# 13 Krasjkurs i PYTHON



Jeg har aldri skrevet et krasjkurs i programmering før. Python kan jeg ikke en gang selv, og jeg er en elendig programmerer. Det tar meg år og dag å finne ut av de enkleste ting.

## Komme igang

1 Installer Python: `https://innsida.ntnu.no/wiki/-/wiki/English/Installing+Python`. Mac-brukere kan bare bruke den versjonen som maskinen kom med. Jeg bruker 2.7 eller noe, som er antikvarisk. Det går helt fint.

2 Finn en måte å lage .py-filer, og lag en fil som heter 'test.py'. Jeg bruker Xcode. Det er nok litt overkill, men de vanlige hurtigtastene for mac fungerer der, og det er fine farger.

3 Åpne en terminal. Du kjører .py-filer ved å skrive

```
python test.py
```

og trykke på ENTER.

Bruksanvisningen for dette krasjkurset er veldig enkel: Alt som står i en slik ramme:

```
print 2.0 / 3.0
```

er kode du skal kjøre. Du limer teksten inn i filen din, hva nå enn den heter, og kjører den i terminalen. NB: husk å stå i rett katalog (den katalogen der test.py er lagret), og husk å lagre filen før du kjører.

Du kan også kjøre kode direkte i terminalen, og bruke Python mer som en vanlig kalkulator. I så fall skriver du

```
python
```

i terminalvinduet ditt. Da får du muligheten.

## Greier du må skrive først

Python er litt vittig, for du må fortelle python hva du skal drive på med, ellers skjønner python ingenting. Vanligvis skriver man øverst i filen

```
import numpy as np
```

Sjekk gjerne numpy.org. Nå har man tilgang på de vanligste matematiske regneoperasjonene. Men ikke for eksempel plotting, da må du skrive noe mer greier. Vi kommer tilbake til det.

## Grunnleggende regneoperasjoner

Python kan så vidt jeg vet brukes til alt, og kan du python, er du hot på arbeidsmarkedet. (Håper jeg aldri må ha noe med det å gjøre.) Vi begynner med de vanlige regnereglene.
Addisjon:

```
print 2 + 3
```

Hvis skriver Pythonkoden direkte i terminalen, trenger du bare skrive

```
2 + 3
```

men heretter skal jeg anta at du tester disse linjene ved å lime dem inn i test.py
Subtraksjon:

```
print 2 - 3
```

Multiplikasjon:

```
print 2 * 3
```

En viktig ting: Python kan fort finne på å ikke skjønne av seg selv hva slags regning du er ute etter. Divisjon:

```
print 2 / 3
```

Med å tvinge inn desimaler, får python inn med teskje hva du mener

```
print 2.0 / 3.0
```

Dette er selvfølgelig ikke nødvendig om du bare skal regne med heltall. Potenser går fint

```
print 2**3
```

Kvadratrøtter tar du med potens

```
print 2**.5
```

Husk å alltid bruke gangetegn!

```
print 2(2 + 3)
```

```
print 2*(2 + 3)
```

## Variable

Vi lagrer variable slik:

```
a=2*(2 + 3);
b=7;
```

Skriver du

```
print a
```

får du en utskrift av verdien lagret i $a$. Du kan herje med variable akkurat som med tall

```
print a+b
```

Hvis du er lei av verdien $a = 12$, kan du overskrive den

```
a=2;
```

## Vektorer

Vi begynner med å lage en vektor

```
x=[2,5,7]
print x
```

Denne kommandoen lager noe som Python kaller et 'array'. Python er i utgangspunktet ikke spesiallaget for matematikk, slik som Matlab. Det er derfor ikke en selvfølge at Python behandler vektorer og matriser som vi gjorde i M3. For eksempel er det ikke en selvfølge at Python skiller mellom rekke- og søylevektorer.

```
print transpose(x)
```

Vi kommer nok ikke til å trenge å skille mellom rekke- og søylevektorer i M4, men om du skulle ha bruk for det, kan du bruke

```
x=matrix([2,5,7])
print x
print transpose(x)
```

Dette er en typisk ting som gjør Python litt mer komplisert å lære, om du skal programmere matematikk. Du må stadig fortelle Python at du driver på med matematikk, og ikke programmerer internettsider eller dataspill. Eller følger Python de vanlige regnereglene for vektorer, men du må vite hva kommandoene heter:

```
x=[2,5,7]
y=[3,6,9]
print np.add(x,y)
```

De vanligste matematiske operasjonene du har lært, er implementert i Numpy. Dette biblioteket importerte vi til å begynne med som 'np', og lurer du på hvordan en funksjon fungerer, googler du 'numpy *operasjonen du lurer på*'. Det finnes en funksjon som heter np.multiply. Den ganger sammen vektorer komponentvis. Hvis den ene vektoren er en skalar, skjønner np.multiply av seg selv at det produkt mellom skalar og vektor det er snakk om

```
print np.multiply(2,x)
print np.multiply(x,y)
```

## Plotting

Nå kan vi nok til å plotte funksjoner. Her er en ting som er viktig å forstå: Python plotter vektorer mot hverandre, så man lager først en vektor med verdier på $x$-aksen, og så en vektor med funksjonsverdier. Pytthon tegner en rett strek mellom punktene, men hvis det er tett nok mellom verdiene på $x$-aksen vil kurven se glatt ut.

Det første du må gjøre er å importere matplotlib

```
import numpy as np
import matplotlib.pyplot as plt
```

Sjekk for øvrig matplotlib.org. La oss plotte funksjonen $f(x) = x \sin x$ på intervallet $[0, 2\pi]$. Først må vi lage en vektor med punkter på $x$-aksen

```
np.linspace(0, 2*np.pi, num=1000)
```

Denne kommandoen lager et gitter med 1000 ekvidistante verdier fra 0 til $2\pi$. Merk at man må kalle np.pi for å få ut pi. Nå lager vi en vektor med funksjonsverdier

```
y=np.multiply(x,sin(x))
```

Vi skriver

```
plt.plot(x,y)
```

for å lage figuren, og

```
plt.show()
```

for å vise den. Hvis du vil lagre den, skriver du

```
fig.savefig("en_flott_figur.png")
```

eller et annet filnavn. Hvis du har lyst til å plotte to funksjoner i samme figur, for eksempel $f(x) = x \sin x$ og $g(x) = x^2 \sin x$, skriver du

```
np.linspace(0, 2*np.pi, num=1000)
y=np.multiply(x,sin(x))
z=np.multiply(x**2,sin(x))
plt.plot(x,y)
plt.plot(x,z)
plt.show()
```

## For og if

En for løkke er en måte å få datamaskinen til å gjøre samme ting om igjen, men med forskjellige parameter

```
for i in range(5):
   disp(i)
```

Python bruker indentering for å skjønne hva som er inne i for-løkken, og hva som er utenfor. Bruk tabtasten for korrekt indentering, hvis ikke teksteditoren din får til sånt av seg selv.

If kan brukes til å gjøre unntak

```
for i in range(5):
   if i == 3:
      print i
```

## Matriser

## 3D-plot

## Animasjon