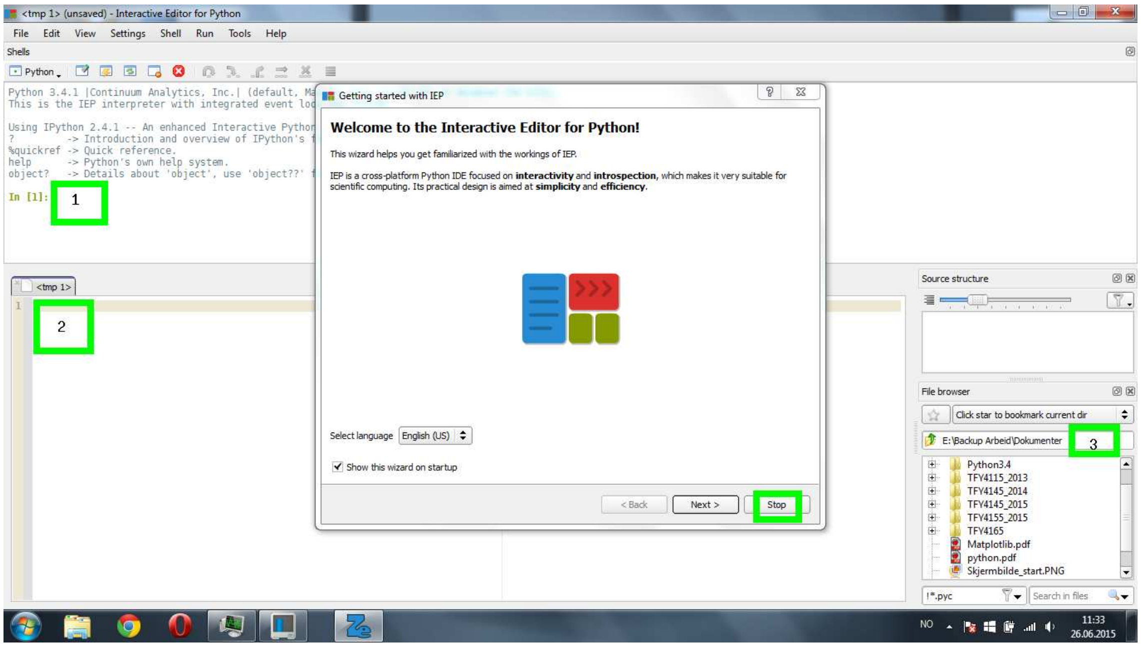
Python – Installering og et par enkle anvendelser

1. Gå til www.pyzo.org/downloads.html

2. Last ned, og installer pyzo for ditt operativsystem (Windows, Linux, OSX):
På Windows kan du laste ned og kjøre installeren, eller laste ned og pakke ut .zip filen.
På Mac monterer du .dmg filen, og kopierer innholdet til user-mappen din.
På Linux pakker du ut innholdet i .tar.gz filen.

3. Start programmet:
Hvis du brukte installeren for Windows, starter du programmet ved å dobbeltklikke på pyzo-ikonet på skjermen.
Installerte du ved å kopiere en mappe, starter du programmet ved å dobbeltklikke på filen pyzo, i mappen du kopierte.
Når programmet har startet, burde det se slik ut



Figur 1: Windows skjermbilde etter oppstart av pyzo.

Skjermen har tre hovedområder (markert med grønt rektangel):
1. Kommandovinduet.
2. Editoren.
3. Filutforskeren.

Du kan nå begynne å skrive inn python-kommandoer på kommandolinja i kommandovinduet, 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. Nå kan du skrive inn python-kommandoer i vinduet øverst, f.eks:

```
1 a = 6
2 b = 7
3 answer = a*b
4 print(answer)
```

Velg *File - Save As* 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. Nå kan programmet kjøres (dvs de fire kommandoene utføres, en etter en) ved å velge (*Run - Execute file*). I kommandovinduet får du nå

```
 In [3]: (executing lines 1 to 4 of ''python1.py'')
 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 *File - New*. Den heter i utgangspunktet <*tmp 2*>, så lagre den like godt som *python2.py* med det samme. Skriv inn de tre linjene

```
1 a=1.5708
2 b=sin(a)
3 print(b)
```

og lagre. Med forventning om at python skal returnere et tall i nærheten av 1 velger vi *Run - Execute file*. Skuffelsen er kanskje stor når python returnerer

```
Traceback (most recent call last):
  File ''/Documents/Arbeid/Python notat/python2.py'', line 2, in <module>
      b=np.sin(a)
      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

```
1 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:

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

Hele programmet blir seende slik ut:

```
1 import numpy as np
2 a=1.57
3 b=np.sin(a)
4 print(b)
```

Nå går det mye bedre! *Run - Execute file* gir oss i kommandovinduet:

```
In[5]: (executing lines 1 to 4 of ''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:

```
1 import matplotlib.pyplot as plt
2 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:

```
3 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

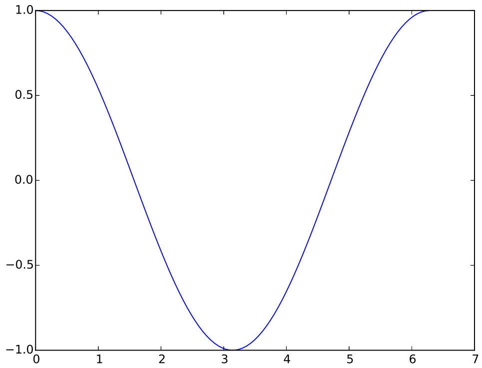
```
4 y = np.cos(x)
```

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

```
5 plt.figure()
6 plt.plot(x,y)
7 plt.show()
```

der nr 7 er nødvendig for å få opp figuren på skjermen. Skriv disse 7 linjene og lagre programmet i fila *python3.py*. Kjøring av programmet gir figur 2.
Vi kan pynte på figuren (der de ulike kommandoene bør tale for seg):

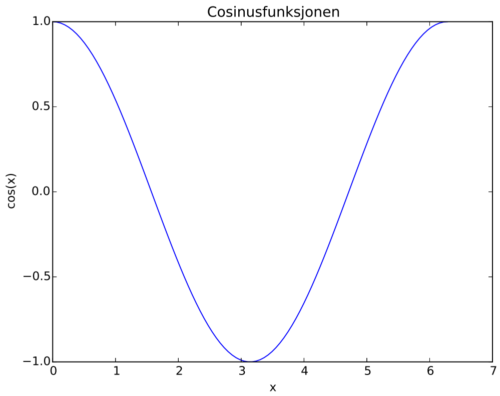
```
7 plt.title('Cosinusfunksjonen')
```



Figur 2: Plotting av $\cos(x)$ med kommandoene plot og show fra modulen pyplot i matplotlib.

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

Figuren blir da som i figur 3.



Figur 3: 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:

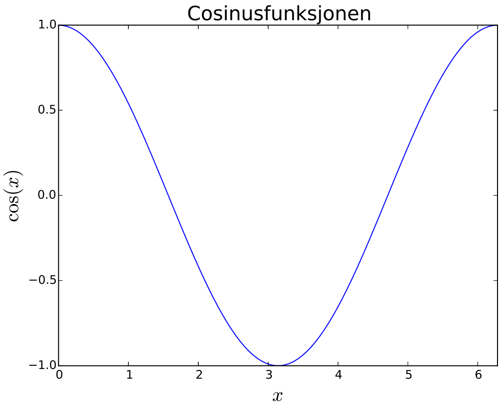
```
plt.xlabel(r'$x$',fontsize=20)
plt.ylabel(r'$\cos(x)$',fontsize=20)
plt.xlim(0,2*np.pi)
```

Merk r'en foran strenger med LaTeX-kommandoer. Hele programmet blir:

```
import matplotlib.pyplot as plt
import numpy as np
x=np.linspace(0,2*np.pi,100)
y=np.cos(x)
plt.figure()
plt.plot(x,y)
```

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

Figuren blir da som i figur 4.



Figur 4: Ganske pent plott av $\cos(x)$.

Oppgave

- 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(...)`.

Python – Tips og triks

- Tabellindekser starter på 0, og hvert element aksesseres med []. Alle elementene i tabellen aksesseres med [:]. Negative indekser teller fra siste element, som har index -1.
  Eks.: Alle radene, alle unntatt 1. og siste kolonne i en 3 x 5 tabell med tilfeldige tall:
  ```
  from numpy.random import rand
  tabell=rand(3,5)
  utdrag=tabell[:,1:-1]
  ```
- Flere plott i samme figur:
  Når vi plotter flere grafer i samme figur, lager vi merkelapper på hver graf, og viser en oversikt med *legend()* kommandoen:
  ```
  ...
  y1=np.cos(x)
  y2=np.cos(2*x)
  plt.plot(x,y1,label=r'$\cos(x)$')
  plt.plot(x,y2,label=r'$\cos(2x)$')
  plt.legend(loc='best')
  ...
  ```
- Strenger med LaTeX-kommandoer og vanlig tekst settes sammen med '+':
  ```
  ...
  plt.title('Plott av ' + r'$\cos(x)$' + ' og ' + r'$\cos(2x)$')
  ...
  ```
- Variable i strenger:
  Heltall, flyttall og flyttall med e-notasjon settes inn for hhv. %d, %f og %e med en % etter strengen. For %f og %e kan vi sette antall desimaler til n med %.nf og %.ne.
  Eks π med 15 desimaler:
  ```
  streng=r'Pi med %d desimaler: %.15f' % (15,np.pi)
  ```
  Eks e med 25 desimaler:
  ```
  streng2=r'e med %d desimaler: %.25f' % (25.np.e)
  ```
- Inndele x-aksen i radianer fra 0 til 2π:
  ```
  ...
  plt.xticks([0,np.pi/2,np.pi,3*np.pi/2,2*np.pi],
   [0,r'$\frac{\pi}{2}$',r'$\pi$',r'$\frac{3\pi}{2}$',r'$2\pi$'])
  ...
  ```
- Funksjoner:
  Skal vi kjøre de samme kommandoene flere ganger i et program, er det lurt, og ryddig, å samle kommandoene i en funksjon. Kommandoene skrives på innrykkede linjer etter en linje med *def*, som er starten på funksjonen. Vi kan ta inn argumenter med en parentes i første linje, og returnere variable med *return* som siste kommando:
  ```
  def funksjonsnavn(a,b):
      c=a+b
      return c
  ```
  Funksjonen kan da kalles direkte:
  ```
  c=funksjonsnavn(a,b)
  ```
- λ-funksjoner:
  For kommandoer som får plass på en linje (såkalte 'one-liners') kan vi enkelt lage en

funksjon med kodeord *lambda* og et kolontegn (:)
```
f=lambda a,b: a+b
```
Funksjonen kan kalles direkte slik:
```
c=f(a,b)
```

- Vi kan importere kun de funksjonene vi trenger, i stedet for hele biblioteket:
  ```
  from numpy import cos,sin
  ```
  De importerte funksjonene kan da kalles direkte:
  ```
  a=sin(x)
  ```

- Importere egne funksjoner:
  Funksjonen *funksjonsnavn* i filen *filavn.py* kan, hvis den er i samme mappe, importeres og brukes enten med
  ```
  from filnavn import funksjonsnavn
  ...
  funksjonsnavn()
  ```
  eller med
  ```
  import filnavn
  ...
  filnavn.funksjonsnavn()
  ```

- Lese data fra txt-fil, hvor de n første linjene ikke inneholder data:
  ```
  from numpy  import loadtxt
  data=loadtxt('filnavn.txt',skiprows=n)
  ```

- Standardoppsett:
  En ryddig og oversiktlig måte å skrive program på, er
  ```
  import ...
  ...
  def prog():
  ...
  if __name__=='__main__':
      prog()
      ...
  ```
  Linjen `if __name__=='__main__':` sørger for at kommandoene på de påfølgende innrykkede linjer kun blir utført når vi kjører denne filen. Dvs. at de ikke blir kjørt når vi importerer denne filen fra et annet program.

- Execute file / Run file as script:
  Når vi kjører programmet med *Run - Run file as script*, startes python-tolkeren på nytt, dvs. alle variable blir slettet, figur-vinduer lukket osv. I tillegg endres 'arbeidsmappen' (working directory) til den mappen programmet er lagret i. Dette gjør det lettere å finne igjen andre filer fra programmet.
  Når vi kjører programmet med *Run - Execute file*, kjører vi koden som om vi hadde skrevet den direkte, linje for linje, i kommandovinduet. Arbeidsmappen endres ikke.

- Dokumentasjon:
  For informasjon om, og dokumentasjon av numpy, matplotlib etc. er 'http://scipy.org/docs.html' et bra sted å begynne.
  Generell dokumentasjon for python finnes på 'https://docs.python.org/3/'