You have to define an *equaliser* function, which has the following arguments:

* *input\_audio*: input audio signal data
* *window*: window function, the default function can be numpy.hanning(M) where M defines a frame size in samples, i.e. if you want to get 20ms frames you need to take M = 0.02\*fs (20ms = 20/1000 s), where fs is a sampling rate and should be fixed at the beginning of a script and you should check that all the input and output audios have the given sampling rate, if not, you have to resample them in advance. Here the recommendation is to take fs = 16000
* *step*: (or in other word *hop*) this is the distance between two overlapping frames (you are familiar with it from spectrogram task), and by default you can pick it as (frame size) / 2,
* *equaliser*: equaliser represents the discrete version of equaliser filter amplitude response. So it has non negative values and is defined in DFT domain and should have the same amount of coeffs as *numpy.fft.rfft*, which is in case of even M (frame size) equals M / 2 + 1, so the len of equaliser should be M / 2 + 1.   
  For instance if we want to take low-pass filter with f\_c cutoff frequency, than we need to put   
  where f[k] is a frequency corresponding to the k-th index.

Your function should apply equaliser using Overlap-Add method and convolution theorem, and return the result, i.e.

It should take the input audio, divide it into overlapping frames by frame size and step (hop), multiply each frame with the window function, apply DFT (np.fft.rfft), then by convolution theorem multiply it by equaliser, apply IDFT (np.fft.irfft) which gives equalised frame, then sum all new frames at their positions in the audio signal. As a result you will get an equalised audio analog.

To verify that the function is working, you should experiment with various equaliser functions, particularly low-pass filtering, and by spectrogram check that it actually cuts the frequencies greater than cut-off frequency (f\_c).