Python – Installering og et par enkle anvendelser

1. Gå til `https://docs.anaconda.com/anaconda/install/#`.

2. Last ned Anaconda for ditt operativsystem. Følg instruksjonene og installer.

3. Start Anaconda Navigator og velg Spyder.

Du kan nå begynne å skrive inn python-kommandoer på kommandolinja i vinduet nederst til høyre, f.eks:

```
In [1]: 2+2
Out[1]: 4
In [2]: 7**(0.5)
Out[2]: 2.6457513110645907
```

Dvs, python kan brukes som en enkel kalkulator. Som regel ønsker vi å skrive et program og lagre dette i ei fil.

4. Vinduet til venstre er en editor der vi kan skrive inn en serie med python-kommandoer, linje for linje. Hos meg (med Mac) heter den nesten tomme fila allerede temp.py, og den har noe kommentargreier i de seks første linjene som vi bare kan la stå der.

Fra linje 7 kan vi skrive, f.eks:

```
a = 6
b = 7
answer = a*b
print (answer)
```

Velg *File - Save As* (eller diskett-ikonet) og lagre programmet (dvs disse fire kommandoene) med et filnavn (som du velger selv) og *extension* .py (som er nødvendig for at python skal skjønne at det er et python-program), f.eks python1.py. For å holde litt orden lager jeg mappa `FY6019_2018` på mitt hjemmeområde. Nå kan programmet kjøres (dvs de fire kommandoene utføres, en etter en) ved å trykke på den grønne pila i menyen I vinduet nederst får jeg nå

```
In [3]: runfile('/Users/stovneng/FY6019_2018/python1.py', wdir='/Users/stovneng/FY6019_2018')
42
```

Med andre ord, python tar inn kommandoen *kjør python1.py*. Der settes variablene a og b lik hhv 6 og 7, variabelen answer settes lik produktet mellom a og b, dvs 42, og til slutt skrives verdien av answer ut til *standard output* (som her er vinduet nederst) med kommandoen print.

5. Du ser allerede nå at python forstår en god del matematikk "uten videre", f.eks at "+" betyr addisjon og at "**" betyr "opphøyd i". Hva med litt mer avanserte matematiske funksjoner som f.eks sinus og cosinus? La oss prøve. Lag ei ny fil med ikonet helt til venstre i menyen eller med File - New file.... Den heter hos meg i utgangspunktet *untitled0.py*, så lagre den like godt som *python2.py* med det samme. Skriv inn de tre linjene

```
a=1.5708
b=sin(a)
print (b)
```

og lagre. Med forventning om at python skal returnere et tall i nærheten av 1 trykker vi på den grønne "run-knappen". Skuffelsen er kanskje stor når python returnerer diverse linjer nede til høyre, blant annet

```
NameError: name 'sin' is not defined
```

Python skjønner med andre ord ikke uten videre hva kommandoen "sin" betyr. Vi må importere *biblioteket* numpy ved å inkludere kommandoen

```
import numpy as np
```

som første linje i programmet. Dessuten må vi fortelle python at funksjonen sin skal hentes i numpy-biblioteket, som vi nå har gitt kortnavnet np:

```
b=np.sin(a)
```

Hele programmet blir seende slik ut:

```
import numpy as np
a=1.5708
b=np.sin(a)
print (b)
```

Nå går det mye bedre! *Grønn run* gir oss i nederste vindu:

```
runfile ('/Users/stovneng/FY6019_2018/python2.py')
0.999999999993
```

Biblioteket numpy (*numerical python*) inneholder de fleste matematiske funksjoner som du kan tenke deg å få bruk for. I tillegg legger det til rette for å utføre operasjoner på hele tabeller (og matriser) med tall "i en smekk". Vi skal se at det er svært nyttig. (Det finnes også et bibliotek som heter math, som vi alternativt kunne ha importert i stedet for numpy, men da hadde vi ikke fått med oss tabell-funksjonaliteten på kjøpet, slik vi gjør med numpy.)

6. Vi har rett som det er behov for å framstille våre resultater ved å plotte en eller flere funksjoner (grafer). Da må vi først importere modulen pyplot fra biblioteket matplotlib. Vi importerer også numpy, slik at vi kan lage tabeller med tall:

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

La oss ta et konkret eksempel: Vi ønsker å plotte funksjonen $\cos(x)$ på intervallet $0 < x < 2\pi$. Vi lager da først en tabell x med et antall verdier, f.eks 100, mellom 0 og 2π:

```
x = np.linspace(0,2*np.pi,100)
```

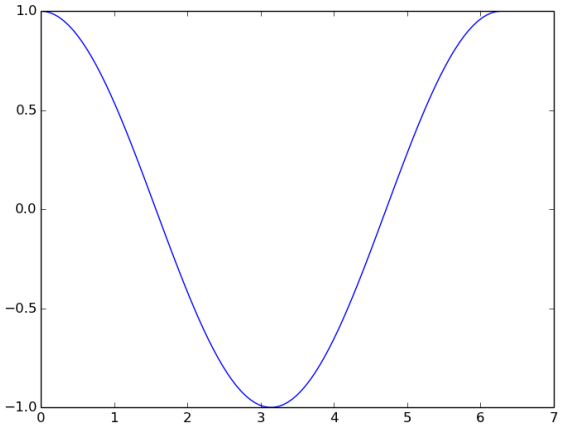
Med andre ord, `x = np.linspace(start,slutt,antall)` medfører at `x` blir en tabell med `antall` elementer, slik at 1. element får verdien `start`, siste element får verdien `slutt`, og resten av elementene er jevnt fordelt mellom `start` og `slutt`. Med kommandoen

```
y = np.cos(x)
```

blir `y` nå automatisk en tabell med like mange elementer som `x`, dvs 100, og slik at 1. element får verdien $\cos(0) = 1$ og siste element får verdien $\cos(2\pi) = 1$. Nå kan vi plotte $y(x)$ med kommandoene

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

der den siste linja er nødvendig for å få opp figuren på skjermen. Skriv disse linjene og lagre programmet i fila *python3.py*. Kjøring av programmet gir figur 1.

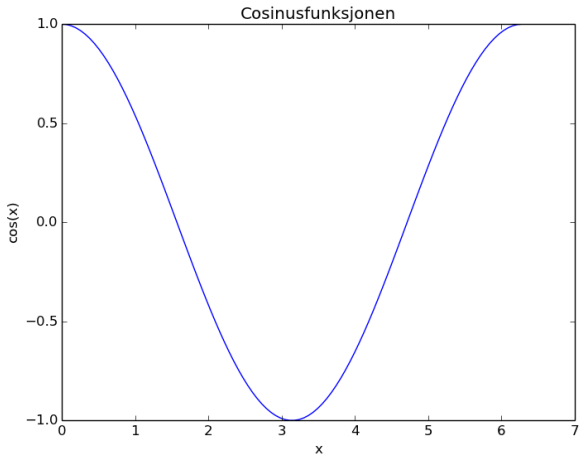


Figur 1: Plotting av cos(x) med kommandoene plot og show fra modulen pyplot i matplotlib.

Vi kan pynte på figuren (der de ulike kommandoene bør tale for seg):

```
plt.title('Cosinusfunksjonen')
plt.xlabel('x')
plt.ylabel('cos(x)')
```

Figuren blir da som i figur 2.

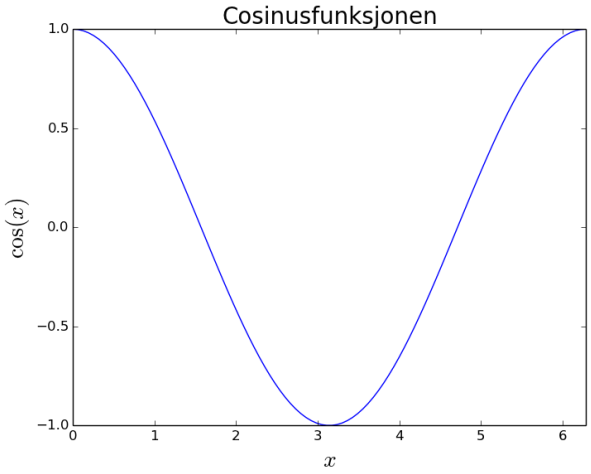


Figur 2: Plotting av cos(x), med tittel og akseangivelser.

Enda penere ser det ut hvis vi avgrenser x-aksen til 2π, øker fontstørrelsen til 20, og i tillegg bruker dollartegn og \ (LaTeX-stil) for å få korrekte fonter i matematiske uttrykk:

```
plt.xlabel('$x$',fontsize=20)
plt.ylabel('$\cos(x)$',fontsize=20)
plt.xlim(0,2*np.pi)
```

Figuren blir da som i figur 3.



Figur 3: Ganske pent plott av $\cos(x)$.

Oppgave (frivillig!)

- Lag en figur som plotter $y(r) = 4r^2\, e^{-2r}$ fra $r = 0$ til $r = 6$. Figuren skal ha tittelen "Grunntilstanden 1s i hydrogen", teksten "Avstand fra kjernen" langs horisontal akse, og teksten "Sannsynlighetstetthet" langs vertikal akse. Tips: Eksponentialfunksjonen heter exp i python (som i de fleste programmeringsspråk), dvs `np.exp(...)`.