NumPy and SciPy

What is NumPy?

- "MATLAB in python"
- provides mathematical functionality for python
- array and matrix datatypes
- element-wise calculations
- faster than lists

Cheatsheet: https://github.com/juliangaal/python-cheat-sheet/blob/master/NumPy/NumPy.md (https://github.com/juliangaal/python-cheat-sheet/blob/master/NumPy/NumPy.md)

Basics - Array creation

```
In [ ]: import numpy

        # or, for convenience:
        import numpy as np     # you could use any name, but "np" is stamdard
```

Arrays can be created from:

- python lists or sequences
- functions
- strings or files

They can only contain one data type!

```
In [ ]: a = np.array([1,2,3,4])          # data type is guessed from the values
        b = np.array([5,6,7,8], float)  # but can also be specified explicitly
        c = np.array([[1.1, 2.2, 3.3], [4.4, 5.5, 6.6]]) # 2D-array (matrix)
        d = np.arange(0,15)             # like range, but an numpy-array
```

```
In [ ]: print(a)
        print(b)
        print(c)
        print(d)
```

```
In [ ]: np.linspace(0, 2, 9)   # (start, end, num_steps)
```

```
In [ ]: np.zeros((3,2))   # ((rows, cols))   argument has to be a tuple!
```

```
In [ ]: np.ones((3,2))
```

```
In [ ]: np.random.random((3,2))
```

Arrays work element-wise!

```
In [ ]: list_a = [1,2,3,4]
        list_b = [8,7,6,5]
        array_a = np.array(list_a)
        array_b = np.array(list_b)
```

```
In [ ]: array_a+array_b
```

```
In [ ]: list_a + list_b
```

```
In [ ]: array_a + 4
```

```
In [ ]: array_a * 4
```

```
In [ ]: list_a * 4
```

```
In [ ]: array_a < 3
```

```
In [ ]: array_a * array_b   # element-wise
```

```
In [ ]: np.dot(array_a, array_b)  # dot (scalar) product
```

```
In [ ]: array_a @ array_b   # this works as well
```

```
In [ ]: np.sin(array_a)
```

Most functions can be used in two ways:

```
np.function(array_1, array_2)
array_1.function(array_2)
```

```
In [ ]: array_1 = np.array((1,2,3,4))
        array_2 = np.array((3,4,5,6))
```

```
In [ ]: np.dot(array_1, array_2)   # 1*3 + 2*4 + 3*5 + 4*6 = 50
```

```
In [ ]: array_1.dot(array_2)
```

Indexing

very similar to MATLAB

```
In [ ]: array_a = np.array([[1, 2, 3], [4, 5, 6]])
        print(array_a)
```

```
In [ ]: array_a[0,2]     # row, col
In [ ]: array_a[1,:]     # all elements in row with index 1
```

```
In [ ]: array_a[:,2]     # all elements in col with index 2
```

Array properties

```
In [ ]: array_a.shape    # again, (row, col)
```

```
In [ ]: array_a.dtype
```

```
In [ ]: array_a.size    # total number of elements
```

```
In [ ]: array_a.ndim    # number of dimensions
```

```
In [ ]: array_b = np.array((1.1, 2.2, 3.8))
        array_b.dtype
```

Working with arrays

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

Datatype can be changed...

```
In [ ]: array_c = array_b.astype(int)
        print(c)
```

Array concatenation

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

```
In [ ]: array_a+array_b   # not like this
```

```
In [ ]: np.append(array_a,array_b)
```

```
In [ ]: np.hstack((array_a,array_b))     # tuple required -> two brackets
```

```
In [ ]: array_c = np.vstack((array_a,array_b))     # tuple required -> two brackets
        print(array_c)
```

```
In [ ]: array_c.transpose()
```

```
In [ ]: array_c.T
```

```
In [ ]: array_c.flatten()
```

In [ ]:

```
for i in array_b:
    print(i)
```

In [ ]:

```
array_b.sum()
```

In [ ]:

```
array_b.prod()
```

In [ ]:

```
array_b.mean()
```

In [ ]:

```
array_b.min()
```

In [ ]:

```
array_b.max()
```

In [ ]:

```
np.abs([-5, 4, -3, 2, -1])
```

NumPy defines some mathematical constants

In [ ]:

```
np.pi
```

In [ ]:

```
np.e
```

NumPy can fit polynomials

In [ ]:

```
x = [1, 2, 3, 4, 5, 6, 7, 8]
y = [0, 2, 1, 3, 7, 10, 11, 19]
fitted_curve = np.poly1d(np.polyfit(x, y, 2))    # x-values, f(x)-values, degree
```

In [ ]:

```
%matplotlib inline
import matplotlib.pyplot as plt
plt.plot(x, fitted_curve(x))
plt.scatter(x,y, c="r")
```

SciPy

Library used for scientific computing and technical computing

- optimization
- linear algebra
- integration
- interpolation
- FFT
- signal and image processing

How to use it?

Import the wanted function with

```
from scipy.module import function
```

Don't know how the function or module is called? Google what you want to do, follow the link to the scipy-documentation...

f.ex. gradient descent (find local minimum of n-dimensional function):
https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.minimize.html
(https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.minimize.html)

```
from scipy.optimize import minimize
```

Example: Spearman correlation coefficient

scipy.stats.spearmanr(x, y)
returns: correlation, pvalue

The Spearman correlation is a nonparametric measure of the monotonicity of the relationship between two datasets. Unlike the Pearson correlation, the Spearman correlation does not assume that both datasets are normally distributed. Like other correlation coefficients, this one varies between -1 and +1 with 0 implying no correlation. Correlations of -1 or +1 imply an exact monotonic relationship. Positive correlations imply that as x increases, so does y. Negative correlations imply that as x increases, y decreases.

The p-value roughly indicates the probability of an uncorrelated system producing datasets that have a Spearman correlation at least as extreme as the one computed from these datasets. The p-values are not entirely reliable but are probably reasonable for datasets larger than 500 or so. source:
https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.spearmanr.html
(https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.spearmanr.html)

In [ ]:

```
from scipy.stats import spearmanr
```

In [ ]:

```
a = np.sin(np.linspace(0,10*np.pi, 1000))
b = -np.sin(np.linspace(0,10*np.pi, 1000))
r, _ = spearmanr(a, b)
print(r)
```