Import Numpy

In [1]:
```
import numpy as np
```

Numpy arrays

In [2]:
```
a = np.array( [1, 2, 3, 4, 5], float)
a
```

Out[2]:
```
array([ 1.,  2.,  3.,  4.,  5.])
```

In [6]:
```
b = np.array( [9,8,9], int)
print(b.dtype)
```

```
int64
```

In [7]:
```
a = np.array([1,2,3])
b = np.array([4,5,6])
a+b
```

Out[7]:
```
array([5, 7, 9])
```

Indexing

In [8]:
```python
# np.arange is the numpy equivalnt to range
a = np.arange(12)

# access to single values
a[1]

# slicing
a[2:5]
a[2:]
a[:3]

# access from the back
a[-1]

# acessing every n-th element
a[2:10:2]
a[::3]


# reassignment
a[0] = 4
a
```

Out[8]:
```
array([ 4,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])
```

In [10]:
```python
# iteration
for elem in a:
    print(elem)
```

```
4
1
2
3
4
5
6
7
8
9
10
11
```

comparison

In [11]:
```python
a==2
a>5

# index using a comparison
a[a > 5]
```

Out[11]:
```
array([ 6,  7,  8,  9, 10, 11])
```

Multi-dimensional arrays

In [12]:
```
A = np.array( [[1,2,3], [4,5,6], [7,8,9]])
A
```

Out[12]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

In [14]:
```
print(A[0,0])
print(A[2,:])
print(A[:,0])
print(A[::2,1])
```

```
1
[7 8 9]
[1 4 7]
[2 8]
```

Array dimensions

In [15]:
```
print(len(A))     # length of 1st dimension
print(A.shape)    # shape of the arrays
```

```
3
(3, 3)
```

Check occurrence

In [16]:
```
print(2 in A)
print(13 in A)
```

```
True
False
```

Changing the shape of an array

In [17]:
```
c = np.array([1,2,3,4,5,6])
print(c)
c.reshape( (3,2) ) # reshape takes a dimension tuple as input (#row
s,#cols)
```

```
[1 2 3 4 5 6]
```

Out[17]:
```
array([[1, 2],
       [3, 4],
       [5, 6]])
```

Attention: deep-copy vs. shallow-copy

In [18]:
```
a = np.array([1,2,3])
b = a
c = a.copy()
a[0]=6
print(a)
print(b)
print(c)
```

```
[6 2 3]
[6 2 3]
[1 2 3]
```

Further array functions

In [19]:
```
# convert to python list
a.tolist()

# fill up with new value
a.fill(12)

print(A)
print(A.transpose()) # transpose matrix
print(A.T)           # same as A.transpose()
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
[[1 4 7]
 [2 5 8]
 [3 6 9]]
[[1 4 7]
 [2 5 8]
 [3 6 9]]
```

In [20]:
```
# concatenate arrays
B = A.transpose()

np.concatenate( (A,B) )
np.concatenate( (A,B), axis=1 )
```

Out[20]:
```
array([[1, 2, 3, 1, 4, 7],
       [4, 5, 6, 2, 5, 8],
       [7, 8, 9, 3, 6, 9]])
```

Convenient functions to construct arrays

In [21]:
```
# equally spaced values within an interval: arange( start, stop, st
ep )
np.arange(0, 5, 0.1)
```

Out[21]:
```
array([ 0. ,  0.1,  0.2,  0.3,  0.4,  0.5,  0.6,  0.7,  0.8,  0.9,
1. ,
        1.1,  1.2,  1.3,  1.4,  1.5,  1.6,  1.7,  1.8,  1.9,  2. ,
2.1,
        2.2,  2.3,  2.4,  2.5,  2.6,  2.7,  2.8,  2.9,  3. ,  3.1,
3.2,
        3.3,  3.4,  3.5,  3.6,  3.7,  3.8,  3.9,  4. ,  4.1,  4.2,
4.3,
        4.4,  4.5,  4.6,  4.7,  4.8,  4.9])
```

In [23]:
```
# generate 1 or 0-arrays
print(np.ones( (2,3) ))
print(np.zeros( (3,4) ))
```

```
[[ 1.  1.  1.]
 [ 1.  1.  1.]]
[[ 0.  0.  0.  0.]
 [ 0.  0.  0.  0.]
 [ 0.  0.  0.  0.]]
```

In [24]:
```
# construct an array with a similar shape to another one
np.zeros_like(A)
```

Out[24]:
```
array([[0, 0, 0],
       [0, 0, 0],
       [0, 0, 0]])
```

In [25]:
```
# identity matrix
Id = np.identity(4)
Id
```

Out[25]:
```
array([[ 1.,  0.,  0.,  0.],
       [ 0.,  1.,  0.,  0.],
       [ 0.,  0.,  1.,  0.],
       [ 0.,  0.,  0.,  1.]])
```

numpy mathematics

All operations on arrays are elementwise per default

In [26]:
```
A = np.array([[1,2],[3,4]])
B = np.array([[0,1],[1,0]])
A + B
A - B
A * B
B / A
A ** B
```

Out[26]:
```
array([[1, 2],
       [3, 1]])
```

Matrix multiplication

In [27]:
```
A = np.array( [[0,1], [1,0]] )
v = np.array( [6,7] )
np.dot( A,v ) # matrix product
```

Out[27]:
```
array([7, 6])
```

smaller arrays are broadcasted automatically

In [28]:
```
A = np.ones( (3,3) )
c = np.array([1,2,3])
# per default arrays are added row-wise
A + c
# like this col-wise
A + c[:,np.newaxis]
```

Out[28]:
```
array([[ 2.,  2.,  2.],
       [ 3.,  3.,  3.],
       [ 4.,  4.,  4.]])
```

useful array functions

In [33]:
```
# element-wise functions
np.sqrt(a) # square root
np.sign(a) # sign
np.log(a) # natural logarithm
np.log10(a) # decadic logarithm
np.exp(a) #exponential
np.sin(a) # trigonometric (also cos, tan, arcsin, arccos, arctan)

# non-element-wise
a.sum() # sum of all elements
a.prod() # product of all elements
a.mean() # mean
a.var() # variance
a.std() # standard deviation
a.max() # maximum
a.min() # minimum
a.sort()

# matrix

np.unique( [1,1,3,3,5] ) # get unique elements
```

Out[33]:
```
array([1, 3, 5])
```

Linear algebra in numpy

In [44]:
```
A = np.array( [[1,2,3], [4,5,6], [7,8,9]])
np.linalg.norm(A)
np.linalg.det(A) # determinant
np.linalg.eig(A) # eigenvalues and eigenvectors
np.linalg.inv(A) # matrix inverse
np.linalg.svd(A) # singular value decomposition
```

Out[44]:
```
(array([[-0.21483724,  0.88723069,  0.40824829],
        [-0.52058739,  0.24964395, -0.81649658],
        [-0.82633754, -0.38794278,  0.40824829]]),
 array([  1.68481034e+01,   1.06836951e+00,   4.41842475e-16]),
 array([[-0.47967118, -0.57236779, -0.66506441],
        [-0.77669099, -0.07568647,  0.62531805],
        [-0.40824829,  0.81649658, -0.40824829]]))
```

Advaned array acessing

In [45]:
```
a = np.array( [1,6,3,4,9,6,7,3,2,4,5] )
a[ a>4 ] # get all elements larger than 4
a[ np.logical_and(a>4, a<12) ]

# acessing via index
indices = [1,3]
a[indices]
```

Out[45]:
```
array([6, 4])
```

Scipy

Scipy is a collection of useful scientific algorithms

- scipy.integrate
- scipy.optimize
- scipy.interpolate
- etc...

In [46]:
```
import scipy as sp
import scipy.optimize
```

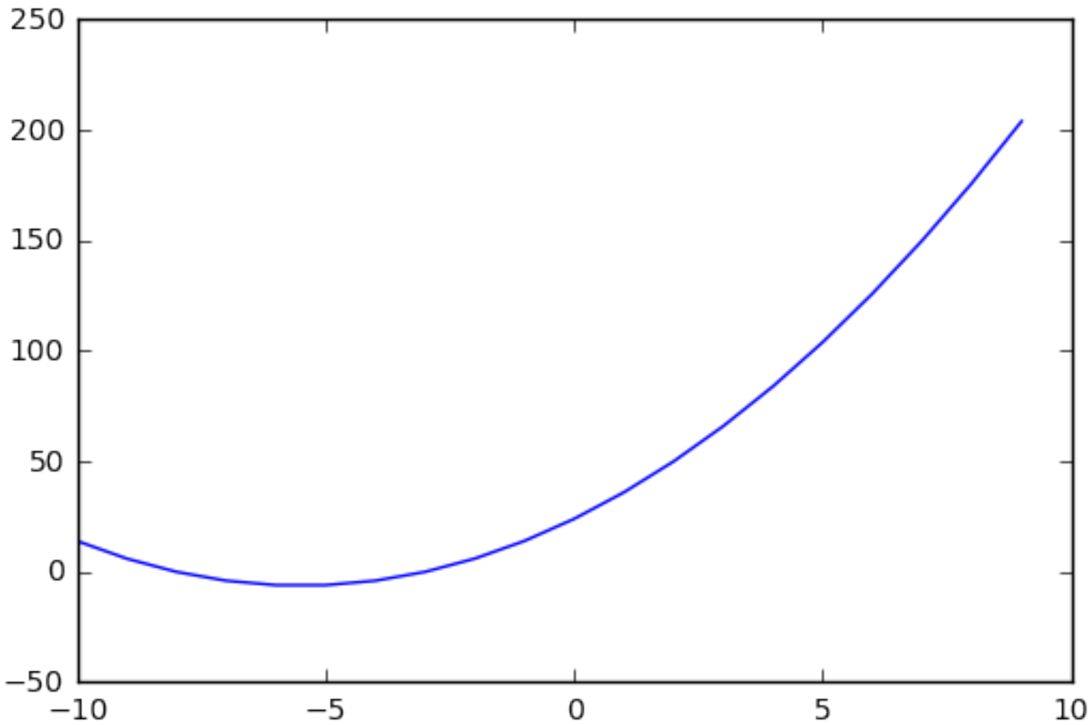
In [47]:
```
from matplotlib import pyplot as plt
#plt.style.use('ggplot')
%matplotlib inline
```

Example 1: numerical minimization of a function

In [48]:
```
# define a function
def myfunc( x ):
    return ( 3 + x ) * (8 + x)


# plot the function
x = np.arange( -10, 10, 1 )
plt.plot(x, myfunc(x) )
```

Out[48]:
```
[<matplotlib.lines.Line2D at 0x7f7450ac5c50>]
```



In [49]:
```
# minimize using scipy
sp.optimize.minimize_scalar( myfunc )
```

Out[49]:
```
     fun: -6.25
    nfev: 5
     nit: 4
 success: True
       x: -5.4999999999999998
```

Example 2: polynomials in scipy

In [65]:
```
p = scipy.poly1d([1,11,24])
print(p)

# derivative
print(p.deriv())

# integration
print(p.integ())
```

```
   2
1 x + 11 x + 24

2 x + 11
        3       2
0.3333 x + 5.5 x + 24 x
```

In [ ]: