In [1]:
```
# Numpy har tre datatyper som representerer tall: Heltall, desimaltall og komplekse tall.
```

In [2]:
```
5.
```

Out[2]:
```
5.0
```

In [3]:
```
# Det komplekse tallet 2+3i skrives i Python slik:
```

In [4]:
```
2+3j
```

Out[4]:
```
(2+3j)
```

In [5]:
```
# Skriver man j alene, oppfatter Python det som en variabel. Skriver man derimot 1j, får man den imaginære enhet:
```

In [6]:
```
1j**2
```

Out[6]:
```
(-1+0j)
```

In [7]:
```
j=2
```

In [8]:
```
j**2
```

Out[8]:
```
4
```

In [9]:
```
import numpy as np
```

In [10]:
```
# Vi definerer en vektor:
```

In [11]:
```
v=np.array([1,2,3])
v
```

Out[11]:
```
array([1, 2, 3])
```

In [12]:
```
2*v
```

Out[12]:
```
array([2, 4, 6])
```

In [13]:
```
v-3*v
```

Out[13]:
```
array([-2, -4, -6])
```

In [14]:
```
w=np.array([3,-1,0])
```

In [15]:
```
# v*w gir ikke skalarprodukt, men komponentvis produkt:
```

In [16]:
```
v*w
```

Out[16]:
```
array([ 3, -2,  0])
```

In [17]:
```
# Skalarprodukt:
```

In [18]:
```
np.dot(v,w)
```

Out[18]:
```
1
```

In [19]:
```
# Komponentene til en vektor indekseres fra 0 og oppover:
```

In [20]:
```
v[0]
```

Out[20]:
```
1
```

In [21]:
```
# Vi definerer en matrise:
```

In [22]:
```
A=np.array([[1,2,3],[4,5,6],[7,8,9]])
A
```

Out[22]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

In [23]:
```
A[0,0]
```

Out[23]:
```
1
```

In [24]:
```
# Rad 0:
```

In [25]:
```
A[0,:]
```

Out[25]:
```
array([1, 2, 3])
```

In [26]:
```
# Søyle 1:
```

In [27]:
```
A[:,1]
```

Out[27]:
```
array([2, 5, 8])
```

In [28]:
```
# Den transponerte:
```

In [29]:
```
B=A.T
B
```

Out[29]:
```
array([[1, 4, 7],
       [2, 5, 8],
       [3, 6, 9]])
```

In [30]:
```
A+B
```

Out[30]:
```
array([[ 2,  6, 10],
       [ 6, 10, 14],
       [10, 14, 18]])
```

In [31]:
```
# Skalarmultiplikasjon:
```

In [32]:
```
2*A
```

Out[32]:
```
array([[ 2,  4,  6],
       [ 8, 10, 12],
       [14, 16, 18]])
```

In [33]:
```
A*2
```

Out[33]:
```
array([[ 2,  4,  6],
       [ 8, 10, 12],
       [14, 16, 18]])
```

In [34]:
```
# Komponentvis produkt:
```

In [35]:
```
A*B
```

Out[35]:
```
array([[ 1,  8, 21],
       [ 8, 25, 48],
       [21, 48, 81]])
```

In [36]:
```
# Matriseprodukt:
```

In [37]:
```
np.dot(A,B)
```

Out[37]:
```
array([[ 14,  32,  50],
       [ 32,  77, 122],
       [ 50, 122, 194]])
```

In [38]:
```
# Matrise-vektor-produkt:
```

In [39]:
```
np.dot(A,v)
```

Out[39]:
```
array([14, 32, 50])
```

In [40]:
```
# Her har Python tolket v som en søylevektor
```

In [41]:
```
np.dot(v,A)
```

Out[41]:
```
array([30, 36, 42])
```

In [42]:
```
# Her har Python tolket v som en radvektor
```

In [43]:
```
# Identitetsmatrisen:
```

In [44]:
```
np.eye(4)
```

Out[44]:
```
array([[1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.]])
```

In [45]:
```
# Determinanten:
```

In [46]:
```
np.linalg.det(A)
```

Out[46]:
```
0.0
```

In [47]:
```
# Vi sjekker nå en utregning fra forelesningen:
```

In [48]:
```
C=np.array([[0,1,-2],[3,2,0],[4,1,-1]])
C
```

Out[48]:
```
array([[ 0,  1, -2],
       [ 3,  2,  0],
       [ 4,  1, -1]])
```

In [49]:
```
# 13 x den inverse matrisen til C:
```

In [50]:
```
13*np.linalg.inv(C)
```

Out[50]:
```
array([[-2., -1.,  4.],
       [ 3.,  8., -6.],
       [-5.,  4., -3.]])
```

In [51]:
```
# Den syvende potens av C:
```

In [52]:
```
np.linalg.matrix_power(C,7)
```

Out[52]:
```
array([[466,  23, -92],
       [-69, 236, 138],
       [115, -92, 489]])
```

In [53]:
```
A
```

Out[53]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

In [54]:
```
# Vi regner ut egenverdiene og egenvektorene til A i en kommando:
```

In [55]:
```
egenverdier, egenvektorer = np.linalg.eig(A)
```

In [56]:
```
egenverdier
```

Out[56]:
```
array([ 1.61168440e+01, -1.11684397e+00, -1.30367773e-15])
```

In [57]:
```
# Søylene i matrisen "egenvektor" er egenvektorer til A:
```

In [58]:
```
egenvektorer
```

Out[58]:
```
array([[-0.23197069, -0.78583024,  0.40824829],
       [-0.52532209, -0.08675134, -0.81649658],
       [-0.8186735 ,  0.61232756,  0.40824829]])
```

In [59]:
```
# Den j-te søylen er en egenvektor som svarer til den j-te egenverdien. Vi sjekker dette for den 0-te søylen:
```

In [60]:
```
la=egenverdier[0]
la
```

Out[60]:
```
16.116843969807043
```

In [61]:
```
v=egenvektorer[:,0]
v
```

Out[61]:
```
array([-0.23197069, -0.52532209, -0.8186735 ])
```

In [62]:
```
np.dot(A,v) - la*v
```

Out[62]:
```
array([ 5.32907052e-15, -1.77635684e-15, -1.77635684e-15])
```

In [63]:
```
# I alminnelighet kan både egenverdier og egenvektorer være komplekse. Her er et eksempel hvor det er tilfelle:
```

In [64]:
```
D=np.array([[0,1],[-1,0]])
D
```

Out[64]:
```
array([[ 0,  1],
       [-1,  0]])
```

In [65]:
```
egenverdier, egenvektorer = np.linalg.eig(D)
```

In [66]:
```
egenverdier
```

Out[66]:
```
array([0.+1.j, 0.-1.j])
```

In [67]:
```
egenvektorer
```

Out[67]:
```
array([[0.70710678+0.j        , 0.70710678-0.j        ],
       [0.        +0.70710678j, 0.        -0.70710678j]])
```

In [68]:
```
# Numpy har ellers funksjoner som sqrt (kvadratrot), cos, sin, exp, log:
```

In [69]:
```
x=np.sqrt(2)
x
```

Out[69]:
```
1.4142135623730951
```

In [70]:
```
x*x
```

Out[70]:
```
2.0000000000000004
```

In [71]:
```
np.pi
```

Out[71]:
```
3.141592653589793
```

In [72]:
```
np.cos(np.pi)
```

Out[72]:
```
-1.0
```

In [73]:
```
# Vil man slippe å skrive np hele tiden, kan man importere alt fra numpy:
```

In [74]:
```
from numpy import *
```

In [75]:
```
pi
```

Out[75]:
```
3.141592653589793
```

In [76]:
```
sin(pi)
```

Out[76]:
```
1.2246467991473532e-16
```

In [77]:
```
linalg.det(A)
```

Out[77]:
```
0.0
```

In [ ]: