

# 작곡 세미나

---

## 9. MIDI 프로그래밍

2024/07/18

Andantino

# 개발자라면 이런 생각 해 본 적 없나요?

NWC나 MuseScore 같은 프로그램은 어떻게 만드는 걸까?

컴퓨터가 음악을 작곡할 수는 없을까?

화음이랑 조성 찾는 거 귀찮은데 자동으로 찾아주면 좋겠다.

# 오늘 강의를 들으면 만들 수 있는 것



<https://github.com/salt26/chromatic-piano-roll>

# 오늘 할 일

- MIDI란?
  - 컴퓨터에서의 음악 표현 방법
  - 기호 음악
  - 재생하는 디바이스와 사운드
  - Message 타입
    - Note on
    - Note off
  - Channel, Key, Velocity
  - 타이밍
    - 음표와 타이밍
    - 쉼표
- MIDI Binary 파일 해부

# 오늘 할 일

- MIDI 프로그래밍 (실습 포함)
  - Python MIDI 라이브러리
    - 설치 방법
    - 예제
- C# MIDI 라이브러리
  - Message 생성 (ChannelMessage, ChannelCommand)
  - 출력 디바이스 설정 및 음 재생 (OutputDevice)
  - MIDI 파일로 저장 (Sequence, Track)

# 시작하기 전에...

- 실습을 위한 준비물이 있습니다.
  - Rider (또는 Visual Studio)
  - Python3
- 설치 방법은 0주차 자료에 잘 안내되어 있습니다.
- 오늘은 7주차 실습 때 사용했던 “작곡 도우미”와 같은 MIDI 프로그램을 짜는 방법을 다룹니다.
  - 오늘과 다음 주에 다룰 내용을 배우면, 피아노 연주, 작곡, 화음 학습, 음악 분석 또는 음악 생성 애플리케이션을 만들 수 있습니다!

# MIDI 기본

---

# 음악 표현 방법

- 컴퓨터에서 음악을 표현하는 방법은 크게 세 가지가 있습니다.

1. 음성 표현법

2. 연주 표현법

3. 악보 표현법



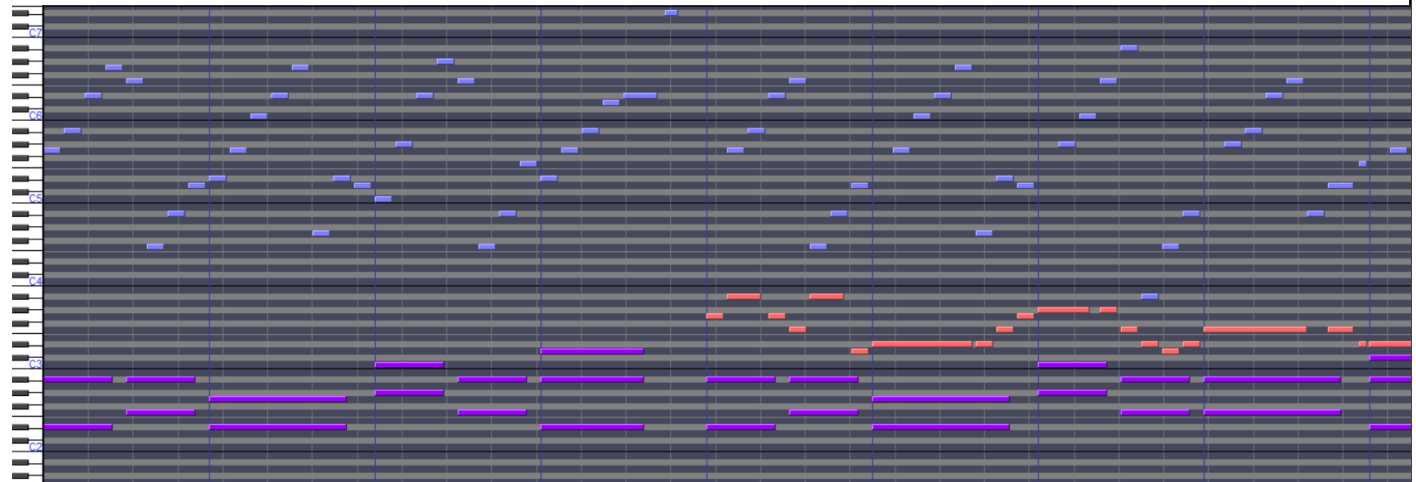
# 음성 표현법

- 가장 낮은 수준의 음악 표현법
- 음악을 오디오 파형의 형태로 저장
- 음색 등의 청각적 특징을 보존합니다.
- 음 높이, 박자 등의 음악적 특징을 얻으려면 별도의 신호 처리가 필요합니다.



# 연주 표현법

- 중간 수준의 음악 표현법
- 연주에 필요한 정보를 저장
  - 건반을 언제 누르고 떼는지, 어떤 세기로 누르는지 등
- 연주에서만 나타나는, 악보에 없는 섬세한 표현을 담고 있습니다.
- 음색 등의 청각적 특징은 담고 있지 않습니다.
- 음표의 악보 상 길이나  
마디 구분과 같은 추상적인  
정보를 다루려면 별도의  
처리 과정이 필요합니다.



# 악보 표현법

- 가장 높은 수준의 음악 표현법
- 음악을 시각적인 악보의 형태로 저장
- 사람이 읽기 쉽고 음악적 특징과 구조를 파악하기에 용이합니다.
- 음악을 소리로 출력하거나 연주하는 방법이 하나로 정해지지 않습니다.

The image displays a musical score for a vocal and piano performance. The score is written in E-flat major (three flats) and 4/4 time. The tempo is marked as 65 beats per minute. The key signature is E-flat major, and the time signature is 4/4. The score is divided into two staves: Vocal and Piano. The Vocal staff is in bass clef and contains four measures of whole notes, each with a chord symbol above it: E-flat, B-flat, C minor, A-flat, B-flat, and E-flat major 7. The Piano staff is in treble and bass clef and contains four measures of music. The first measure is marked with a tempo of 65 and a dynamic of *mf*. The second measure is marked with a tempo of 65 and a dynamic of *mf*. The third measure is marked with a tempo of 65 and a dynamic of *mf*. The fourth measure is marked with a tempo of 65 and a dynamic of *mf*. The piano part features a melody in the right hand and a bass line in the left hand, with a variety of note values and rests.

# 기호 음악 (Symbolic Music)

- 추상적이고 이산적인(discrete) 기호들로 표현한 음악
- 연주 표현법과 악보 표현법이 여기에 속합니다.
- 사람이 이해할 수 있는 음악적 특징을 컴퓨터로 추출하고 다루기 용이합니다.
  - 음 높이, 길이, 박자, 빠르기, 화음, 조성, 구조 등
- 대표적인 기호 음악으로 MIDI 음악이 있습니다.
  - MIDI는 연주 표현법에 해당합니다.

# MIDI란?

- Musical Instrument Digital Interface(악기 디지털 인터페이스)의 줄임말
- 전자 악기끼리 주고 받는 디지털 신호 체계를 표준화한 일종의 규약
- 응용
  - 악기의 제어
    - 하나의 악기로 다른 악기를 제어할 수 있습니다.
    - 컴퓨터, 전자 키보드, 신시사이저(Synthesizer) 등으로 연주합니다.
  - 합성
    - 컴퓨터 소프트웨어로 만들어낼 수 있습니다.
    - 시퀀서(Sequencer), 악보 작성 프로그램(Scorewriter) 등으로 작업합니다.
- <https://ko.wikipedia.org/wiki/MIDI>

# MIDI 재생 디바이스와 사운드

- MIDI 파일(.mid, .midi)은 연주 정보를 담고 있습니다.
  - 악기 명령들의 모임이지, 실제 오디오의 녹음물이 아닙니다.
  - 다른 사운드 파일보다 용량이 매우 작습니다.
- MIDI는 통신 규약에 불과하기 때문에,
  - 자체적으로 사운드를 가지고 있지 않습니다.
  - 따라서 재생하는 디바이스의 사운드에 의존하여 음악을 재생하게 됩니다.
- “비극의 선율.mid”(2016)의 일부분을 서로 다른 디바이스에서 재생하였습니다.
  - 왼쪽: Android 휴대폰에서 ‘Samsung Music’ 앱으로 재생
  - 오른쪽: Windows PC에서 ‘Windows Media Player’로 재생



# MIDI 스펙

- MIDI는 1.0과 2.0이 있습니다.
- 이 강의에서 다루는 MIDI 스펙은 1.0 스펙입니다.
- 2023년에 MIDI 2.0이 출범하였습니다.
  - MIDI 1.0을 기반으로 두면서 여기에 몇 가지 기능이 추가되었습니다.
  - MIDI 기기 간 양방향 소통에 있어서 확장된 기능들을 제공한다고 합니다.
  - 이 강의에서는 다루지 않습니다.
- <https://midi.org/specs>

# Message 타입

- MIDI 파일에서 사용되는 명령(Message)들은 크게 5가지 그룹으로 나뉩니다.
  - Channel Voice Messages
    - 각 채널 별로 음표를 켜거나, 끄거나, 또는 Controller(NWC의 MPC)를 바꿀 때 사용됩니다.
  - Channel Mode Messages
    - 모든 소리를 끄거나, 모든 Controller를 초기화하는 등의 특별한 명령을 제공합니다.
  - System Exclusive Messages
    - 디바이스마다 다른 기능을 수행하는 명령을 사용합니다.
  - System Common Messages
  - System Real-Time Messages
    - 시퀀스(곡)의 시작, 정지, 이어하기, 초기화 명령을 제공합니다.



# Message 타입

- 각 message들은 3바이트 이내의 binary로 표현됩니다.
  - Message마다 필요한 바이트 수가 다릅니다.
- 가장 많이 사용되는 message은 Note on과 Note off입니다.
  - 이들은 ChannelVoice Messages에 포함됩니다.
- Note on
  - 음표가 눌리기 시작할 때 보내는 message입니다.
- Note off
  - 음표를 떼기 시작할 때 보내는 message입니다.

# Meta Message 타입

- 곡의 메타 정보를 갖고 있는 message 타입도 있습니다.
  - text
  - track\_name
  - instrument\_name
  - lyrics
  - end\_of\_track
  - set\_tempo
  - time\_signature
  - key\_signature
  - 등등...
- [https://mido.readthedocs.io/en/latest/meta\\_message\\_types.html](https://mido.readthedocs.io/en/latest/meta_message_types.html)

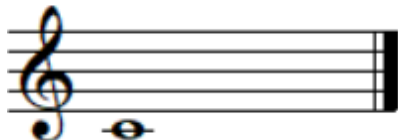
# Channel, Key, Velocity

- Message들의 binary 인코딩을 살펴보기 전에 알아야 할 것들이 있습니다.

- **Channel**

- 0 ~ 15 사이의 값을 가지며, MIDI 채널 번호 1 ~ 16에 대응됩니다.
- 각 채널은 서로 다른 Staff에 해당한다고 보면 됩니다.
  - 타악기 전용 Staff는 Channel 값이 9입니다.
- (3주차 발표 자료의 58페이지 참조)

- **Key**

- 0 ~ 127 사이의 값을 가지며, 음 높이를 의미합니다.
-  왼쪽 음(C<sub>4</sub>)의 값은 60(0x3C)입니다.

- 반음 높아질 때마다 값이 1씩 오르고, 반음 낮아질 때마다 값이 1씩 내려갑니다.

# Key 값(10진수)과 실제 음

C	C#/D♭	D	D#/E♭	E	F	F#/G♭	G	G#/A♭	A	A#/B♭	B
0	1	2	3	4	5	6	7	8	9	10	11
12	13	14	15	16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31	32	33	34	35
36	37	38	39	40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55	56	57	58	59
60	61	62	63	64	65	66	67	68	69	70	71
72	73	74	75	76	77	78	79	80	81	82	83
84	85	86	87	88	89	90	91	92	93	94	95
96	97	98	99	100	101	102	103	104	105	106	107
108	109	110	111	112	113	114	115	116	117	118	119
120	121	122	123	124	125	126	127				

# Channel, Key, Velocity

- **Velocity**
  - Message마다 의미가 다르지만, 보통 “어느 정도로” 할 지를 결정하는 값입니다.
  - 0 ~ 127 사이의 값을 가지며, 기본값은 64입니다.
  - Note on, Note off에서 Velocity는 얼마나 세게 음표를 누르고 뗄 지 결정하는 값입니다.
- [https://mido.readthedocs.io/en/latest/about\\_midi.html](https://mido.readthedocs.io/en/latest/about_midi.html)
- <https://midi.org/summary-of-midi-1-o-messages>

# Note On Event

- **Note on**
  - 음표가 눌리기 시작할 때 보내는 message입니다.
  - Binary로 "1001nnnn 0kkkkkkk 0vvvvvvv"입니다.
    - nnnn(0 ~ 15): Channel (MIDI 채널 번호 1 ~ 16)
    - kkkkkkk(0~127): Key (음 높이 번호)
    - vvvvvvv(0~127): Velocity (얼마나 세게 음표를 누를지)
- 예) Hexadecimal로 "92 3C 40"
  - **Note on**, Channel 2에서 60번째 음(C<sub>4</sub>)을 64의 Velocity로 입력

# Note Off Event

- **Note off**
  - 음표를 떼기 시작할 때 보내는 message입니다.
  - Binary로 "1000nnnn 0kkkkkkk 0vvvvvvv"입니다.
    - nnnn(0 ~ 15): Channel (MIDI 채널 번호 1 ~ 16)
    - kkkkkkk(0~127): Key (음 높이 번호)
    - vvvvvvv(0~127): Velocity (얼마나 세게 음표를 뺄지)
- 예) Hexadecimal로 "82 3C 40"
  - **Note off**, Channel 2에서 연주하던 60번째 음(C4)을 64의 Velocity로 멈추기

# Ticks Per Beat

- 모든 Note on, Note off 이벤트가 동시에 일어난다면 음악이 아니게 되겠죠?
- 각 이벤트에는 타이밍 값이 있습니다.
  - 타이밍을 설명하기 위해 **박자 당 틱 수**(ticks per beat)와 **빠르기**(tempo)를 소개합니다.
- MIDI 파일마다 헤더에 **ticks per beat**가 정의되어 있습니다.
  - Tick은 일정한 시간 간격으로 반복하여 발생하는 이벤트입니다.
  - 4분음표 한 번의 길이를 나타내기 위해 몇 번의 tick을 발생시킬 것인지 정의합니다.
  - NWC로 만든 곡들은 ticks per beat가 **192**입니다.
  - Ticks per beat 값은 주로 3의 배수이면서 2의 거듭제곱의 배수인 수로 정합니다.
  - 한 번 설정되면 곡 내에서 변하지 않습니다.



# Tempo

- **빠르기**(tempo)는 **set tempo meta message**를 보내어 정해집니다.
  - 여러분이 생각하시는 그 빠르기(예: ♩=120)를 바꾸는 message입니다.
  - 1분에 4분음표 120개를 연주하는 기본 빠르기의 tempo 값은 500,000입니다.
  - Tempo 값이 클수록 실제 빠르기는 느려집니다.
- 분당 4분음표 수(Beats per minute; BPM) 계산법
  - $BPM = 60,000,000 \div tempo$
  - BPM을 계산할 때 tempo를 알고 있다면 ticks per beat는 몰라도 됩니다.
- <https://ssomu.tistory.com/entry/%EC%BB%B4%ED%93%A8%ED%84%B0%EC%9D%8C%EC%95%85-Computer-Music-MIDI-ticks-2>

# 타이밍

- 각 이벤트에는 시간을 얼마나 쉴 것인지에 대한 값(timing)이 들어갑니다.
  - 이 타이밍 값의 단위는 tick입니다.
  - 바로 직전 message의 이벤트가 일어나고 몇 tick 후에 이 이벤트를 실행할 것인지 나타냅니다. 즉, delta tick입니다.
  - 이 타이밍은 곡의 빠르기(tempo)와 박자 당 틱 수(ticks per beat)의 영향을 받습니다.
- 동시에 일어나는 두 이벤트 사이의 타이밍(delta tick) 값은 0입니다.
  - 예) 화음 입력 / 서로 다른 channel에서 동시에 울리는 음
- 타이밍 값은 앞의 이벤트에 대해 상대적입니다.
  - 즉, 타이밍은 곡 전체에서의 절대적인 시간적 위치를 지정하는 것이 아닙니다.

# 타이밍

- 현재 이벤트를 포함하여 그 앞에 있었던 모든 MIDI 이벤트의 timing(delta tick) 값을 누적해야 현재 이벤트가 실제로 재생되는 시각을 알 수 있습니다.
- 그러나 set tempo message를 통해 언제든지 tempo가 바뀔 수 있습니다.
- 따라서 단순히 모든 이벤트의 timing 값을 더해 현재 tempo에 대해 계산하면 계산한 재생 시각과 실제 재생 시각에 차이가 발생할 것입니다.



빨간색 음표의 실제 재생 시각을 구할 때, 단순히 앞에 있는 모든 음표의 timing 값을 더하고 BPM 135 기준으로 계산하면 실제 재생 시각과 어긋납니다.

BPM 120일 때에 시간이 느리게 흘렀기 때문입니다.

# 타이밍

- 이벤트의 타이밍을 실제 재생 시각(초)으로 환산하는 방법
  1. MIDI 파일의 **ticks per beat** 값을 기억합니다.
  2. MIDI의 맨 앞에서부터 순서대로 이벤트를 순회(iterate)합니다.
  3. 현재 보고 있는 이벤트가 있기 전에 마지막으로 있었던 set tempo 이벤트의 tempo 값을 기억합니다.  
(이를 **current tempo**라고 합시다.)
  4. 현재 보고 있는 이벤트에 대해 타이밍(**delta tick**) 값을 확인합니다.
  5. 바로 직전 이벤트로부터 현재 이벤트가 몇 초 후에 재생될지를 다음과 같이 구합니다.
    - $\text{Delta time} = \text{delta tick} \times \text{current tempo} \div \text{ticks per beat}$
  6. 이전까지의 모든 이벤트가 재생된 후의 시각을 기록하는 **global time** 값에 이 **delta time** 값을 누적합니다.
    - $\text{Global time} += \text{delta time}$
  7. 현재 이벤트의 실행 시작 시각은 재생 시작으로부터 **global time** 초 후입니다. (값은 복사하여 저장)
  8. 곡이 끝날 때까지(end of track meta message를 만날 때까지) 순회를 돌며 다음 이벤트에 대해 3.부터 7. 사이의 과정을 반복합니다.

# 음표를 MIDI로 구현

- 음표 하나를 여러 가지 방법으로 MIDI로 구현할 수 있습니다.
  - NWC에서 4분음표 C<sub>4</sub> 음 하나를 재생하면 "00 90 3C 6E 81 20 90 3C 00 20" (TPB = 192)
    - 00: 0의 tick 동안 쉬기 (곡의 첫 번째 음표라서 곡이 시작하자마자 다음 note on 실행)
    - 90 3C 6E: Note on, 1번째 channel의 C<sub>4</sub> 음의 velocity를 110(0x6E)으로 입력
    - 81 20: 288(0x120)의 tick 동안 쉬기
    - 90 3C 00: Note on, 1번째 channel의 C<sub>4</sub> 음의 velocity를 0으로 입력 (Note off의 기능을 함)
    - 20: 32(0x20)의 tick 동안 쉬기 (뒤에 다음 음표가 옴)
  - "작곡 도우미"(7주차 실습 프로그램)에서는 "00 90 3C 7F 15 80 3C 7F 04" (TPB = 25)
    - 00: 0의 tick 동안 쉬기 (곡의 첫 번째 음표라서 곡이 시작하자마자 다음 note on 실행)
    - 90 3C 7F: Note on, 1번째 channel의 C<sub>4</sub> 음의 velocity를 127(0x7F)로 입력
    - 15: 21(0x15)의 tick 동안 쉬기
    - 80 3C 7F: Note off, 1번째 channel의 C<sub>4</sub> 음을 velocity 127(0x7F)로 멈추기
    - 04: 4(0x04)의 tick 동안 쉬기

# 음표와 타이밍

- 앞으로 “작곡 도우미”의 타이밍 단위로 설명합니다.
- “작곡 도우미”에서는 꼭 찬 4분음표 길이(ticks per beat)를 25로 설정하였습니다.
- 음표에 이음줄이 붙어 있지 않다면,
  - 각 음표는 꼭 찬 길이로 연주되지 않고
  - **어느 정도 길이로 연주되다가 중간에 멈춰야 합니다.**
  - “작곡 도우미”에서는 그 길이를 꼭 찬 박자의 6/7 지점으로 잡았습니다.
  - 예) “작곡 도우미”에서 4분음표 2개(C<sub>4</sub>, D<sub>4</sub>)를 연속으로 놓을 때
    - “00 90 3C 7F 15 80 3C 7F 04 90 3E 7F 15 80 3E 7F”
    - 21(0x15)는  $25 \times (6 \div 7)$ 을 반올림한 값이며, 04(0x04)는  $25 - 21$ 로 계산된 값입니다.
- 곡의 맨 마지막 음표 뒤에는 타이밍 값을 붙이지 않습니다.

# 쉼표

- 쉼표는 따로 message를 보내지 않고,
  - 쉬어야 하는 길이만큼 타이밍 값을 갖습니다.
- 즉, 쉼표는 음표를 재생하지 않고 기다리는 것과 같습니다.
- NWC로 만든 MIDI 곡의 맨 마지막에 쉼표가 놓여 있다면
  - MIDI 파일에는 이 쉼표가 반영되지 않습니다.
  - 곡의 맨 끝에는 타이밍 값을 쓰지 않기 때문입니다.
- NWC로 만든 MIDI 곡에 음표 없이 쉼표만 놓여 있다면
  - MIDI 파일에는 곡의 시작과 끝 외에 아무 것도 기록되지 않습니다.
  - 이 파일의 재생 길이는 0초입니다.
    - 일부 미디어 플레이어에서 재생하려고 하면 오류가 납니다.

# MIDI Binary 파일 해부

- “작곡 도우미”로 만든 왼쪽 악보를 MIDI로 저장한 후 binary를 뜯어 보면 아래와 같습니다.

```
4D 54 68 64 00 00 00 06 00 01 00 01 00 18 4D 54
72 6B 00 00 00 38 00 91 30 7F 00 92 30 7F 00 34
7F 00 37 7F 15 81 30 7F 04 91 37 7F 15 81 37 7F
04 91 34 7F 15 81 34 7F 04 91 37 7F 0B 82 37 7F
00 34 7F 00 30 7F 0A 81 37 7F 01 FF 2F 00
```

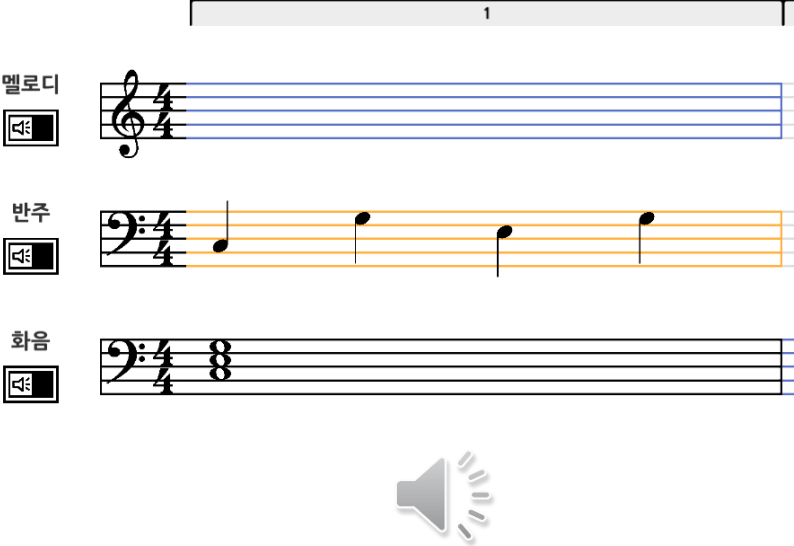
```
MThd - r r ↑MTrk 8 ?0 ?0 40 70┐?0┐?0┐?0┐?0┐?0?0 40 00?0r /
```

- 반주 Channel 값은 1, 화음 Channel 값은 2입니다.
- 색깔별로 타이밍, Note on, Note off입니다.

멜로디

반주

화음





# 라이브러리의 필요성

- 앞에서 MIDI 파일을 binary로 해부해 보았습니다.
  - 그러나 우리가 MIDI 파일을 생성하는 프로그램을 짜야 한다면
  - MIDI Message들의 binary 코드를 모두 외워서 짤 수는 없을 겁니다.
- 그래서 MIDI 프로그래밍을 할 때 유용한 라이브러리들이 이미 나와 있습니다.
  - 여기서는 Python 라이브러리 하나와
  - C# 라이브러리 두 개를 중점적으로 다뤄보도록 하겠습니다.

# MIDI 프로그래밍

---

Python 라이브러리

# Python MIDI 라이브러리

- Mido
  - Python(2 또는 3)에서 MIDI Message와 Port를 작업할 수 있는 라이브러리
    - MIDI Message를 생성하거나([문서](#))
    - Message들을 특정 Port로 출력하거나([문서](#))
    - MIDI 파일을 열거나 저장할 수 있습니다.([문서](#))
  - 설치법 및 사용법을 포함한 문서화가 잘 되어 있습니다.
  - <https://mido.readthedocs.io/en/latest/index.html>

# Mido 설치

- 먼저 Python 가상 환경(venv)을 만들고 활성화합니다.
  - (운영체제 공통) 작업할 경로에서 midi-seminar 폴더를 새로 생성합니다.
  - (운영체제 공통) 터미널을 실행합니다.
  - (운영체제 공통) cd 명령을 통해 midi-seminar 폴더로 이동합니다.
  - (운영체제 공통) `python -m venv .venv`
  - (Windows PowerShell) `Set-ExecutionPolicy -ExecutionPolicy Unrestricted -Scope CurrentUser`
  - (Windows PowerShell) `.venv\Scripts\Activate.ps1`
  - (macOS / Linux) `source .venv/bin/activate`
- 처음 설정할 때에는 연두색 글씨까지 모두 입력하고, 두 번째 사용할 때부터는 검은색 글씨만 입력합니다.

# Mido 설치

- 가상 환경을 활성화한 상태에서 아래 명령을 실행하면 설치가 완료됩니다.

```
python3 -m pip install mido  
Python3 -m pip install python-rtmidi
```

- 가상 환경을 종료하려면 `deactivate` 를 입력합니다.
- Mido를 사용한 Python 코드를 실행할 때에는 항상 앞 슬라이드의 가상 환경을 활성화한 상태에서 실행해 주세요.

# 무엇을 해볼 것인가?

1. 간단한 MIDI 파일 생성
2. MIDI 파일 재생
3. MIDI messages 출력
4. MIDI messages를 음표 단위로 변환

# Mido 예제

- Git을 사용한다면, 아래의 명령으로 저장소를 clone받을 수 있습니다.
  - `git clone https://github.com/mido/mido.git`
- Git을 사용하지 않으면,
  - <https://github.com/mido/mido/>에서 직접 다운받을 수 있습니다.
- mido/examples/midifiles 폴더에서 여러 예제를 찾을 수 있습니다.
  - create\_midi\_file.py
  - play\_midi\_file.py
  - print\_midi\_file.py

# 간단한 MIDI 생성

- create\_midi\_file.py
  - 19 ~ 22줄: 파일 출력 준비
  - 28 ~ 34줄: 4번 반복
  - 29줄: E4, B4, E5 중 음 랜덤 선택
  - 30줄: **delta**의 tick만큼 쉬고 **음표 입력**
  - 31 ~ 33줄
    - 총 **delta**의 tick 동안,  
**ticks\_per\_expr**의  
tick 간격마다 **pitch bend 적용**
  - 34줄: 바로 **음표 떼기**
  - 36줄: **test.mid**로 저장

```
1  #!/usr/bin/env python3
2
3  # SPDX-FileCopyrightText: 2013 Ole Martin Bjorndalen <ombdalen@gmail.com>
4  #
5  # SPDX-License-Identifier: MIT
6
7  """
8  Create a new MIDI file with some random notes.
9
10 The file is saved to test.mid.
11 """
12 import random
13 import sys
14
15 from mido import MAX_PITCHWHEEL, Message, MidiFile, MidiTrack
16
17 notes = [64, 64 + 7, 64 + 12]
18
19 outfile = MidiFile()
20
21 track = MidiTrack()
22 outfile.tracks.append(track)
23
24 track.append(Message('program_change', program=12))
25
26 delta = 300
27 ticks_per_expr = int(sys.argv[1]) if len(sys.argv) > 1 else 20
28 for i in range(4):
29     note = random.choice(notes)
30     track.append(Message('note_on', note=note, velocity=100, time=delta))
31     for j in range(delta // ticks_per_expr):
32         pitch = MAX_PITCHWHEEL * j * ticks_per_expr // delta
33         track.append(Message('pitchwheel', pitch=pitch, time=ticks_per_expr))
34     track.append(Message('note_off', note=note, velocity=100, time=0))
35
36 outfile.save('test.mid')
```



# MIDI 재생

- play\_midi\_file.py
  - 20 ~ 24줄
    - 첫 번째 인자로 재생할 파일 이름,  
두 번째 인자로 출력할 포트 지정
    - 예) ./play\_midi\_file.py test.mid
  - 26 ~ 38줄: 출력 포트 열어서 사용
  - 28줄: MIDI 파일 열기
  - 29줄: 재생 시작 시간 측정
  - 30 ~ 32줄: 재생하면서 message 실행
  - 33 ~ 34줄: 총 재생 시간 출력
  - 36 ~ 38줄: 예외 처리

```
1  #!/usr/bin/env python3
2
3  # SPDX-FileCopyrightText: 2013 Ole Martin Bjorndalen <ombdalen@gmail.com>
4  #
5  # SPDX-License-Identifier: MIT
6
7  """
8  Play MIDI file on output port.
9
10 Run with (for example):
11 |
12 |     ./play_midi_file.py 'SH-201 MIDI 1' 'test.mid'
13 |
14 """
15 import sys
16 import time
17
18 import mido
19 from mido import MidiFile
20
21 filename = sys.argv[1]
22 if len(sys.argv) == 3:
23     portname = sys.argv[2]
24 else:
25     portname = None
26
27 with mido.open_output(portname) as output:
28     try:
29         midifile = MidiFile(filename)
30         t0 = time.time()
31         for message in midifile.play():
32             print(message)
33             output.send(message)
34         print('play time: {:.2f} s (expected {:.2f})'.format(
35             time.time() - t0, midifile.length))
36     except KeyboardInterrupt:
37         print()
38         output.reset()
```

# MIDI messages 출력

- print\_midi\_file.py
  - 17줄: 입력 인자로 파일 이름 지정
    - 예) ./print\_midi\_file.py "test.mid"
  - 19줄: MIDI 파일 열기
  - 21 ~ 24줄: 각 track에 대해
  - 23 ~ 24줄: Message 출력
- play\_midi\_file.py와 달리 track을 구분하며 meta message들도 출력됩니다.
- 기존 MIDI 파일을 분석할 때 유용합니다.

```
1  #!/usr/bin/env python3
2
3  # SPDX-FileCopyrightText: 2013 Ole Martin Bjorndalen <ombdalen@gmail.com>
4  #
5  # SPDX-License-Identifier: MIT
6
7  """
8  Open a MIDI file and print every message in every track.
9
10 Support for MIDI files is still experimental.
11 """
12 import sys
13
14 from mido import MidiFile
15
16 if __name__ == '__main__':
17     filename = sys.argv[1]
18
19     midi_file = MidiFile(filename)
20
21     for i, track in enumerate(midi_file.tracks):
22         sys.stdout.write(f'=== Track {i}\n')
23         for message in track:
24             sys.stdout.write(f' {message!r}\n')
```

# MIDI messages 출력 결과

```
=== Track 0
MetaMessage('text', text='By <Name>', time=0)
MetaMessage('copyright', text='Copyright "I 2015 <Name>', time=0)
MetaMessage('copyright', text='All Rights Reserved', time=0)
MetaMessage('text', text='Generated by NoteWorthy Composer', time=0)
MetaMessage('time_signature', numerator=4, denominator=4, clocks_per_click=24, notated_32nd_notes_per_beat=8, time=0)
MetaMessage('end_of_track', time=0)
=== Track 1
MetaMessage('midi_port', port=0, time=0)
MetaMessage('track_name', name='Staff', time=0)
Message('control_change', channel=0, control=7, value=127, time=0)
Message('control_change', channel=0, control=10, value=64, time=0)
Message('program_change', channel=0, program=2, time=0)
Message('note_on', channel=0, note=64, velocity=60, time=0)
Message('note_on', channel=0, note=64, velocity=0, time=80)
Message('note_on', channel=0, note=60, velocity=60, time=16)
Message('note_on', channel=0, note=60, velocity=0, time=80)
Message('note_on', channel=0, note=55, velocity=75, time=16)
Message('note_on', channel=0, note=55, velocity=0, time=24)
Message('note_on', channel=0, note=60, velocity=75, time=0)
Message('note_on', channel=0, note=60, velocity=0, time=24)
Message('note_on', channel=0, note=62, velocity=75, time=0)
Message('note_on', channel=0, note=62, velocity=0, time=24)
Message('note_on', channel=0, note=64, velocity=75, time=0)
Message('note_on', channel=0, note=64, velocity=0, time=24)
Message('note_on', channel=0, note=65, velocity=75, time=0)
Message('note_on', channel=0, note=65, velocity=0, time=24)
Message('note_on', channel=0, note=67, velocity=75, time=0)
Message('note_on', channel=0, note=67, velocity=0, time=24)
Message('note_on', channel=0, note=69, velocity=75, time=0)
Message('note_on', channel=0, note=69, velocity=0, time=24)
```

어디서부터 어디까지가 하나의 음표인지, 각 음표가 언제 재생되는지 한눈에 보기 어렵습니다.

# 음표 단위로 보고 싶다면?

- 같은 음표를 나타내는 Note on 과 note off 를 찾아 서로 묶어야 합니다.
- 어떻게 프로그램을 짜야 할지 감이 잡히시나요?
- 9주차 실습 자료의 “midi-seminar” 폴더에 실습 코드를 넣어두었습니다.
  1. “input” 폴더에 분석하고 싶은 MIDI 파일을 넣으세요.
  2. 터미널을 실행하고 “midi-seminar” 폴더로 이동하세요.
  3. Mido를 사용할 수 있도록 가상 환경을 활성화해주세요. ([36번째 슬라이드 참고](#))
  4. `python midi_to_json.py input`
  5. “output” 폴더에 생성된 JSON 파일을 열어보세요.
    - 곡에 사용된 모든 음표들의 정보가 저장되어 있습니다.

# midi\_to\_json.py 실행 결과

- 음표들은 음표 재생 시작 위치가 빠른 순으로 정렬됩니다.
  - ID: 음표 번호
  - Start\_timing: 음표 재생 시작 시각 (1 / 100만 초 단위)
  - End\_timing: 음표 재생 끝 시각 (1 / 100만 초 단위)
  - Channel: MIDI 채널 (9인 경우 타악기)
  - Note\_position: 음 높이 (60이 C4)
  - Note\_velocity: 음 세기 (0 ~ 127)
  - Start\_seq\_index: 음표 시작 위치의 마디 번호 (0부터 시작, 4/4 박 기준)
  - Start\_note\_index: 음표 시작 위치의 마디 내 위치 (0 ~ 15, 16분음표 단위)
  - End\_seq\_index: 음표 끝 위치의 마디 번호
  - End\_note\_index: 음표 끝 위치의 마디 내 위치
  - Note\_duration\_units: 음표 길이 (16분음표 단위)
    - 이음줄이 없는 음표의 경우 악보에 표기된 길이보다 짧게 연주함

```
1  [
2  {
3      "ID": 0,
4      "Start_tick": 0,
5      "End_tick": 80,
6      "Start_timing": 0,
7      "End_timing": 208333,
8      "Channel": 0,
9      "Note_position": 64,
10     "Note_velocity": 60,
11     "Note_pitch_class": "E",
12     "Note_octave": 4,
13     "Start_seq_index": 0,
14     "Start_note_index": 0,
15     "End_seq_index": 0,
16     "End_note_index": 1,
17     "Note_duration_units": 1
18 },
19 {
20     "ID": 1,
21     "Start_tick": 0,
22     "End_tick": 160,
23     "Start_timing": 0,
24     "End_timing": 416667,
25     "Channel": 2,
26     "Note_position": 43,
27     "Note_velocity": 92,
28     "Note_pitch_class": "G",
29     "Note_octave": 2,
30     "Start_seq_index": 0,
31     "Start_note_index": 0,
32     "End_seq_index": 0,
33     "End_note_index": 3,
34     "Note_duration_units": 3
35 },
36 {
37     "ID": 2,
```

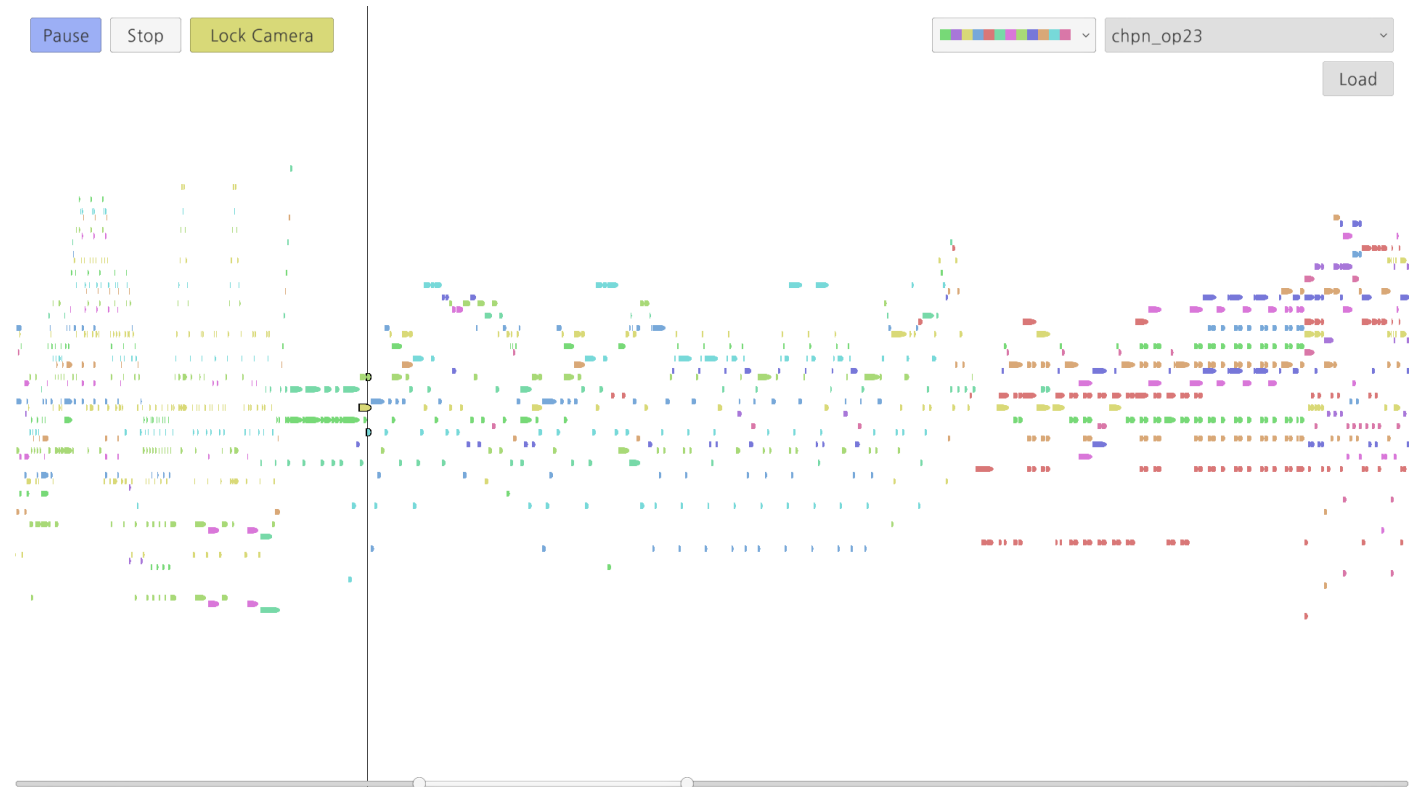
# 이를 이용해 무엇을 할 수 있나요?

- <https://github.com/salt26/chromatic-piano-roll>

- Piano Roll 표현법의 형태로 음악을 시각화할 수 있습니다.

- 음악의 구조를 한눈에 보기에 편리합니다.

- 이 프로그램에서는 음이름마다 5도권에 기반한 고유한 색상을 붙여 곡의 화성적 구조를 시각적으로 쉽게 파악할 수 있습니다.



# MIDI 프로그래밍

---

C# 라이브러리

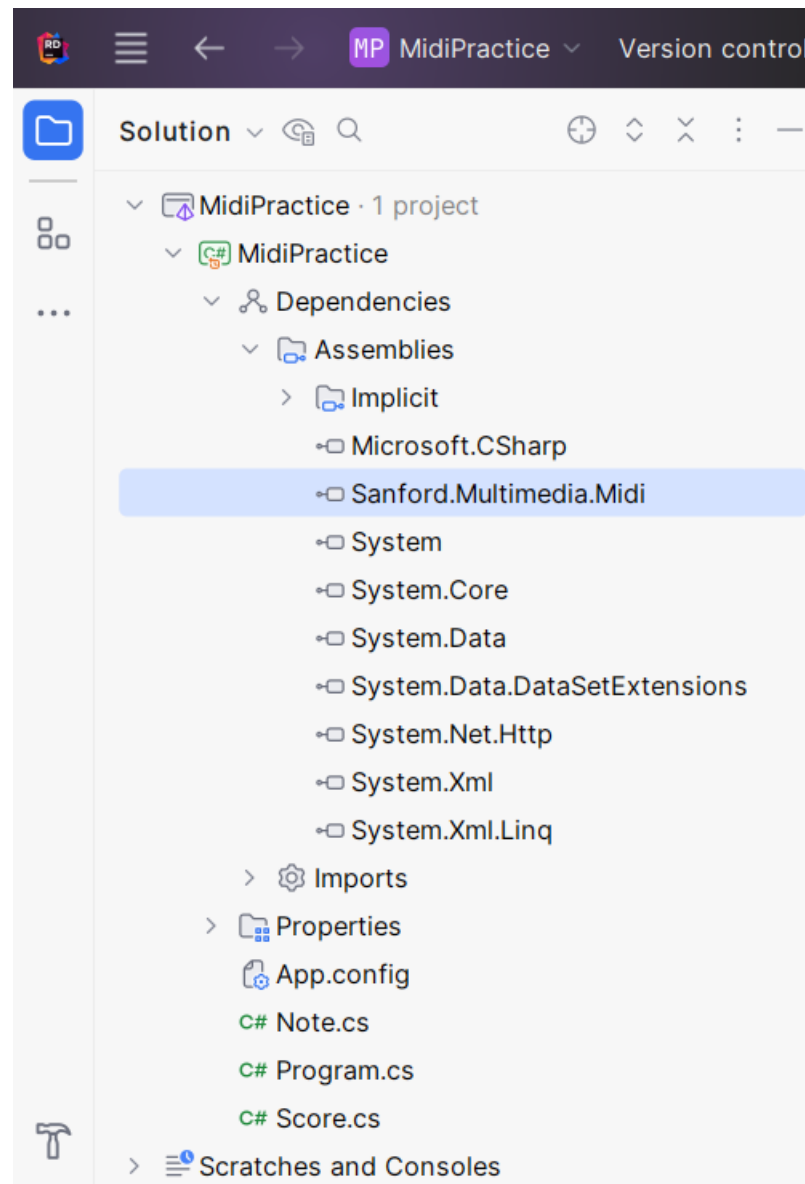
# C# MIDI 라이브러리

- Sanford.Multimedia.Midi
  - C#에서 사용할 수 있는 MIDI 라이브러리
  - 소스 코드(데모 포함)
    - <https://github.com/tebjan/Sanford.Multimedia.Midi>
  - 설명 및 다운로드
    - <https://www.codeproject.com/Articles/6228/C-MIDI-Toolkit>
- 이번 세미나 실습 자료에서 제공하는 “Sanford.Multimedia.Midi.dll” 파일은
  - Windows Forms 관련 코드를 제거하여
  - Unity와 호환되도록 만든 라이브러리 파일입니다.
  - 이 파일을 C# 프로젝트 또는 Unity 프로젝트에 포함하여 사용할 수 있습니다.
  - 단점이 있다면, Windows에서만 사용 가능합니다.



# 예제 코드 분석 실습

- “MidiPractice-CSharp.zip” 파일을 받아서
  - 압축을 해제하고
  - Rider로 .sln 파일을 열어보세요.
- 3개의 C# class 파일(.cs)이 있습니다.
  - Note.cs
  - Score.cs
  - Program.cs
- 참조에 “Sanford.Multimedia.Midi”가 포함됩니다.



# 예제 코드 분석 실습

- Note.cs
  - 하나의 음표를 나타내는 클래스입니다.
  - **Private 필드(Field)**
    - pitch: 음 높이
    - rhythm: 음표 길이
    - measure: 음표가 위치할 마디 번호
    - position: 음표의 마디 내 위치
    - staff: 음표가 놓일 Staff 번호
  - **생성자(Constructor)**
    - Note(pitch, rhythm, measure, position, [staff=0]): 음표 생성
  - **Public 메서드(Method)**
    - ToMidi(): 음표를 재생할 때 필요한 Note on, Note off message의 정보를 담은 Pair 생성

C# Note.cs X

```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6
7 namespace MidiPractice
8 {
9     /// <summary>
10    /// 음표 클래스입니다.
11    /// </summary>
12    [11 usages]
13    class Note
14    {
15        /// <summary>
16        /// 음 높이(0 ~ 127).
17        /// 예) 60: C4 / 64: E4 / 67: G4 / 72: C5
18        /// </summary>
19        private int pitch;
20
21        /// <summary>
22        /// 음표의 길이(1 이상). 4/4박에서 한 마디를 16등분한 길이를 기준으로 합니다.
23        /// 예) 16: 온음표 / 4: 4분음표 / 1: 16분음표
24        /// </summary>
25        private int rhythm;
```

# 예제 코드 분석 실습

- Score.cs
  - 음표들을 담는 악보를 나타내는 클래스입니다.
- Private 필드(Field)
  - score: 음표들의 리스트
  - isPlaying: 악보를 재생 중인 동안 true가 됩니다.
- 프로퍼티(Property)
  - IsPlaying
    - isPlaying 필드의 읽기 전용 프로퍼티
    - Score 외부의 다른 클래스에서 사용할 수 있습니다.

C# Score.cs X

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6  using Sanford.Multimedia.Midi;
7
8  namespace MidiPractice
9  {
10     /// <summary>
11     /// 음표들을 담고 있는 악보 클래스입니다.
12     /// </summary>
13     [2 usages]
14     class Score
15     {
16         /// <summary>
17         /// 음표들을 담을 리스트.
18         /// </summary>
19         private List<Note> score = new List<Note>();
20
21         /// <summary>
22         /// 현재 재생 중이면 true가 됩니다.
23         /// </summary>
24         private bool isPlaying = false;
25
26         /// <summary>
27         /// 현재 재생 중이면 true를 반환합니다. 읽기 전용입니다.
28         /// </summary>
```

# 예제 코드 분석 실습

- Score.cs

- Public 메서드(Method)

- AddNote(pitch, rhythm, measure, position, [staff=0])

- 음표를 생성하여 추가합니다.

- AddNote(note)

- 음표를 추가합니다. (오버로딩)

- RemoveNote(note)

- 특정 음표를 악보에서 제거합니다.

- Save([name="Sample"])

- 악보를 MIDI 파일로 저장합니다.

- Play(outputDevice)

- 악보를 재생합니다. (한 번에 하나만 재생 가능)

```
C# Score.cs x
66 }
67
68 /// <summary>
69 /// 악보를 Midi 파일로 저장합니다.
70 /// </summary>
71 /// <param name="name">저장할 파일 이름(확장자 제외)</param>
72 > public void Save(string name = "Sample"){...}
98
99 /// <summary>
100 /// 악보를 재생합니다.
101 /// 이미 재생 중인 악보는 중복하여 재생할 수 없습니다.
102 /// </summary>
103 /// <param name="outputDevice">출력 디바이스</param>
104 /// (만약 Unity에서 작업할 경우, 타입을 void 대신 IEnumerator로 바꿔서 Coroutine으로 사용하세요.)
105 > public void Play(OutputDevice outputDevice){...}
141
142 /// <summary>
143 /// 음표들을 연주하기 위해 Midi message pair 리스트로 변환합니다.
144 /// (이 Pair들은 재생하거나 저장할 때 Message로 번역됩니다.)
145 /// </summary>
146 /// <returns></returns>
147 > List<KeyValuePair<float, int>> ToMidi(){...}
166 }
167 }
```

# 예제 코드 분석 실습

- Score.cs
  - Private 메서드(Method)
    - ToMidi()
      - 악보에 있는 음표들을 재생할 때 필요한 Note on, Note off message의 정보를 담은 pair 생성

```
C# Score.cs x
66 }
67
68 /// <summary>
69 /// 악보를 Midi 파일로 저장합니다.
70 /// </summary>
71 /// <param name="name">저장할 파일 이름(확장자 제외)</param>
72 > public void Save(string name = "Sample"){...}
73
74
75
76
77
78
79
80
81
82
83
84
85 > public void Play(OutputDevice outDevice){...}
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147 > List<KeyValuePair<float, int>> ToMidi(){...}
148
149
150
151
152
153
154
155
156
157 }
```

# 예제 코드 분석 실습

- Program.cs
  - Main 메서드를 포함한 클래스입니다.
  - 18줄: 출력 디바이스를 설정합니다.
  - 21줄: 악보를 생성합니다.
  - 24 ~ 37줄: 악보에 음표를 추가합니다.
  - 40줄: 악보를 파일로 저장합니다.
    - "MidiPractice/MidiPractice/bin/Debug/sample.mid"
  - 43줄: 악보를 재생합니다.
  - 47줄
    - 출력 디바이스를 닫습니다.
    - 이 코드가 빠지면 프로그램이 끝나지 않습니다!

C# Program.cs

```
13 class Program
14 {
15     static void Main(string[] args)
16     {
17         // 출력 디바이스 설정 (재생 시 필요)
18         OutputDevice outDevice = new OutputDevice(0);
19
20         // 음표들을 담은 악보 생성
21         Score score = new Score();
22
23         // 악보에 음표 추가
24         score.AddNote(new Note( pitch: 48, rhythm: 4, measure: 0, position: 0)); // 4분음표 C3
25         score.AddNote(new Note( pitch: 52, rhythm: 4, measure: 0, position: 4)); // 4분음표 E3
26         score.AddNote(new Note( pitch: 55, rhythm: 4, measure: 0, position: 8)); // 4분음표 G3
27         score.AddNote(new Note( pitch: 58, rhythm: 4, measure: 0, position: 12)); // 4분음표 Bb3
28         score.AddNote( pitch: 48, rhythm: 16, measure: 1, position: 0); // 온음표 C3
29         score.AddNote( pitch: 52, rhythm: 16, measure: 1, position: 0); // 온음표 E3 (화음)
30         score.AddNote( pitch: 55, rhythm: 16, measure: 1, position: 0); // 온음표 G3 (화음)
31         score.AddNote( pitch: 58, rhythm: 16, measure: 1, position: 0); // 온음표 Bb3 (화음)
32         score.AddNote( pitch: 60, rhythm: 16, measure: 1, position: 0); // 온음표 C4 (화음)
33         score.AddNote( pitch: 48, rhythm: 16, measure: 2, position: 0, staff: 9); // 타악기
34         score.AddNote( pitch: 52, rhythm: 16, measure: 2, position: 0, staff: 9); // 타악기 (화음)
35         score.AddNote( pitch: 55, rhythm: 16, measure: 2, position: 0, staff: 9); // 타악기 (화음)
36         score.AddNote( pitch: 58, rhythm: 16, measure: 2, position: 0, staff: 9); // 타악기 (화음)
37         score.AddNote( pitch: 60, rhythm: 16, measure: 2, position: 0, staff: 9); // 타악기 (화음)
38
39         // 악보를 "Sample.mid" 파일로 저장
40         score.Save();
41
42         // 악보 재생
43         score.Play(outDevice);
44
45         // 출력 디바이스 종료
46         // (프로그램이 끝날 때 반드시 적어주세요!)
47         outDevice.Close();
48     }
49 }
50 }
```

# 예제 코드 분석 실습

- MidiPractice 프로젝트를 실행해 보면서
  - 각 코드가 어떤 역할을 하는지 살펴보세요.
  - 키워드: ChannelMessage, ChannelCommand, OutputDevice, Sequence, Track
  - 이해가 되지 않는 코드가 있으면 질문하세요.
- 코드를 변형해 보세요.
  - 악보에 들어가는 음표를 바꾸거나 추가해 보세요.
  - 4/4박자 대신 3/4박자의 음악을 만들 수 있도록 변형해 보세요.
  - 16분음표보다 더 짧은 32분음표까지 사용할 수 있도록 변형해 보세요.
- 코드 응용
  - Unity 등을 활용하여, 악보를 편집할 수 있는 User Interface(UI)를 만들어 보세요.
  - 피아노 소리를 내는 악기 애플리케이션을 만들어 보세요.

# macOS에서 C# MIDI를 다룰 수 없나요?

- FluidSynth를 활용하면 다른 운영체제에서도 C#으로 MIDI 음악을 재생하는 프로그램을 만들 수 있습니다!
  - 또는 C++ 등의 다른 언어로 짜는 것도 가능합니다.
- 이는 10주차 때 다뤄보도록 하겠습니다.



다음 세미나 공지

---

# 다음 세미나는...

- 10주차 세미나는 원래 7월 25일(목)에 진행하려 했으나
  - 저만 참석을 누르고 여섯 분이 불참을 눌러 주셨습니다.
  - 세미나... 미뤄야겠죠?
- 8월 1일(목) 오후 7시 30분, 이때 가능하신가요?
  - 장소는 서울대학교 301동 203호가 될 것입니다.
- 처음 준비하려고 한 내용을 다 준비하지 못해서, 나머지는 다음 세미나 때 다루도록 하겠습니다.
  - 다음에 하는 10주차가 마지막 세미나입니다!!

# 10주차 세미나 미리보기

- 실습: C#에서 FluidSynth 사용하기
- 음악정보검색 맛보기
  - 멜로디 추출
  - 조성 인식
  - 화음 인식
  - 오디오 신호 처리 (진행하지 못할 수도 있음)
- 실습: Python으로 음악적 특징 추출하기
- 실습: 신청곡 두 곡 분석

감사합니다!

---