng
med tre anførselstegn.
"""
```

Merk:

- Alt etter # på en linje er en kommentar og ignoreres av Python
- Kommentarer brukes til å forklare hva variabler betyr, hvordan programmereren resonnerer da programmet ble skrevet, etc.
- Dårlige kommentarer sier ikke mer enn koden:
a = 5 # sett a til 5

Programstruktur og syntaks

Et program består av *setninger* (statements):

```
a = 1          # 1. setning (tilordningssetning)
b = 2          # 2. setning (tilordningssetning)
c = a + b      # 3. setning (tilordningssetning)
print(c)       # 4. setning (utskriftsetning / print-setning)
```

Hovedregel: én setning per linje, men flere setninger per linje er mulig med et semikolon mellom setningene:

```
a = 1; b = 2; c = a + b; print(c)
```

Setninger inneholder *uttrykk* (expressions):

```
c = a + b      # a+b er et uttrykk, c = a+b er en setning
```

Syntaks er reglene som sier hvordan koden kan bygges opp

Programmer må ha riktig syntaks, altså riktig bruk av reglene i dataspråket, og ingen skrivefeil!

Dette programmet har to syntaksfeil:

```
myvar = 5.2  
prinnt(Myvar)  
  
prinnt(Myvar)  
NameError: name 'prinnt' is not defined
```

Bare den første feilen oppdages og rapporteres, og programmet stopper. Vi må rette den feilen og gå videre til neste feil.

Mellomrom (whitespace) brukes for å formatere programteksten pent

- Mellomrom kan være viktige eller uviktige i Python-programmer.
- Disse setningene er ekvivalente (mellomrom betyr ingenting):

```
v0=3  
v0  = 3  
v0= 3  
v0 = 3
```

- Mellomrom i starten av en linje betyr noe:

```
v0 = 3  
  g = 9.81  #ugyldig, gir en feilmelding
```

- I Python blir slike mellomrom brukt til å gruppere kodeblokker sammen (mer om dette i kap. 3)

Eksempel 2: formatering av utskrift

Utskrift fra beregninger inneholder ofte tekst og tall, f.eks.,

Ved $t = 0.6$ s er $y = 1.23$ m.

Oppgave: Tilordne verdier til to variabler; $t = 0.6$ og $y = 1.2342$.

Skriv ut verdiene som vist ovenfor, med én desimal for t og to for y .

Pythons f-streng gir kontroll over utskriften

```
t = 0.6; y = 1.2342  
print(f'Ved t={t} s er y {y} m.')
```

F-prefikset foran strengen indikerer at dette er en spesiell type strengvariabel. Alt inne i krøllparentesene behandles som Python-uttrykk, og resultatet blir "satt inn" i strengen.

Vil du ha mer kontroll over formateringen?

Vi kan legge til en formateringsspesifikator inne i krøllparentesene, for å kontrollere antall desimaler osv:

```
t = 1.234567
print(f"Standard utskrift gir t = {t}.")
print(f"Vi kan angi presisjon: t = {t:.2}.")
print(f"Kontrollere antall desimaler: t = {t:.2f}.")
print(f"Eller plassen som brukes til utskriften: t = {t:8.2f}.")
```

Den siste linjen er nyttig for å skrive ut flere linjer i en tabell-lignende stil, for å sikre at tallene blir riktig justert under hverandre.

Lignende spesifikatorer finnes for heltall (ved å bruke d for digit):

```
r = 87
print(f"Heltallsutskrift som tar 8 tegn med plass: r = {r:8d}")
```

Formatering av flere linjer

Strenger over flere linjer, definert ved hjelp av tre anførselstegn, fungerer også med f-streng-formatering:

```
v0 = 5
g = 9.81
t = 0.6

print(f"""v0 = {v0} m/s
g = {g} m/s^2
t = {t} s
y = {v0*t - 0.5*g*t**2} m""")
```

Kjøring av programmet:

```
Terminal> python ball_print2.py
v0 = 5 m/s
g = 9.81 m/s^2
t = 0.6 s
y = 1.2342 m
```

Hvordan kan vi gi input til programmet når det kjører?

Ofte ønsker vi å kjøre samme program flere ganger, men med ulike verdier for noen av variablene. Da er det upraktisk å måtte redigere koden hver gang.

Eksempel 4: Input

Evaluer formelen

$$y(t) = v_0 t - \frac{1}{2} g t^2$$

og skriv ut resultatet (som over), men la brukeren av programmet velge verdien for t .

Enkleste løsning: Python-funksjonen `input` (alternative løsninger kommer senere i kurset).

Hva hvis vi trenger en mer avansert matematisk formel?

- Hva hvis vi trenger å beregne $\sin x$, $\cos x$, $\ln x$, osv. i et program?
- Slike funksjoner er tilgjengelige i ulike Python-biblioteker:
 - `math` for "vanlig" matematikk,
 - `cmath` for regning med komplekse tall
 - `numpy` for effektive numeriske beregninger.
- Nesten alle funksjoner i `math` finnes også i `numpy`, så vi bruker primært `numpy`
- Generelt: mye nyttig funksjonalitet i Python er tilgjengelig i biblioteker som må *importeres* i programmene våre

Oppgave: Evaluer uttrykket

$$Q = \sin x \cos x + 4 \ln x$$

for $x = 1.2$, og skriv ut resultatet på skjermen.

Ulike måter å importere

```
from numpy import *           #importer alt
from numpy import pi, sin     #importer det du trenger
import numpy                  #importer selve modulen/biblioteket
import numpy as np            #importer biblioteket med et alias
```

Import på formen `from numpy import *` (såkalt stjerneimport) er regnet som dårlig kodestil. Hvorfor? Hva kan gå galt?

Sammendrag av kapittel 2

- Variabler er navn for objekter, som har en *type*: `int`, `float`, `str`, osv.
- Vanlig likhetstegn (`=`) brukes til å *tilordne* verdier til en variabel: `a = 2`
- Gode variabelnavn gjør programet enklere å lese og forstå
- Aritmetiske operasjoner i Python: ledd for ledd (`+`/`-`) fra venstre mot høyre, potens før `*` og `/` - som i matematikk; bruk parenteser når det er tvil
- Matematiske funksjoner som $\sin x$ og $\ln x$ importeres fra `numpy`-biblioteket (eventuelt `math`)
- Bruk `f`-strenger for å kontrollere utskriftsformatet for tekst og tall