# Speech Communication

## Exercise 1

---

Further help can be found by searching the documentation of python modules such as *`numpy`*, *`matplotlib`*, *`librosa`* or *`scipy`* in the Internet. Python (the command line interpreter) also provides integrated help about the language, language features and available functions by using the command `help()` or `help("functionname")`.

## Exercise 1.1 Input audio data

Download the wave file `13ZZ637A.wav` from ISIS which contains the `exercise_1.py` file. `librosa` is a useful python module which contains various audio processing functionalities. Use `librosa.load("<filename>", sr=None)` to get the audio data as a `numpy.ndarray` (which is 1D for mono-channel audio, a matrix for multi-channel audio).

`matplotlib` provides Matlab-like diagram-plotting functionalities. Import `matplotlib.pyplot` and create a figure with **two** diagrams inside by using the `pyplot.subplots()` function. Diagrams are called `Axes` in matplotlib and they contain the method `Axes.plot(self, x_values, y_values, …)`. Note: methods are functions which are contained in objects and the owning object is implicitly passed as the first method argument `self` to the function. Use this `Axes.plot(self, …)` method to draw the time sequence of samples of the signal into the first diagram. Add labels to both axes with either `Axes.set(self, …)` or with the individual methods `Axes.set_xlabel(self, …)`, `Axes.set_ylabel(self,…)` etc.

The x-axis represents the duration of the signal, the unit is seconds. Therefore you have to convert each position ("index") of the samples you got from the `librosa.load` function into seconds.

Tip: try `numpy.linspace()` for this purpose and obtain the total number of samples with either the `shape` attribute contained in the vector, or by using Python's builtin function `len()`.

Now, print out answers to some questions on the console.
- Which sampling frequency does the signal have?
- How many samples are in the signal?
- How long is the duration of the signal in seconds?
- What is the value of the greatest amplitude? Tip: Python's builtin functions `max()` and `abs()`

## Exercise 1.2 Zero crossing rate

In Exercise 1.2 we want to know, how often does the signal intersect with zero in time intervals of the signal.

First, *`def`*ine a function `compute_zero_crossings(samples: numpy.ndarray) -> numpy.ndarray` which takes a 1D array of sound samples. It should detect the positions with zero intersections in the signal and output a 1D array with only `0`s and `1`s. A `1` indicates a zero intersection starts in the current sample at position `i` or after the previous sample in the signal. The first vector element will be zero.

Assume, a zero crossing occurs at `i` when the last sample at `i-1` is not zero and has a different `numpy.sign()` than the current sample at `i`. Note: a zero crossing will include the case where one ore more sequential zeros are surrounded by samples with equal sign.

*Example:*

```
signal = numpy.array([1, 0.5, -0.5, -1, 0, -1, 2, 1.5, 0, 0.5, 0])
zero_crossings = compute_zero_crossings( signal )
```

In the 2nd part, you should use the output of `compute_zero_crossings()` in order to compute the sum of zero crossings over intervals of 10ms preceding each sample in the signal. For example, when computing the cumulative zero crossings at sample 1000, it is supposed to look at all samples in a 10ms interval (including sample 1000 itself) and count the number of zero crossings in that interval. This procedure should be repeated for all samples. Tip: you can either do it manually with Python's builtin `sum()` function together with array slicing `vector[start_position : end_position+1]`. The advanced solution uses `numpy.convolve()` with `numpy.ones()`.

Please plot the course of the zero-crossing rate over time into the 2nd diagram.

What do you think are the parts of the signal that contain a high frequency of zero crossings, and which are the parts that contain a low frequency?