# 데이터통신

## - 실습 5주차. Unicode over sound with noise -

충남대학교 컴퓨터융합학부

# Feedback

- 제출률: 87% ( 40/46 )
- 과제 익명 설문조사 [클릭] / 난이도 및 개선 희망 사항 - 희망자 진행
- 4주차 과제 정답 코드 배포

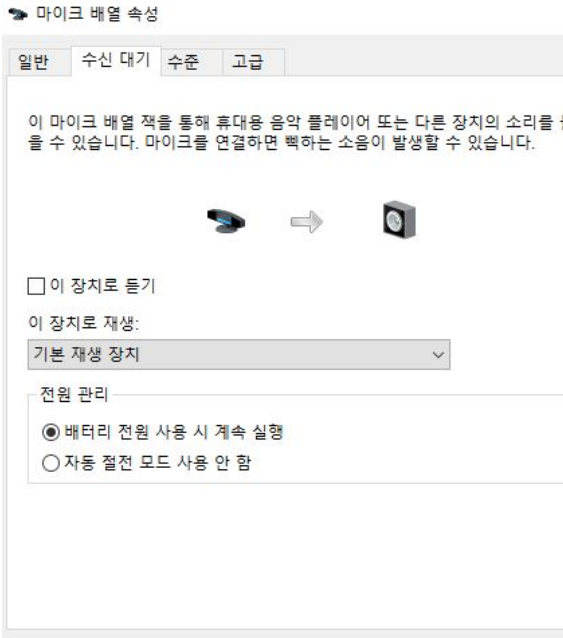
# 많이 받은 질문

**마이크 관련 문의**

- 현재 안되는 것이 무엇인지 정확히 파악하는 것이 중요.
  [마이크]가 문제인지, [파이썬 자체]가 문제인지, [내가 작성한 파이썬 코드]가 문제인지 파악하는 내용이 중요.

  **마이크 문제 확인 방법:**
  소리 – 녹음 – 사용하고자 하는 마이크 – 속성 – 수신 대기 – 이 장치로 듣기 체크
  해당 내용 실행시 마이크에 들어가는 소리가 나한테 다시 들림.
  이 방법을 통해서 마이크 자체가 작동이 제대로 안되는지,
  마이크에 들어가는 잡음이 너무 많은지,
  마이크가 자동으로 잡음(모스부호 등)을 차단하는 지 판단 가능.

  마이크에 문제가 있을 시, 연구실에서 마이크 대여 가능.



# 그러니 한번 더!: Data over sound with noise

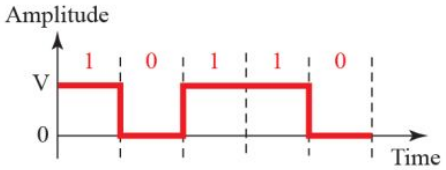
- 물리계층 (L1) 에 어떻게 데이터를 보낼 수 있는가?
- **그런데, 소음이 있다면? ⇒ Threshold (noise gate) 처리**
- **그런데, 언제 데이터가 시작하는지 모른다면? ⇒ Data start detection 처리**
- **그런데, 언제 데이터가 끝나는지 모른다면? ⇒ Data end detection 처리**
- Send data over sound

  ⇒ **Send ~~morse code~~ unicode over sound with noise**

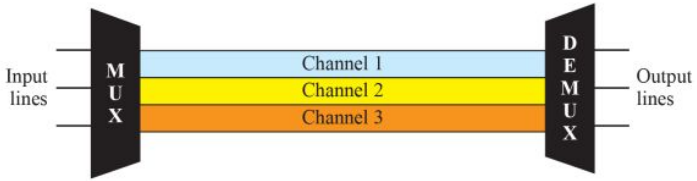
# 주차별 이론 - 실습 연계: Data over sound

Data over sound

- 물리레이어 (L1, 공기) 에 데이터를 실어서 통신




***Unipolar NRZ scheme***




1. 통제된 환경에서의 데이터 전송
    - WAV 파일의 시작 ⇒ 데이터의 시작
2. Stream 환경에서의 데이터 전송
    - Wifi ⇒ Sound, 전파 ⇒ 음파
    - 신호가 계속 입력되는 중 (Streaming)
    - 간섭 대응하지 않음 → 조용한 곳에서만!
    - 해결해야할 문제: 데이터의 시작과 끝 인식?
3. 간섭이 있는 Stream 환경에서의 데이터 전송
4. 오류 탐지 및 복원

# 실습: Unicode ↔ Bytes 변환

# Unicode [클릭]

- **Unicode**, formally the Unicode Standard, is an information technology standard for the consistent **encoding, representation, and handling** of text expressed in most of the world's writing systems.
  - ex) 'A' - Ascii (0x41), Unicode (U+0041)
- 최신 입력기는 Unicode 를 입력함!

데

데 Hangul Syllable De

U+B370

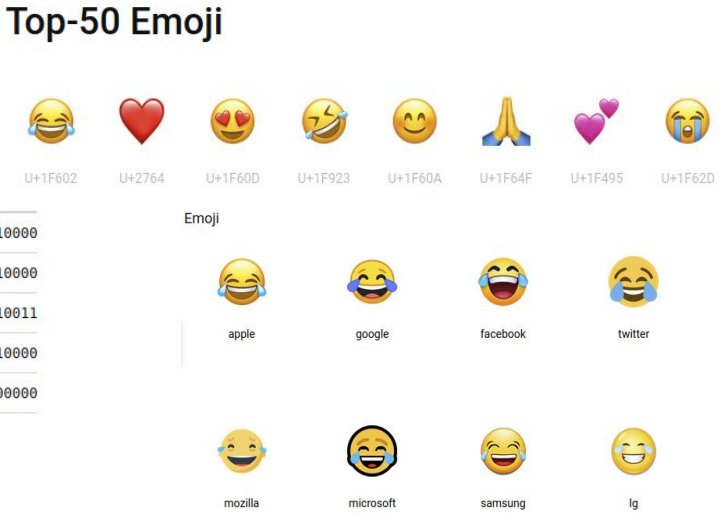
Encoding

| Encoding | hex | dec (bytes) | dec | b |
|---|---|---|---|---|
| UTF-8 | EB 8D B0 | 235 141 176 | 15437232 | 11101011 10001101 10110000 |
| UTF-16BE | B3 70 | 179 112 | 45936 | 10110011 01110000 |
| UTF-16LE | 70 B3 | 112 179 | 28851 | 01110000 10110011 |
| UTF-32BE | 00 00 B3 70 | 0 0 179 112 | 45936 | 00000000 00000000 10110011 01110000 |
| UTF-32LE | 70 B3 00 00 | 112 179 0 0 | 1890779136 | 01110000 10110011 00000000 00000000 |

```
>>> b'\xEB\x8D\xB0'.decode('utf-8')
'데'
```

직접 유니코드를 살펴보자! [클릭]

- 심지어 이모지도 있음!!
  - 운영체제, 프로그램별로 이모지 모양만 다름 (코드는 같음)

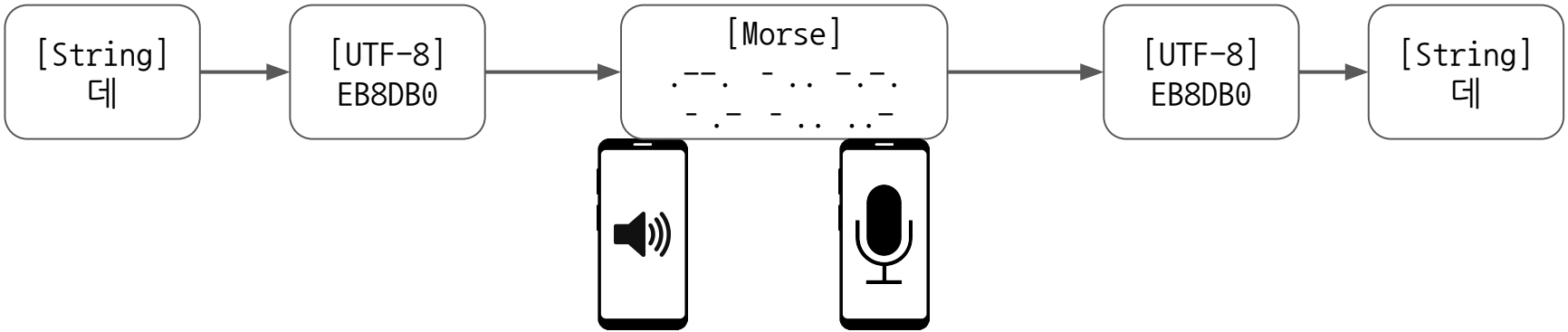


# 실습4에서 변경: Only english ⇒ Unicode!

아이디어:

- Morse code 로 Encoding 된 Text 전송

  ⇒ Unicode Encoding 된 Hex String 을 Morse code 로 Encoding 하여 전송



# 실습) Unicode → Bytes → String → String → Bytes → Unicode

- 사용자가 입력한 내용을 Unicode 인코딩

⇐ 이 사이에 전송이 이루어 짐! ⇒

- 인코딩된 Bytes 를 Unicode로 디코딩

```
# 사용자 입력 파트
user_input = input('Input the text (Unicode): ').strip()
print(f'User input: {user_input}')

# str => hex 변경 파트
byte_hex = user_input.encode('utf-8')
byte_string = byte_hex.hex().upper()
print(f'byte_string: {byte_string}')

# 여기서 byte_string을 서로 통신했을 것!
# 이 아래의 byte_string은 통신으로 받은 str 이라고 가정

# byte_string => str 변경 파트
client_byte_hex = bytes.fromhex(byte_string)
client_byte_string = client_byte_hex.hex().upper()
print(f'client_byte_string: {client_byte_string}')


# str => 사용자 출력
client_output = client_byte_hex.decode('utf-8')
print(f'User output: {client_output}')
```

```
harny@LuHa-X1C9 ~/Github/Hobby/Unicode $ python3 PythonUnicode.py
Input the text (Unicode): 한글 English $#@!? 😂
User input: 한글 English $#@!? 😂
byte_string: ED959CEAB88020456E676C69736820242340213F20F09F9882
client_byte_string: ED959CEAB88020456E676C69736820242340213F20F09F9882
User output: 한글 English $#@!? 😂
```

# Code map 변경!!) Unicode Morse Code Map

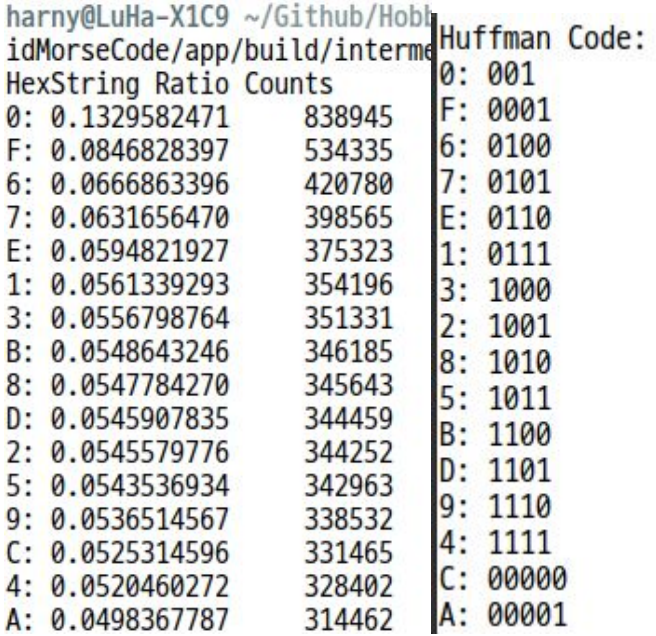
- 0~F 까지만 사용하므로 불필요한 것 제거

```
code = {'0': '..-',
        '1': '.---',
        '2': '-..-',
        '3': '-...',
        '4': '----',
        '5': '-.--',
        '6': '.-..',
        '7': '.-.-',
        '8': '-.-.',
        '9': '---.',
        'A': '....-',
        'B': '--..',
        'C': '.....',
        'D': '--.-',
        'E': '.--.',
        'F': '...-'}
```

# 참고) Unicode Morse Code Map 어떻게 만들었을까?

목적: 전송 시간 단축

1. '0~F' 중에서 가장 많이 사용되는 문자는 무엇일까?
2. 이론 5주차 PDF 파일 Binary 분석: F, 0 이 가장 많이 사용됨
3. '0' 전송 시간: '-----' ⇒ 3x5 + 1x4 = **19 units**
4. 변경 아이디어: 많이 사용될 수록 짧게 인코딩
   ⇒ Huffman Code
5. '0' 전송 시간: '..-' ⇒ 1x2 + 3x1 + 1x2 = **7 units**



```
harny@LuHa-X1C9 ~/Github/Hobb
idMorseCode/app/build/interme
HexString Ratio Counts
0: 0.1329582471     838945
F: 0.0846828397     534335
6: 0.0666863396     420780
7: 0.0631656470     398565
E: 0.0594821927     375323
1: 0.0561339293     354196
3: 0.0556798764     351331
B: 0.0548643246     346185
8: 0.0547784270     345643
D: 0.0545907835     344459
2: 0.0545579776     344252
5: 0.0543536934     342963
9: 0.0536514567     338532
C: 0.0525314596     331465
4: 0.0520460272     328402
A: 0.0498367787     314462
```

```
Huffman Code:
0: 001
F: 0001
6: 0100
7: 0101
E: 0110
1: 0111
3: 1000
2: 1001
8: 1010
5: 1011
B: 1100
D: 1101
9: 1110
4: 1111
C: 00000
A: 00001
```

# 실습: PyAudio 활용 오디오 재생 및 녹음

# 오늘의 실습 및 과제: Unicode 메신저 on x86 PC

Unicode 발신

- 사용자로부터 텍스트 및 이모티콘 입력
  - 예외처리 추가해도 좋음
- 입력된 Unicode **스피커**로 재생

Unicode 수신

- **마이크**로부터 소리 입력
  - 같은 소리도 인식이 안 될 때가 있음
  - 소음 때문에!
- 입력된 소리 Unicode 로 변환

# PyAudio 활용 재생, 녹음

Python 3 필요 패키지: pyaudio [클릭]

- Cross-platform 오디오 처리 라이브러리
- OS 마다 설치 방법이 다르므로 확인!
- venv 활용해서 독립 설치 추천!! [클릭]
    - 다른 개발 환경과 충돌 안 됨

우리가 필요한 기능

- PCM 32bit array 스피커에서 재생
- PCM 32bit array 마이크에서 녹음

Python 3 다른 Audio 패키지 사용 가능!

- 자신의 PC 환경에서 PyAudio가 정상 작동하지 않는다고 판단되면 다른 Audio 패키지 사용 가능
    - 보고서에 적어둘 것!

# PyAudio 활용 사인파 재생

- 당연히 사운드 출력 장치가 있어야 함!
  - `pactl list sources short`
- PC에 성능에 따라서 chunk_size를 조정해야할 수 있음
  - 소리에 노이즈가 끼거나,
    모든 데이터가 재생 안 되거나

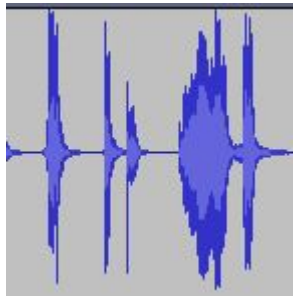
```
$ pactl list sources short
1	alsa_output.usb-Lenovo_ThinkPad_Thunderbolt_4_Dock_USB_Audio_000000000000-00.analog-stereo.monitor	module-alsa-card.c	s16le 2ch 44100Hz	IDLE
2	alsa_input.usb-Lenovo_ThinkPad_Thunderbolt_4_Dock_USB_Audio_000000000000-00.mono-fallback	module-alsa-card.c	s16le 1ch 44100Hz	SUSPENDED
3	alsa_output.pci-0000_00_1f.3-platform-skl_hda_dsp_generic.HiFi__hw_sofhdadsp_5__sink.monitor	module-alsa-card.c	s16le 2ch 48000Hz	SUSPENDED
4	alsa_output.pci-0000_00_1f.3-platform-skl_hda_dsp_generic.HiFi__hw_sofhdadsp_4__sink.monitor	module-alsa-card.c	s16le 2ch 48000Hz	SUSPENDED
5	alsa_output.pci-0000_00_1f.3-platform-skl_hda_dsp_generic.HiFi__hw_sofhdadsp_3__sink.monitor	module-alsa-card.c	s16le 2ch 48000Hz	SUSPENDED
6	alsa_output.pci-0000_00_1f.3-platform-skl_hda_dsp_generic.HiFi__hw_sofhdadsp__sink.monitor	module-alsa-card.c	s16le 2ch 48000Hz	SUSPENDED
7	alsa_input.pci-0000_00_1f.3-platform-skl_hda_dsp_generic.HiFi__hw_sofhdadsp__source	module-alsa-card.c	s16le 2ch 48000Hz	SUSPENDED
8	alsa_input.pci-0000_00_1f.3-platform-skl_hda_dsp_generic.HiFi__hw_sofhdadsp_6__source	module-alsa-card.c	s16le 4ch 48000Hz	SUSPENDED
9	alsa_input.usb-046d_Logitech_BRIO_262080C8-02.analog-stereo	module-alsa-card.c	s16le 2ch 48000Hz	SUSPENDED
10	monitorsink.monitor	module-null-sink.c	s16le 2ch 44100Hz	SUSPENDED
```

```python
import math
import struct
import time

import pyaudio

INTMAX = 2**(32-1)-1


def main():
    t = 10
    fs = 48000
    f = 261.626 # C4
    audio = []
    for i in range(int(t*fs)):
        audio.append(int(INTMAX*math.sin(2*math.pi*f*(i/fs))))

    p = pyaudio.PyAudio()

    stream = p.open(format=pyaudio.paInt32,
                    channels=1,
                    rate=fs,
                    output=True)

    chunk_size = 1024
    for i in range(0, len(audio), chunk_size):
        chunk = audio[i:i+chunk_size]
        stream.write(struct.pack('<' + ('l'*len(chunk)), *chunk))

    stream.stop_stream()
    stream.close()
    p.terminate()


if __name__ == '__main__':
    main()
```

# PyAudio 활용 녹음

- 당연히 사운드 입력 장치가 있어야 함!
  - `pactl list sinks short`
- 마이크 성능 및 주변 소음에 따라서 조용할 때 표준 편차가 차이남

```
1861783.8747994613
1977229.9031550928
1476431.0843265809
169089351.43982953
771911094.0400583
763446804.1845648
866265343.0994262
631217117.3065822
208707774.6364386
82640538.73715007
31017519.473244455
10596873.294875246
```

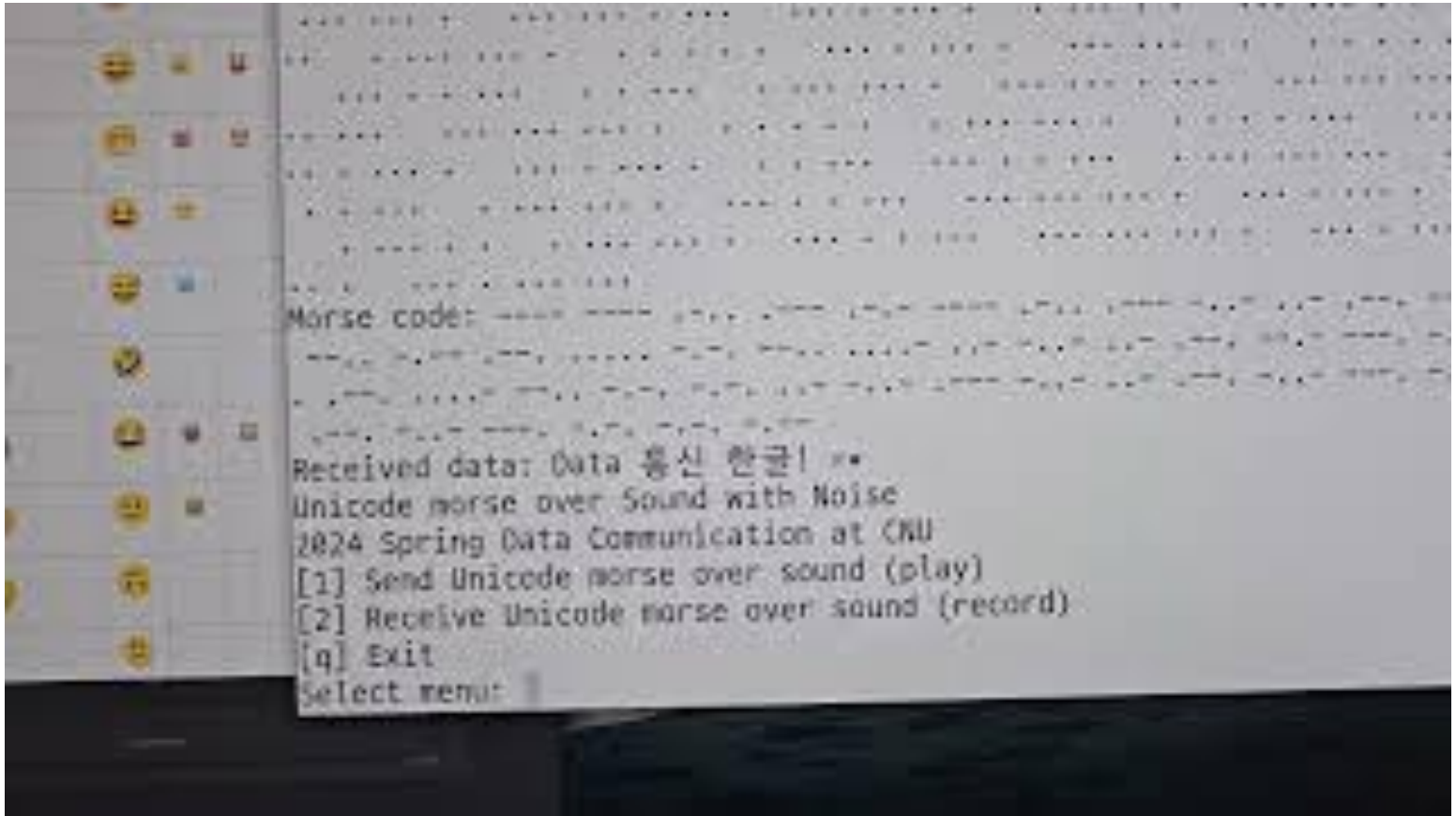


```
$ pactl list sinks short
1	alsa_output.usb-Lenovo_ThinkPad_Thunderbolt_4_Dock_USB_Audio_00000000000-00.analog-stereo	module-alsa-card.c	s16le 2ch 44100Hz	SUSPENDED
2	alsa_output.pci-0000_00_1f.3-platform-skl_hda_dsp_generic.HiFi__hw_sofhdadsp_5__sink	module-alsa-card.c	s16le 2ch 48000Hz	SUSPENDED
3	alsa_output.pci-0000_00_1f.3-platform-skl_hda_dsp_generic.HiFi__hw_sofhdadsp_4__sink	module-alsa-card.c	s16le 2ch 48000Hz	SUSPENDED
4	alsa_output.pci-0000_00_1f.3-platform-skl_hda_dsp_generic.HiFi__hw_sofhdadsp_3__sink	module-alsa-card.c	s16le 2ch 48000Hz	SUSPENDED
5	alsa_output.pci-0000_00_1f.3-platform-skl_hda_dsp_generic.HiFi__hw_sofhdadsp__sink	module-alsa-card.c	s16le 2ch 48000Hz	SUSPENDED
6	monitorsink	module-null-sink.c	s16le 2ch 44100Hz	SUSPENDED
```

```python
import math
import statistics
import struct
import time

import pyaudio


def main():
    t = 10
    fs = 48000

    p = pyaudio.PyAudio()

    stream = p.open(format=pyaudio.paInt32,
                    channels=1,
                    rate=fs,
                    input=True)


    audio = []

    chunk_size = 1024
    for _ in range(0, math.ceil(fs / chunk_size)*10):
        data = struct.unpack('<' + ('l'*chunk_size),
stream.read(chunk_size))
        audio.extend(data)
        print(statistics.stdev(data))

    stream.stop_stream()
    stream.close()
    p.terminate()


if __name__ == '__main__':
    main()
```

# 과제: Send Unicode on air

# 실습 및 과제 데모


Morse code:
Received data: Data 통신 한글! ^•
Unicode morse over Sound with Noise
2024 Spring Data Communication at CNU
[1] Send Unicode morse over sound (play)
[2] Receive Unicode morse over sound (record)
[q] Exit
Select menu:

# Unicode over Sound 사용자 인터페이스

- 사용자가 데이터 전송, 입력 선택
- 데이터 송신을 선택하면 전송할 데이터를 다시 입력받아야 함!
- 데이터 수신을 선택하면 녹음을 시작하고 동시에 들어오는 데이터를 해석해야 함!
  - 환경에 맞게 소리에 대한 Threshold 설정!

```
import math
import statistics
import struct
import time
import wave

import pyaudio

INTMAX = 2**(32-1)-1


def send_data():
    pass


def receive_data():
    pass


def main():
    while True:
        print('Unicode over Sound with Noise')
        print('2025 Spring Data Communication at CNU')
        print('[1] Send Unicode over sound (play)')
        print('[2] Receive Unicode over sound (record)')
        print('[q] Exit')
        select = input('Select menu: ').strip().upper()
        if select == '1':
            send_data()
        elif select == '2':
            receive_data()
        elif select == 'Q':
            print('Terminating...')
            break;


if __name__ == '__main__':
```

# 보고서에 꼭 적어야 할 것!: 혼자서 or 둘이서

- 가능하면 컴퓨터 2대 이상! 어떻게 해도 방법이 없을 때 혼자, 스마트폰을 이용해서!
- 소스코드 유사도 검사 예정! **실험을 같이 해도 된다는 것!** 카피 허용이 아님!

- 컴퓨터 (스피커, 마이크 포함) 2대 이상
  - 한쪽에서 재생
  - 다른 한쪽에서 녹음
- 컴퓨터 1대, 스마트폰 1대
  - 컴퓨터에서 재생, 동시에 스마트폰으로 녹음
  - 스마트폰으로 녹음한 파일 재생, 동시에 컴퓨터에서 녹음

# 목표1) 소음에 대한 처리

- 적응형으로 하면 좋지만… 어려우므로 상수로 고정
- MORSE_THRESHOLD 와 같은 변수를 선언해두고 해당 변수보다 큰 값만 ‘소리’로 인정
    - OBS Noise gate [링크]

```
if data >= MORSE_THRESHOLD:
```

# 목표2) 데이터가 언제 시작하는지 알아내기 (동기화)

- 소음이 아닌 데이터의 도착 ⇒ 데이터의 시작
- 따라서, 계속 녹음하면서 데이터가 시작하는 순간부터 처리 시작

```
if not tuning:
```

# 목표3) 데이터가 언제 끝나는지 알아내기 (동기화)

- 소음만 계속 도착 ⇒ 데이터의 끝
- 따라서, 소음만 N초 발생하면 녹음 종료

```
if unseen >= (UNSEEN_THRESHOLD/UNIT):
```

# 힌트) 어떻게 unit size 만큼의 오디오 데이터를 모으나요?

1. data 도 일종의 배열이므로 (tuple) data 를 순회하며 값을 N개 모은다.
    - 리스트에 넣기 or 값을 더해서 나중에 평균을 내기 등등…
2. N 이 unit size가 되면 연산 시작!
    - N 계산 방법: samplerate * unit = 48000 * 0.1 = 4800 개
    - 즉, 우리의 경우 4800개의 데이터가 0.1초 동안의 데이터라는 의미
3. 이후 진행…

# 우리의 Morse code 생성 규칙

**Our morse code rules**

- Timing unit: **100 ms (0.1 s)**
- Frequency: **523.251 hz (C5)**

Our WAV file specification

- Channels: 1 (Mono)
- Sample rate: 48000

국제 표준 적용

- dits: 1 unit
- dahs: 3 units
- dits dahs 사이: 1 unit
- 문자 사이: 3 units
- 단어 사이 (공백, 스페이스): 7 units

**International Morse code is composed of five elements:**[1]:§3

1. short mark, dot or *dit* ( ▪ ): "dit duration" is one time unit long
2. long mark, dash or *dah* ( ▬ ): three time units long
3. inter-element gap between the *dits* and *dahs* within a character: one dot duration or one unit long
4. short gap (between letters): three time units long
5. medium gap (between words): seven time units long

# 실습5 과제 정리 및 채점 배점

채점 배점:

- 보고서 (제출): 1점
- Unicode 송신 기능 완성: 2점
- Unicode 수신 기능 완성: 4점
  - 디코딩해서 출력할 것!
  - 수신 받을때 **실시간 출력** 되게 할 것
  - 수신 받을시 영어 + 숫자 + 한글 + 특수문자 포함
    (Ex. 2025 데이터 Communication ★ )
    글자수는 최소 4글자 이상 (ex. 2 데 D ★)
    인식에 대한 부분점수 제공 (ex. 0 데 D ★ 로 인식시 3점)
- 목표 1,2,3 해결 : 각 1점
  - 데이터 도착할 때부터 연산 (이전 녹음 데이터 버림)
  - 데이터 도착 완료후 N초뒤 종료 (자동 녹음 종료, 최소 3초)

- Timing 미준수시 감점

- zip 파일 압축: DC02-학번-이름.zip
  - **보고서 (학번-이름.pdf)**
    - 실험 환경: 컴퓨터 2대 or 스마트폰 포함
      - 컴퓨터 2대인 경우 둘이서 or 혼자서
    - 목표 1, 2, 3을 어떻게 해결했는지 보고서에 꼭 포함!
      - 알고리즘 설명 or 코드 설명
    - 송신/수신 영상 제출 필수!
      - Youtube or 영상 파일
  - **소스코드 (.py)**
- e-learning 사이버캠퍼스 제출

- 제출 기한: 2025. 04. 08. 23:59 (+ 추가 제출 1일)
  추가제출기간내 제출시 30% 감점.