Part (a)

Listen to the recording you made, stored in the file `recording.wav` . You can load recordings using the `load_recording` function that we have written for you and imported. You can play recordings using the `play` function that we have also written and imported.

In [1]:
```
import numpy as np
from utils import load_recording, play, save_recording

RECORDING_FILE = "recording.wav"

r = load_recording(RECORDING_FILE)
play(r)
```

-0:10

Part (b)

Let $\vec{r}$ be your recording. Let us say you have access to the true lecture given by $\vec{l}$. You know that your received vector and the lecture have the relationship

$$\vec{r} = \alpha\vec{l} + \vec{n},$$

where α is an unknown constant. Estimate $\vec{n}$ by projecting $\vec{r}$ ontol $\vec{l}$ to recover α. What remains is $\vec{n}$. Assume that $\vec{l}$ is orthogonal to $\vec{n}$.

In [5]:
```
# Note that l and r are 1D arrays, not 2D arrays, so calling np.linalg
def projection(l, r):
    return np.dot(l, r) / (np.linalg.norm(l)**2) * l
```

In [6]:
```
def recover_noise(r, l):
    return r - projection(l,r)
```

In [7]:
```
#We use the technique above to recover candidate interference signals.

#noisy_lectures contains the lecture recordings with interference
noisy_lectures = [load_recording("noisy_lecture_{}.wav".format(i+1)) f

# lectures contains the clean lectures that you played to understand t
lectures = [load_recording("lecture_{}.wav".format(i+1)) for i in rang

# interferences is a matrix whose columns contain the possible interfe
interferences = np.column_stack([recover_noise(r_i, l_i) for r_i, l_i

#you can change the index 0 below to play different lectures and recor
play(lectures[0])
play(noisy_lectures[0])
play(interferences[:, 0])
```

-0:14

-0:20

-0:21

Part (c)

Now, given $\vec{r}$ and the $\vec{n}_i$, and the model

$$\vec{r} = \vec{l} + \sum_{i=1}^{s} \beta_i \vec{n}_i,$$

use least squares to recover $\vec{l}$. The $\vec{n}_i$ are computed from the $\vec{r}_i$ using your function from the previous part.

In [11]:
```
#r is the signal you have recorded
r = load_recording(RECORDING_FILE)

# Project r onto the interference signals to recover the component of
# What remains must be the lecture.

A = interferences
b = r

# Hint, use least squares
betas = np.linalg.lstsq(A, b)[0]
print(betas)

# This is the recovered lecture. Have you successfully recovered a
# noise-free signal? Or is it still noisy?
l = b - A.dot(betas)

play(l)
```

```
[-0.07080106 -0.09364032  0.11021623  0.02728798]
```

-0:16

Part (d)

Now, we will include the effect of the travel time of the noise signals, using the model

$$\vec{r} = \vec{l} + \sum_{i=1}^{s} \beta_i \vec{n}_i^{(k_i)}.$$

Recover $\vec{l}$ using this new model, using OMP, by filling in the blanks in the below code block.

In [51]:
```
from utils import cross_correlate

r = load_recording(RECORDING_FILE)
interferences = [recover_noise(r_i, l_i) for r_i, l_i in zip(noisy_lec

k = np.zeros(4, "int")

vecs = []

# the initial residual for OMP
residual = r

for _ in range(4):
    best_corr = float("-inf")
    best_vec = None
    # We first iterate over all the interferences n_i
    for i, n_i in enumerate(interferences):
        # for each interference, we look through its correlation with

        # Fill in the arguments to cross_correlate
        for k_i, corr in enumerate(cross_correlate(
            residual,
            n_i
        ) # This function returns a vector of cross correlation values
          # the residual/received signal with every possible delay of
        ):
            # we find the (noise, shift) pair that maximizes the corre
            if corr > best_corr:
                best_corr = corr
                best_vec = (i, k_i)
    i, k_i = best_vec
    k[i] = k_i

    # we shift the best noise by the best shift and add it to our list
    vecs.append(np.roll(interferences[i], k[i]))

    A = np.column_stack(vecs) # this is the matrix that captures all t

    # Use least squares to update the residual
    residual = residual - np.array(vecs).T.dot(np.linalg.lstsq(np.arra

l = residual
play(l)
```

-0:00

In [ ]: