

enkel_input_output

November 9, 2019

1 Input og output fra programmer

Vi skal først se på hvordan vi kan skrive noen enkle programmer som leser inn input fra brukeren, gjør noe med denne inputen, og skriver noe ut igjen til brukeren.

Vi starter med det aller enkleste, klassiske første programmet, nemlig et program som skriver "hello, world!" ut til brukeren når programmet kjøres.

1.1 Hello, World!

Dette programmet skriver ut teksten "Hello, world!" ut til brukeren. Prøv å forandre på teksten mellom hermetegnene å se hva som skjer.

```
[4]: print("Hello, World!")
```

Hello, World!

Aktivitet: Det anbefales at du kjører programmet slik at du vet hvordan du kjører python-programmer på din datamaskin.

1.1.1 Kort "obduksjon" av programmet

Programmet sender inn en *streng* "Hello, world!" til *funksjonen* print. Funksjonen print virker som et bindeledd mellom programmet og standard output. I mer avanserte anvendelser kan det være hendig å endre på denne - men vanligvis er det greit å bruke print til å skrive tekst ut til kommandolinjen eller slik du ser her

1.1.2 Konsepter

Datatypen streng Du har møtt på datatypen str. En streng i python kan du tenke på som tekst. Husk at for en datamaskin er det bare en remse med bokstaver - helt i bunnen er det en remse med bytes. Teksten 'Hello, world!' er for eksempel følgende remse med bytes:

```
01001000 01100101 01101100 01101100 01101111 0101100 00100000 01110111 01101111  
01110010 01101100 01100100 00100001
```

Som du kanskje synes, er det ikke spesielt lesbart. Derfor er det skrevet programmer som gjør dette om til tekst du kan lese, uten at du trenger å tenke noe på hvordan programmet fungerer. Du trenger vanligvis ikke tenke på at programmet en gang eksisterer. Dette kalles i informatikken for *abstraksjon*, og er et viktig konsept vi kommer tilbake til senere.

Funksjoner Du har møtt din første funksjon, nemlig print. Funksjonen print tar inn en streng og skriver det ut til kommandolinjen/under cellen. Funksjoner generelt i python tar inn en verdi og gjør noe med eller noe som avhenger av disse verdiene.

Remsen med bytes for å representere strengen Hello, world! er generert med med programmet under. Ikke bruk tid på å gruble over koden enda, du vil om tid og stunder forstå helt greit hva som står her.

```
[2]: # generer bytes for tegnene i strengen 'Hello, world!' slik de samsvarer i
      ↳ standarden utf8
for byte in map(bin, bytearray('Hello, world!', 'utf8')):
    print(byte, end=" ")
```

```
0b1001000 0b1100101 0b1101100 0b1101100 0b1101111 0b1011100 0b1000000 0b1110111
0b1101111 0b1110010 0b1101100 0b1100100 0b1000001
```

Orienteringsstoff om strenger og utf8-encoding Standarden utf-8 bruker én byte for de fleste vanlige tegn. For nordiske tegn bruker den to bytes, og for mange spesielle tegn kan den bruke tre eller fire bytes.

I tabellen under ser du noen eksempler med bytene representert som binære tall.

bokstav	byte nummer 1	byte nummer 2	byte nummer 3	byte nummer 4
a	0b1100001			
b	0b1100010			
c	0b1100011			
ä	0b11000011	0b10100100		
æ	0b11000011	0b10100101		
ø	0b11000011	0b10111000		
å	0b11000011	0b10100101		
à	0b11000011	0b10100000		
â	0b11000011	0b10100010		
	0b11001111			
	0b10000000	0b11100010	0b10000100	0b10110101
	0b11100010	0b10001000		
	0b10000111	0b11110000	0b10100011	0b10110100
				0b10101111

1.1.3 Lese input fra bruker

Vi kan bruke kommandoen input til å lese input fra en bruker. Under ser du et eksempel

```
[3]: # les input fra bruker og skriv ut en hilsen me brukerens navn
name = input('Hva er navnet ditt? Skriv her: ')
print("Hallo " + name + "!")
```

```
Hva er navnet ditt? Skriv her: Eindride
```

```
Hallo Eindride!
```

1.1.4 Kort obduksjon

Programmet ignorerer teksten bak tegnet #. Teksten bak dette tegnet er en kommentar for å gjøre koden lettere å lese for mennesker.

Programmet kaller på en funksjon input. Argumentet til input er strengen Hva er navnet ditt? Skriv her: \. Denne strengen blir skrevet ut til kommandolinje / under cellen. Poenget med

denne beskjednen er å informere brukeren om hva slags input som skal skrives inn. Deretter lagres det brukeren mater inn i programmet fra kommandolinjen i *variabelen* name.

Det anbefales nå at du gjør oppgave 1, 2 og 3

1.1.5 Flere konsepter

Variabler En variabel i Python kan du tenke på som navnet på en adresse i minnet til datamaskinen. Å lagre data i variabler gjør koden enklere å lese, og vi kan kombinere dataene med andre data på mange ulike måter.

Legg til deg gode vaner, og start å bruke variabler allerede nå!

Store programmer kan være vanskelige å lese. For å gjøre programmene mer lesbare, er det viktig at du lager *beskrivende* navn til variablene dine. Dersom variabelen inneholder navnet til en kunde, kan den f.eks hete navn, men et enda bedre navn er da kunde_navn.

NB! Et lite aber er når vi jobber med matematikk - da ønsker vi å velge variablene slik at koden minner mest mulig om de matematiske uttrykkene.

Kommentarer Kommentarer i python starter med tegnet #. Kommentarene blir ikke kjørt som kode, men ignorert av datamaksinen når du kjører programmet. De gjør det lettere å forstå koden! Dersom kommentaren skal gå over flere linjer, bruker vi en såkalt doc-string. Du kan skrive dem med tredoble anførselstegn (dobbel x3 `"""` eller enkel x3 `'''`).

Du kan bruke kommentarer til å holde orden på enheter, og fortelle i korte trekk hva en samling kommandoer skal gjøre. Skriv heller for mange kommentarer enn for få! Det er aldri et problem at kode er *for godt dokumentert*. Blir dokumentasjonen for lang, har de fleste *editorer* der koden skrives funksjonalitet for å “kollapse” tekst slik at du ikke trenger å bla i teksten, men kan åpne det du trenger å se. En svært vanlig “nybegynnerfeil” er å slurve med kommentarer, for deretter å bruke lang tid på å forstå egen kode noen dager eller uker senere. Å skrive gode kommentarer er en ferdighet på lik linje med selve programmeringen, og blir svært viktig når man arbeider i team med andre mennesker.

Under ser du et eksempel på kode der vi har skrevet en del kommentarer. Prøv å bruke kommentarene til å forstå hva koden gjør i hvert steg. Som du ser blir ikke tabellen helt “fint” formatert. Vi skal senere se på hvordan dette kan gjøres.

```
[46]: """
Dette programmet viser noen egenskaper ved strenger:
Hvis vi multipliserer en streng med et heltall n, gjentas strengen n antall_
    ↳ ganger.
Når to strenger a og b legges sammen, føyes teksten i b på teksten i a.
For eksempel blir 'Martin' + ' Andersen' til strengen 'Martin Andersen'

Koden skriver ut en tabell med fornavn, telefonnummer og adresse
"""

n = 10 # antall mellomrom
mellomrom = "    "*n
```

```

# Tabelloverskrifter
tlf = "Telefon"
adr = "Adresse"
nvn = "Navn"

# Innhold i tabellen.
# Merk at etter semikolon kan en ny kommando skrives uten å starte en ny linje
navn1 = "Kåre"; telefon1 = "91248953"; adresse1 = "Pøbelringen 5"
navn2 = "Marie"; telefon2 = "39025847"; adresse2 = "John Tullings Gate 10"

print(nvn + mellomrom + tlf + mellomrom + adr) # tabelloverskrifter
print('--'*35)
print(navn1 + mellomrom + telefon1 + mellomrom + adresse1)
print(navn2 + mellomrom + telefon2 + mellomrom + adresse2)

```

Navn	Telefon	Adresse
Kåre	91248953	Pøbelringen 5
Marie	39025847	John Tullings Gate 10

Minne på datamaskinen Det kan være lurt å tenke på minnet i datamaskinen som en lang remse med hus som hver inneholder 8 bokser. I hver av disse boksene kan vi sette enten verdien 0 eller 1. Vi kaller disse verdiene **bits**

Et hus er da en **byte**, og vi kan tenke på navnet til en variabel som adressen til huset. Nå er det ikke likevel alltid helt så enkelt, da vi for eksempel trenger åtte bytes til å representere for eksempel et flyttall. Men så lenge disse ligger inntil og i en bestemt retning fra vår adresse, går dette helt greit!

Bits i minnet til datamaskinen representert ved at en transistor styrer strøm til en kondensator. Ved å måle spenningen over kondensatoren vet man om transistoren er slått “av eller på”, og dermed kan man lese av verdien 1 når spenningen er over 50% av en “maksverdi”, og 0 ellers. Du kan lese mer her: <https://computer.howstuffworks.com/ram.htm>.

1.2 Bruk av ipython som en kalkulator

Vi kan bruke ipython som en kalkulator. Hvis du har en åpen terminal, kan du skrive inn kommandoen ipython. Da får du noe som ser ut som i kodesnutten under Ofte ønsker vi å bruke matematiske funksjoner som ikke er med i python fra før. Da må vi importere dem. Det kan vi gjøre med kommandoen `from pylab import *`. Vi importerer da alle kommandoer fra pakken pylab.

1.2.1 Eksempel

Under har vi regnet ut verdien av uttrykket

$$5\sqrt{2} + \left(\frac{3}{4}\right)^2.$$

```
In [1]: from pylab import *
```

```
In [2]: 5*sqrt(2) + (3/4)**2
```

```
Out[2]: 7.6335678118654755
```

```
In [3]: pi
```

```
Out[3]: 3.141592653589793
```

```
[4]: from pylab import *  
     5*sqrt(2) + (3/4)**2
```

```
[4]: 7.6335678118654755
```

Sammensatte uttrykk Regn ut verdien av uttrykket

$$E = gv^n + \frac{b}{|u - w| + 1},$$

for $g = 14000$, $v = 1.0275$, $n = 20$ og $b = 2.3 \cdot 10^4$.

I eksempelet under har vi definert tre variable g , v , n , og b . Vi gjør noen beregninger med disse og lagrer dem inn i variabelen E .

Python 3.6.7 (default, Jul 2 2019, 02:21:41) [MSC v.1900 64 bit (AMD64)]

Type 'copyright', 'credits' or 'license' for more information

IPython 7.8.0 -- An enhanced Interactive Python. Type '?' for help.

```
In [1]: g = 14000
```

```
In [2]: v = 1.0275
```

```
In [3]: n = 20
```

```
In [4]: b = 2.3E4
```

```
In [5]: u = 45
```

```
In [6]: w = 36
```

```
In [7]: E = g*v**n + b/(abs(u - w) + 1)
```

```
In [8]: E
```

```
Out[8]: 26385.99803800328
```

Hvis du jobber i en notebook, kan du gjøre slike interaktive beregninger i en kode-celle. Prøv deg på noen av oppgavene under

Aktivitet: oppgave 4 til oppgave 10

1.3 Flyttall

Til å starte med kan du tenke på flyttall som et desimaltall lagret på datamaskinen. På engelsk kalles flyttall for *floating point number*, eller i korthet bare *float*. Derav navnet på datatypen i eksempelet over. Likevel er det noen spesielle egenskaper ved flyttall det er vel verd å være oppmerksom på.

Eksempelet under illustrerer et av problemene ved flyttall. Hvordan kan vi forklare hva som skjer - og er dette generelt litt skummelt?

```
[6]: inaccurate_approximation = 1E23*(1/3)

# Skriv ut tallet inaccurate_approximation
# med fire desimalers nøyaktighet
print(f'{inaccurate_approximation:.4f}')
```

333333333333333327740928.0000

Dette ser kanskje ikke lovende ut. Men med nærmere inspeksjon er det kanskje ikke så ille som det ser ut som. Vi må telle 15 siffer til høyre for det første sifferet. Den relative feilen er da omtrentlig av størrelsesorden

$$\epsilon \approx \frac{10^{15}}{10^{22}} = 10^{-7}.$$

Det bruker å gå fint å gjøre beregninger med flyttall - men vi skal senere se litt på noen tilfeller der det likevel ikke går fullt så bra.

1.4 Heltall

Heltall har i python typen `int`. I motsetning til flyttall, kan alle heltall representeres nøyaktig, så lenge de ikke er altfor store. Python har ingen “hard grense” på hvor store heltall kan være slik mange andre språk har, men begrensningen går på hvor stor plass det er tilgjengelig i minnet til datamaskinen.

Du kan gjøre en `int` `i` om til en `float` `f` ved kommandoen `f = float(i)`. Hvis du skal runde av et desimaltall `f` til et heltall `i`, kan du bruke kommandoen `i = int(round(f))`.

1.5 Regnerekkefølge

Du lurer kanskje på hvordan python prioriterer mellom operasjonene `+`, `-`, `*` og `/`. Regnerekkefølgen er den “vanlige”: 1. Paranteser 2. Eksponenter, røtter, funksjoner 3. multiplikasjon og divisjon 4. addisjon og subtraksjon

Utrykkene leses fra venstre til høyre. Når to operasjoner har lik prioritet, utføres den lengst til venstre først.

1.5.1 Eksempel

```
# Programmet skal regne ut kraften i en hydraulisk jekk på én av stemplene.
# Definer parametere
```

```
A1 = 10
F = 30
A2 = 15
```

```
# Regn ut krafta
F2 = F/A1*A2
```

Utrykket $F/A1*A2$ leses fra venstre til høyre, og operasjonene har lik prioritet. Altså regnes først $F/A1$ ut. Deretter multipliseres dette med $A2$.

1.6 Tall som input fra bruker

Hvis vi ønsker å lese inn et tall, må vi fortelle Python at det er et tall vi er ute etter

```
[ ]: name = input('Hva er navnet ditt? Skriv her: ')
      print("Hallo " + name + "!")

      height_centimeters = input('Hvor høy er du? Gi svaret i cm uten enheter.')

      #gjør strengen height_centimeters om til en float
      height_centimeters = float(height_centimeters)

      height_meters = height_centimeters/100

      print("Høyden din i meter er: ", height_meters)
```

1.7 Input fra kommandolinje

Når vi lager programmer vi kjører ofte, er det vanligvis lettere å gi input fra kommandolinjen. Til dette kan vi bruke pakken `sys` Under ser du et utsnitt av et program `add_args.py` som tar inn to tall fra kommandolinjen og skriver ut summen av dem.

```
[42]: !ls
```

```
add_args.py
enkel_input_output.ipynb
hello_you.py
```

```
[45]: %%writefile add_args.py
      import sys

      a = sys.argv[1] # legg første argument fra kommandolinjen inn i a
      a = float(a)    # omgjør a fra streng til desimaltall

      b = sys.argv[2] # Legg andre argument fra kommandolinjen inn i b
      b = float(b)
```

```
print(a + b)
```

Overwriting add_args.py

Vi kan nå kjøre programmet. Enten fra notebooken som i cellen under, eller fra en terminal. Tilsvarende kjøring fra kommandolinjen ville vi fått med kommandoen `python add_args.py 1.5 2.8`

```
[49]: run add_args.py 1.5 2.8
```

4.3

Aktivitet: Oppgaver 11 - 15

1.7.1 Orienteringsstoff: Et veldig typisk eksempel

Vi tar et kort eksempel på hvordan input kan leses fra kommandolinjen. Vi skal i de neste notebookene se på hvilke konsepter som brukes i programmet slik at du snart skal kunne skrive slike programmer selv

```
[1]: %%writefile aplot_andregradsfunksjon.py
from pylab import *
import sys

#programmet plotter grafen til en funksjon  $a*x**2 + b*x + c$ 

# les inn nødvendige parametere for å representere funksjonen
a = float(sys.argv[1])
b = float(sys.argv[2])
c = float(sys.argv[3])

# les inn nødvendige parametere for xmin og xmax for plottet.
# verdimengden behøver vi ikke bekymre oss om for plotting
xmin = float(sys.argv[4])
xmax = float(sys.argv[5])

N = ceil(abs(xmax - xmin)*100) # hundre punkter i et intervall med bredde 1

# Lag et intervall (array) med flyttall fra xmin til xmax med N punkter
x = linspace(xmin, xmax, N)

# beregn en y-verdi for hvert av flyttallene i arrayet x
y = a*x**2 + b*x + c

plot(x, y)

#pynt på grafen
xlabel('x', fontsize='24')
```



```

xticks(fontsize='16', rotation=40)
ylabel('y', fontsize='24')
yticks(fontsize='16')
title(fr'$a x^2 + b x + c$', fontsize='32') # skriv funksjonsuttrykket i
→tittelen
grid()

# gcf: get current figure
fig = gcf()
fig.set_size_inches(10, 7)

show()

```

Overwriting `aplot_andregradsfunksjon.py`

```
[3]: run aplot_andregradsfunksjon.py -2 2 5 -2 3
```

enkel_input_output_files/enkel_input_output_52_0.png

Det er nå enkelt å endre på parametrene funksjonen får inn. Prøv selv!

1.8 Orienteringsstoff: Bruk av pakken `argparse` når mange argumenter skal leses inn

du synes kanskje at det kan være vanskelig å holde styr på hvilke argumenter som skal leses inn til programmet over. Vi kan lette dette ved å bruke pakken `argparse`. Da kan vi kjøre programmet ved å bruke *key-value pairs*.

```
run plot_andregradsfunksjon.py --a 0 --b 2 --c 5 --xmin -2 --xmax 3
```

Det vil da være lettere for brukeren å holde styr på hvilke argumenter som er hva

```
[38]: %%writefile plot_andregradsfunksjon.py

from pylab import *
import argparse

parser = argparse.ArgumentParser() # lag et objekt parser som kan lese input fra
→kommandolinjen

```

```

# definer forventede argumenter fra kommandolinjen med key-value pairs
parser.add_argument('--a', type=float)
parser.add_argument('--b', type=float)
parser.add_argument('--c', type=float)

parser.add_argument('--xmin', type=float)
parser.add_argument('--xmax', type=float)

# antall grid-punkter i et intervall med lengde 1
parser.add_argument('--grid_density', type=int, default=100)

# les argumentene fra kommandolinjen
args = parser.parse_args()
a, b, c, xmin, xmax, grid_density = args.a, args.b, args.c, args.xmin, args.
    ↪xmax, args.grid_density

N = round(abs(xmax - xmin)*grid_density) # antall punkter i gridet

x = linspace(xmin, xmax, N)
y = a*x**2 + b*x + c

plot(x, y)

xlabel('x', fontsize=16); xticks(fontsize=14)
ylabel('y', fontsize=16); yticks(fontsize=14)

title(fr'$\{a\}x^2 + \{b\}x + \{c\}$', fontsize=20) # skriv funksjonsuttrykket i
    ↪tittelen
grid()

# gcf: get current figure
fig = gcf()
fig.set_size_inches(10, 7)

show()

```

Overwriting plot_andregradsfunksjon.py

```
[39]: run plot_andregradsfunksjon.py --a -0.5 --b 3 --c 2 --xmin -1.5 --xmax 7
```

enkel_input_output_files/enkel_input_output_56_0.png

[]:

1.9 Orienteringsstoff: Lesing av data med loadtxt

Når vi gjør store simuleringer eller har måledata, kan vi bruke pakken `loadtxt` fra `numpy` til å lese store filer inn i såkalte *arrays*. (Du kan for øyeblikket tenket på de som lange remser med tall lagret i minnet på datamaskinen).

Aktivitet En av dine venner har sett video på youtube og ønsker å hoppe fra verdensrommet med en romdrakt. Felix Baumgartner hoppet i 2012 fra 39,045 meter(<https://www.youtube.com/watch?v=FHtvDA0W34I>). Vennen din ønsker å doble dette, men du er usikkert på om det er trygt. Du har fått tilgang til simuleringer i en fil `jump_simulation_data_time_altitude.txt` og `jump_simulation_data_time_velocity.txt`. Bruk `numpy.loadtxt` til å laste data fra filene inn i arrays og plot dem.

Under ser du hvordan `jump_simulation_data_time_altitude.txt` lastes inn i arrays `t` og `v`.

```
[14]: from pylab import *
      t, h = loadtxt('jump_simulation_data_time_altitude.txt', dtype=float,
      ↪delimiter=',')
```

a) Last inn hastighetene fra filen `jump_simulation_data_time_velocity.txt`. Plot både hastigheten og høyden som funksjoner av tiden.

Vurder om det er trygt for vennen din å hoppe fra 80000 meter.

[]:

b) Bruk pakken `numpy.savetxt` til å lagre dataene fra arrayene i én fil. kallet er da `savetxt('filnavn', [t, v, h], delimiter=',', header='time, velocity, altitude')`

1.10 Litt om skrivning til fil

De finnes flere måter å skrive til fil. Man kan skrive linje for linje eller bruke pakker slik som her.

Hvis du skal skrive store mengder data anbefales `savetxt` eller å bruke modulen `shelve`. Med modulen `shelve` kan du også lagre ethvert type objekt til fil. Du kan dermed lagre tilstanden programmet ditt er i og kjøre det senere, eller hente opp egendefinerte datastrukturer.

Hvis du er nysjerrig kan du allerede nå se hvordan man kan skrive informasjon til fil i denne videoen til [Corey Shafer](#).

Vi skal senere se på hvordan dette gjøres i litt mer detalj

2 Oppsummering

2.1 Gode vaner

- **variabler** En variabel i python kan inneholde alle typer objekter. De gjør det lettere å lese programmet, og gir det mer fleksibilitet til å bygge opp programmet ditt
- **variabelnavn** Velg gode navn til variablene dine. Er det en matematisk formel, bør du gjøre koden mest mulig lik de matematiske uttrykkene. Er det noe annet, velg et nøye uttenkt og beskrivende navn til variabelen din
-

2.2 kommentarer En kommentar i python starter med en hashtag (#) eller kan gis som en doc-string. Kommentarer gjør koden din lettere å lese for andre mennesker - ikke minst deg selv om 2-3 dager, uker eller måneder.

2.3 Vanlige datatyper

- **str** - datatypen streng i python. En remse med tekst.
- **float** - datatypen flyttall i python. En etterlikning av desimaltall. I motsetning til reelle tall finnes det bare et endelig antall flyttall på datamaskinen siden den bare kan lagre et endelig antall bits. Derfor får vi *avrundingsfeil*
-

2.4 int - datatypen integer eller et heltall. I motsetning til flyttall kan disse representeres nøyaktig!

2.5 Ofte brukte kommandoer og pakker

- `print('tekst i en streng markert med enkle hermetegn')` skriv ut strengen til terminal/kommandolinje/under kode-celle.
- `input('tekst som blir skrevet ut til bruker)` skriv ut streng til bruker, og les inn input til programmet fra bruker.
- `import sys` importerer pakken sys
- `sys.argv` inneholder argumenter til programmet fra kommandolinjen. Kan gis før programmet kjører og er lettere å endre hvis programmet kjøres mange ganger
- `from pylab import *` - importerer alle funksjoner og kommandoer fra pakken pylab.
- `import pylab as pl` - importerer pakken pylab. Du får tilgang til alle kommandoer fra pylab ved å f.eks skrive `x = pl.linspace(-1, 1, 100)`
- `%%writefile filename.py` skriver innholdet i en celle til filen `filename.py`
- `%run filename.py` kjører programmet `filename.py` fra en celle

2.5.1 Omgjøring av datatyper

- `float(<input: streng, int, bool>)` omgjør argumentet til et flyttall
- `int(<input: streng, bool>)` omgjør argumentet til et heltall

2.6 Avrunding

- `round(<input: float f>, <input: int i>)` rund av et flyttall f til i desimaler etter komma.
- Ofte sett i plotting: $N = \text{int}(\text{ceil}((x_{\max} - x_{\min})/dx))$ for å beregne antall punkter langs en akse i et intervall $[x_{\min}, x_{\max})$ med avstand dx mellom hvert punkt

2.7 Python som kalkulator

- `a**b` beregner a^b
- `a*b` beregner ab
- `a + b` beregner $a + b$
- `a - b` beregner $a - b$
- `(a - b)**2/4` beregner $\frac{(a - b)^2}{4}$
- `sqrt`, `ln`, `log10`, `exp`, `sin`, `cos`, `tan`, `arcsin`, `arccos`, `arctan`, `sinh`, `cosh`, `tanh` funksjoner tilgjengelig fra `pylab`
- Beregn n 'te rot $\sqrt[n]{x} = x^{1/n}$ ved å skrive `x**(1/n)`

2.8 Plotting

- `linspace`
- `plot(x, y)`
- `xlabel('navn på x-akse')`
- `ylabel('navn på y-akse')`
- `title('tittel')`
- `grid` vis rutenett
- `show()` vis plottet
- `figure()` lag nytt plot

2.9 Generelle læringsbeskrivelser

Du har lært om følgende: * Skrive verdien av en variabel ut til brukeren * Lese inn data fra brukeren, både "interaktivt" med `input` og fra kommandolinja ved å bruke `sys.argv` * Gjøre om mellom datatyper, f.eks fra `int` til `float` eller fra `str` til `float` * Bruke python som kalkulator til enkle beregninger

3 Oppgaver

3.0.1 Oppgave 1

Spør brukeren om navnet til brukeren. Skriv ut en hyggelig hilsen til brukeren som inneholder brukerens navn

3.0.2 Oppgave 2

I denne oppgaven skal du lese inn fornavn, etternavn og alder fra brukeren. Du skal kombinere disse for å skrive ut en hyggelig hilsen til brukeren. Du kan starte programmet med noe som ligner dette:

3.0.3 Oppgave 3

Skriv et program som spør brukeren om etternavn, fornavn og alder. Alle strenger har en funksjon `lower` som kan brukes på følgende måte:

```
[ ]: etternavn = 'Marchussen'  
      etternavn.lower()
```

Programmet skal skrive ut etternavnet med små bokstaver

```
[ ]:
```

3.0.4 Oppgave 4

Regn ut følgende verdier med python eller ipython

- `sqrt(3)`
- `log10(10)`
- `log10(1)`
- `2*pi`

```
[3]: # Du kan skrive koden din her
```

3.0.5 Oppgave 5

a) Hva gjør koden under?

```
In [1]: from pylab import *
```

```
In [2]: rad2deg(2*pi)  
Out[2]: 360.0
```

```
In [3]: rad2deg(3*pi)  
Out[3]: 540.0
```

```
In [4]: rad2deg(pi)  
Out[4]: 180.0
```

```
In [5]: rad2deg(pi/2)  
Out[5]: 90.0
```

```
In [6]: rad2deg(pi/4)  
Out[6]: 45.0
```

```
In [7]: rad2deg(0)  
Out[7]: 0.0
```

b) Hva tror du `rad2deg(pi/3)` vil gi ut? Du kan bruke cellen under eller i en egen ipython-terminal til å sjekke svaret ditt.

```
[ ]: from pylab import *
```

3.0.6 Oppgave 6

a) Regn ut `deg2rad(0)` og `deg2rad(180)`

Hvis vi gir grader som input til trigonometriske funksjoner i python, får vi ikke riktig svar:

```
In [2]: from pylab import *  
In [2]: sin(180)  
Out[2]: -0.8011526357338304
```

Hvis vi bruker funksjonen `deg2rad` på gradene slik at vi får vinkelen i radianer, går det bra.

b) Bruk funksjonen `deg2rad` og regn ut to eller flere av verdiene under

$$\sin(30) \tag{1}$$

$$\sin(45) - \frac{\sqrt{2}}{2} \tag{2}$$

$$\sin(60) - \frac{\sqrt{3}}{2} \tag{3}$$

$$\sin(90) \tag{4}$$

$$\tag{5}$$

3.0.7 Oppgave 7

I denne oppgaven undersøker vi hva funksjonen `abs` gjør.

- Regn ut `abs(-1)`, `abs(-2)`, `abs(-5)`. Hva gjør funksjonen?
- Skriv inn kommandoen `abs?` i en ipython-terminal og les en setning om funksjonen `abs`

```
[5]: abs?
```

```
[1;31mSignature: [0m [0mabs[0m[1;33m([0m[0mx[0m[1;33m, [0m [1;33m/[0m[1;33m) [0m[1;33m[0m[1;33m[0m[  
[1;31mDocstring: [0m Return the absolute value of the argument.  
[1;31mType: [0m builtin_function_or_method
```

3.0.8 Oppgave 8

a) La $G = 450000$, $V = 0.88$ og $n = 6$. Bruk ipython til å beregne verdien av N , som er gitt ved likningen

$$N = GV^n.$$

[3]: `# Du kan skrive programmet ditt her`

b) Lag et eksempel på en praktisk situasjon som passer til likningen i oppgave a).

3.0.9 Oppgave 9

a) La $g = -9.81$, $h_0 = 40$ og $t = 4.5$.

La

$$h = h_0 + \frac{1}{2}gt^2.$$

Fullfør ipython-økten under slik at den regner ut verdien av h

In [1]: `g = -9.81`

In [2]: `h_0 = 40`

In [3]: `t = 4.5`

b) Hvilken praktisk situasjon kan likningen i oppgave a) beskrive?

[]:

3.0.10 Oppgave 10

Se på ipython-økten under.

Kan du se hva som skjer?

In [1]: `1%4`

Out[1]: `1`

In [2]: `2%4`

Out[2]: `2`

In [3]: 3%4
Out[3]: 3

In [4]: 4%4
Out[4]: 0

In [5]: 5%4
Out[5]: 1

In [6]: 6%4
Out[6]: 2

In [7]: 7%4
Out[7]: 3

In [8]: 8%4
Out[8]: 0

In [9]: 9%4
Out[9]: 1

3.0.11 Oppgave 11

a)

- 1) Forklar hva programmet under gjør
- 2) Modifiser programmet slik at verdien av variabelen LHS blir skrevet ut

```
[24]: x = 7  
      y = 13  
  
      LHS = x*y + x/(y + 1)
```

b) Skriv et program som tar inn to tall x og y fra brukeren, og regner ut en verdi for uttrykket $xy + x + y$

```
[ ]:
```

3.0.12 Oppgave 12

Skriv et program som tar inn to tall x og y og regner ut en verdi for uttrykket

$$xy + \frac{x + 2y}{3}$$

```
[ ]:
```

3.0.13 Oppgave 13

Noen amerikanske og engelske matlagingsprogrammer har irriterende høy bruk av temperaturskalaen fahrenheit. Formelen for å gjøre om fra fahrenheit til celcius er gitt ved

$$C = (F - 32) \cdot \frac{5}{9}$$

Skriv et program som leser inn grader fahrenheit fra brukeren, og skriver ut temperaturen i grader celcius.

3.0.14 Oppgave 14

I fysikken har du kanskje lært å bruke formelen

$$h(t) = h_0 + v_0 t - \frac{1}{2} g t^2$$

a) Se på koden under. Kan du fullføre den slik at det regnes ut en verdi for h ?

```
[ ]: g = 9.81 # tyngdens akselerasjon
     h_0 = 4.0 # starthøyde
     v_0 = 3.4 # vertikal start-fart

     h = h_0 + v_0 - 0.5*...
```

b) Hvilke verdier vil variere fra kast til kast? Les disse verdiene inn fra brukeren, og regn ut hvor høyden. Skriv dette ut i en lesbar forståelig melding til brukeren.

```
[ ]: 
```

c) Utvid programmet fra b) slik at det leser inn den totale farten og utgangsvinkelen på kastet. Bruk en parameterfremstilling til å regne ut posisjonen etter t sekunder.

Skriv denne posisjonen ut til brukeren

Du kan bli nødt til å importere funksjoner fra pakken pylab

```
[47]: from pylab import *
     # skriv resten av koden under her...
```

3.0.15 Oppgave 15

Under har Pål prøvd å skrive et program, men han får en feilmelding. ##### a) Hjelp Pål med å reparere programmet

```
[ ]: #gjør om mellom lysår og kilometer

     light_year = 9460730472580.8 # ett lysår målt i km
```

```
distance_light_years = input('Write the distance in light years: ')

distance_kilometers = distance_light_years*light_year

# print distansen i antall kilometer på standardform
print(f'distansen i km er: {distance_kilometers:g}')
```


I den siste linjen bruker vi en spesiell formatering av strenger. For å fortelle python at vi skal bruke en f-streng, setter vi bokstaven f foran strengen.

b) Fjern tegnene kolon og g (: og g) i siste linjen i programmet. Hva skjer?

Mer om f-strings Vi vil komme tilbake til bruk av f-strings senere. Hvis du er nysjerrig - kan du se en video av [Corey Shafer om f-strenger](#). Videoen antar at du har hatt litt mer om dictionaries og løkker, men dette er ikke essensielt for innholdet i videoen. Du kan enten se begynnelsen av filmen og spare resten til senere, eller hvis du klarer å holde deg rolig uten å forstå alle detaljene kan du kanskje ha utbytte av flere deler av videoen.

3.0.16 Oppgave 16

I denne oppgaven skal vi se litt på *objekter*. I python er alt objekter. Det betyr at alle variabler du lager, har et sett innebygde *metoder* som avhenger av hvilken datatype (f.eks int, float eller str) variabelen inneholder. For eksempel har en variabel som inneholder datatypen streng blant annet



attachment:image.png

metodene startswith, endswith, find og replace. Se figuren under.

a) Bruk et ipython-shell og lag gjør følgende tilordning

```
fornavn = 'doNAlDinh0'
```

Skriv `dir(fornavn)`, får du alle metodene til objektet fornavn

Skriv `help(fornavn.startswith)`. Du får ut en teknisk manual for hvordan metoden brukes.

Skriv `fornavn.startswith('doN')` og `fornavn.startswith('don')`. Hva betyr svarene?

b) Skriv et program som spør brukeren om en tekst. Programmet skal skrive teksten ut igjen med store bokstaver.

3.0.17 Oppgave 17

Strenger har også metoden `replace`. En av vennene dine har skrevet en lang tekst, der minst to ord er skrevet feil.

Mari gikk till skolen. Hun gikk klokka åtte. På skolen hadde hun matematikk, nrosk, engelsk og gym. På ettermiddagen gikk Mari hjem. Når hun gikk hjem, drodde hun

innom en butik. På butiken kjøpte hun en baget med laks og egg. Når Mari kom hjem, gjorde hun nrosk lekse før hun så hun på TV. Programmet handlet om en full man som gikk rundt i en stue, og sa "same procedure as last year!". I stua satt det en dame og skålte for Lord Pomeroy og andre venner. Hun sa alltid "same procedure as every year, James!" Mari kom alltid i jule stemmning når hun så dete programmet. På kvelden ringte bestemor till Mari. Di snakket lenge i telefonen. Det var lenge siden di hadde snakket med verandre. I sommerfeiren pleide Mari og besøke bestemoren og slekta. Da brukte di å kjøre bil till bestemor. Bestemor var også kommet i jule stemmning, så de hade mye å snakke om. Etterpå sa di hade bra till verandre..

Se etter noen ord som ofte er skrevet feil. Bruk python og metoden `str.replace` til å rydde litt i teksten slik at den ser noe bedre ut.

```
[22]: # du kan løse oppgaven her, kopier teksten og sett den inn i variabelen "tekst"

tekst = """Mari gikk till skolen. Hun gikk klokka åtte. På skolen hadde hun
    ↳mattematikk, nrosk, engelsk og gym. På ettermiddagen gikk Mari hjem. Når hun
    ↳gikk hjem, drodde hun innom en butik. På butiken kjøpte hun en
    baget med laks og egg. Når Mari kom hjem, gjorde hun nrosk lekse før hun så hun
    ↳på TV. Programmet handlet om en full man som gikk rundt i
    en stue, og sa "same procedure as last year!". I stua satt det en dame og skålte
    ↳for Lord Pomeroy og andre
    venner. Hun sa alltid "same procedure as every year, James!" Mari kom alltid i
    ↳jule stemmning når hun så dete
    programmet. På kvelden ringte bestemor till Mari. Di snakket lenge i telefonen.
    ↳Det var lenge siden di hadde snakket sammen.
    I sommerfeiren pleide Mari og besøke bestemoren og slekta. Da brukte di å kjøre
    ↳bil till bestemor. Bestemor var også kommet
    i jule stemmning, så de hade mye å snakke om. Etterpå sa di hade bra till
    ↳hverandre.
    """

# Fortsett med å erstatte flere ord under
tekst.replace('gikk', 'gikk')
```

```
[22]: 'Mari gikk till skolen. Hun gikk klokka åtte. På skolen hadde hun matematikk,
nrosk, engelsk og gym. På ettermiddagen gikkk Mari hjem. Når hun gikk hjem,
drodde hun innom en butik. På butiken kjøpte hun en\nnbaget med laks og egg. Når
Mari kom hjem, gjorde hun nrosk lekse før hun så hun på TV. Programmet handlet
om en full man som gikk rundt i\nnen stue, og sa "same procedure as last year!".
I stua satt det en dame og skålte for Lord Pomeroy og andre\nnvenner. Hun sa
alltid "same procedure as every year, James!" Mari kom alltid i jule stemmning
når hun så dete \nprogrammet. På kvelden ringte bestemor till Mari. Di snakket
lenge i telefonen. Det var lenge siden di hadde snakket sammen.\nI sommerfeiren
pleide Mari og besøke bestemoren og slekta. Da brukte di å kjøre bil till
bestemor. Bestemor var også kommet\nni jule stemmning, så de hade mye å snakke
om. Etterpå sa di hade bra till hverandre.\n'
```

- c) **Refleksjon:** Kan du komme på tilfeller der `str.replace` vil bytte ut ord som faktisk er riktige? Finnes det særtrekk for disse tilfellene? Kan du for noen slike ord beskrive en *systematisk* måte å unngå å bytte ut feil ord? Du skal ikke skrive kode, bare gi en systematisk fremgangsmåte.

3.0.18 Oppgave 18

Modifiser programmet til Pål i oppgave 15 slik at den leser inn informasjon fra kommandolinja

3.0.19 Oppgave 19

I denne oppgaven må du modifisere programmet `aplot_andergradsfunksjon.py`. Du kan selvfølgelig velge å heller modifisere programmet `plot_andregradsfunksjon.py`.

Vi skal starte med å stille opp en annen matematisk modell. Du har følgende scenarie:

Antall bakterier i en bakteriekultur er ved morgenen klokken 8.00 målt til 12 000 bakterier. Klokken 10.00 er antall bakterier målt til 13 000 bakterier.

- a) Still opp to ulike modeller $A(t)$ og $B(t)$ for antall bakterier i bakteriekulturen. For hver modell skal du tegne grafen til modellen.

Tenk gjennom følgende for hver av modellene:

- Hvilken informasjon trenger du fra brukeren for å tegne grafen?
- Hva må endres i programmet?

- c) Velg modellen fra oppgave b) du syns er mest realistisk.

Gjør de nødvendige endringene i programmet slik at du kan plote grafen til funksjonen.

Lagre endringene i et nytt program `plot_bakteriepopulasjon.py`

```
[15]: %%writefile plot_bakteriepopulasjon.py
      # Du kan skrive koden din her
```

Writing `plot_bakteriepopulasjon.py`

3.0.20 Oppgave 20

Les inn noen data for loddrett kast i programmet over og plot grafen slik at du får høyde over bakken på y-aksen som funksjon av tiden etter kastet i sekunder på x-aksen.

```
[ ]:
```