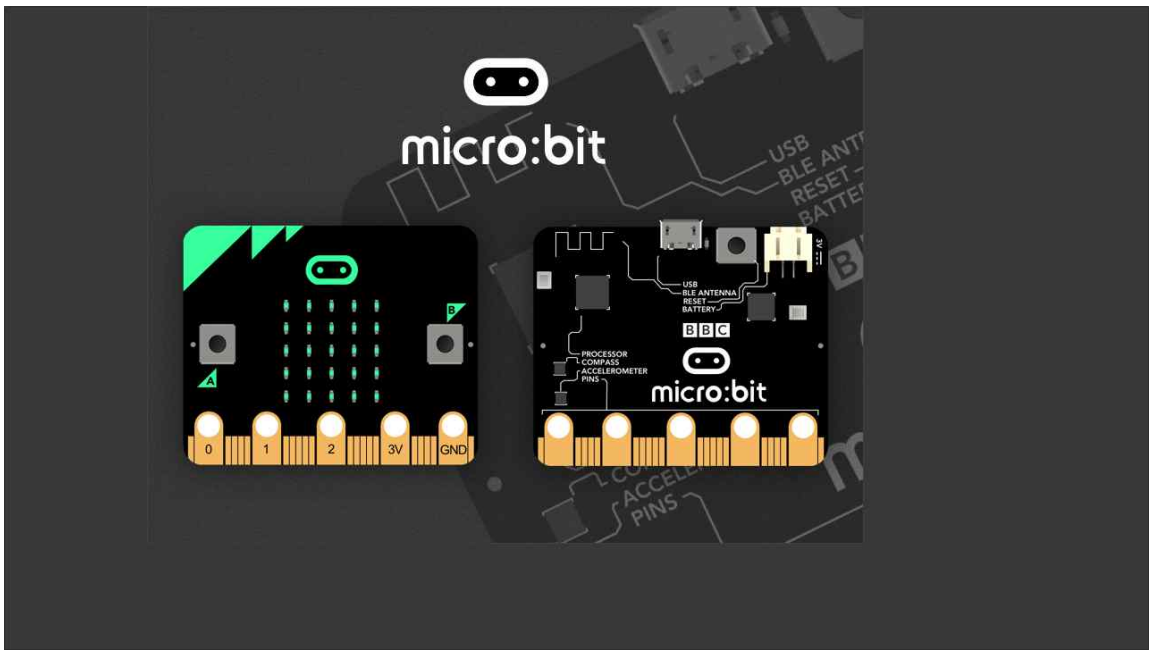
2025. CMOOC
마이크로비트로 실습하며 배우는 파이썬기초

충북교육연구정보원 CMOOC 강사 박정진

Part 01

마이크로비트 기초



피지컬 컴퓨팅은 댄 오설리번(Dan O'Sullivan)과 톰 아이고(Tom Igoe)가 처음 사용한 용어로 '물체의 움직임을 감지하는 센서와 이를 제어하는 소프트웨어를 이용하여 현실 세계와 상호 작용이 가능한 물리적 시스템을 만드는 것'을 의미합니다. 예를 들어 어두워지면 켜지는 가로등, 사람이 감지되면 열리는 자동문, 문이 열리면 울리는 사이렌 등이 이러한 피지컬컴퓨팅 사례에 속한다고 볼 수 있습니다.

마이크로비트는 기존에 피지컬컴퓨팅 교육용 교구로 많이 사용되고 있는 아두이노, 라즈베리 파이와 같은 SW 교육용 보드입니다. 외부 입출력을 할 수 있는 많은 단자들이 있으며, 자체 나침반 센서, 온도 센서, 가속도 센서 등을 내장하고 있어 추가적인 부품을 사용하지 않고 마이크로비트만 가지고 있어도 된다는 점이 장점입니다. 또한 통신을 위한 블루투스 4.0을 내장하고 있어 블루투스 모듈 없이 스마트폰 및 스마트 기기와 연동할 수도 있습니다.

1. 마이크로비트란?

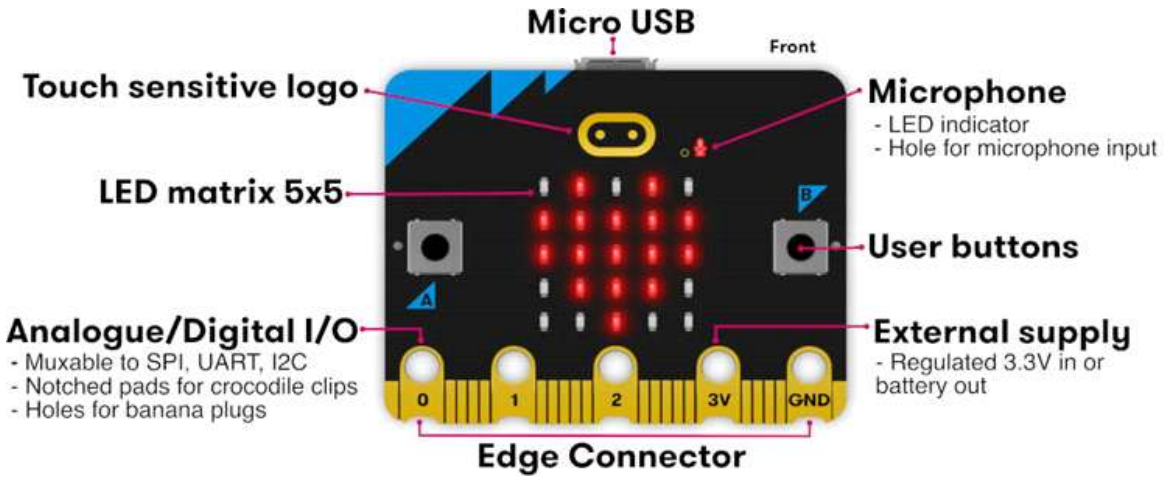
1) 탄생

- 마이크로비트는 영국의 국영방송사인 BBC에서 마이크로소프트, 삼성전자, ARM 등과 같은 회사와 함께 개발한 4 x 5cm 크기의 소형 교육용 보드입니다.
- 외부 입출력을 할 수 있는 많은 단자들이 있으며, 자체 나침반 센서, 온도 센서, 가속도 센서 등을 내장하고 있어 추가적인 부품을 사용하지 않고 마이크로비트만 가지고 있어도 된다는 점이 장점입니다. 또한 통신을 위한 블루투스 4.0을 내장하고 있어 블루투스 모듈 없이 스마트폰 및 스마트 기기와 연동할 수도 있습니다.

2) 재원

- Arm Cortex-M4 32bit processor with FPU @ 64Mhz 마이크로컨트롤러
  - 프로그래밍이 가능한 입/출력 모듈을 가지고 있는 작은 컴퓨터 칩
  - 연산장치, 기억장치, 입출력 장치가 모두 합쳐진 기능 수행
- 512KB ROM, 128KB RAM
- 3.3V에서 작동
- 자체 나침반 센서, 온도 센서, 가속도 센서 등을 내장

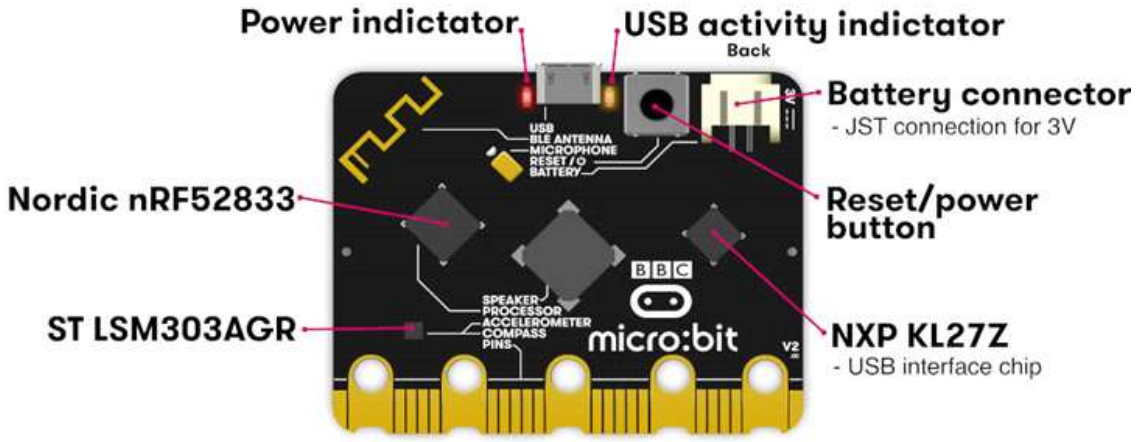
3) 전면부의 구성



[마이크로비트 전면부의 모습]

- 전면부는 다음과 같은 구성을 가지고 있습니다. LED 25개(가로, 세로 5개씩), A 버튼, B 버튼, I/O 핀, 3V, GND.
- LED는 전면부의 가운데 가로와 세로 5개씩 총 25개가 있으며 밝기 조절이 가능하고 디스플레이 용도뿐만 아니라 광센서의 역할도 합니다. A 버튼은 전면부의 가운데 왼쪽에 있는 버튼이며, B 버튼은 전면부의 가운데 오른쪽에 있는 버튼으로 프로그래밍을 통해 사용할 수 있는 푸시버튼으로 게임 및 음악 컨트롤을 위해 사용할 수 있습니다. I/O 핀은 전면부 하단에 있는 5개의 구멍 중 왼쪽부터 3개의 구멍을 말하며 이는 0, 1, 2로 표시되어 있으며 악어 클립 또는 케이블을 사용해서 외부 하드웨어와 연결하여 전력을 공급하거나 디지털 혹은 아날로그 데이터의 입, 출력이 가능한 핀입니다. 3V와 GND는 전원 공급을 필요로 하는 전동 모터 등과 같은 외부 기기에 전원을 공급하는데 사용합니다.

4) 후면부의 구성



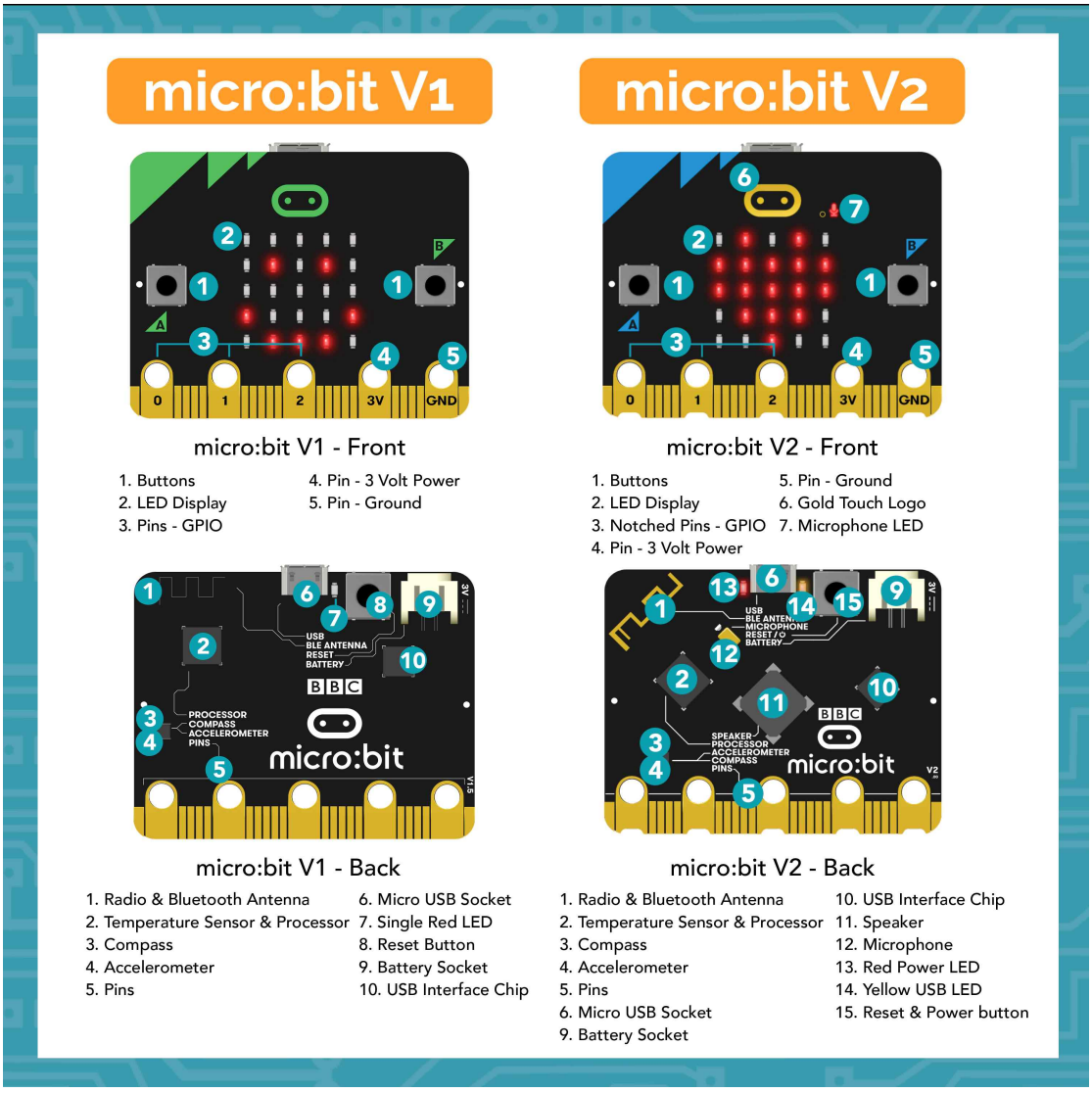
[마이크로비트 후면부의 모습]

- 후면부는 다음과 같은 구성을 가지고 있습니다. 저전력 블루투스 안테나, USB 소켓, 재시작 버튼, 배터리 소켓, 상태 표시 램프, 마이크로컨트롤러, 가속도 및 지자기 센서입니다. 저전력 블루투스 안테나는 후면부 상단 왼쪽에 있는 직선 모양 음파로 되어 나온 부분이며 스마트폰, 태블릿PC 혹 PC 등 BLE 안테나를 가진 다른 기기들과 정보를 주고받을 수 있어 사물인터넷(IoT)을 위한 프로그램에도 활용할 수 있습니다.
- USB 소켓은 후면부 상단 가운데 있는 처음으로 돌출된 곳이며 제작한 프로그램은 USB 케이블을 통해 마이크로비트로 다운로드할 수 있습니다. 재시작 버튼은 USB 소켓 오른쪽에 있는 돌출된 곳으로 후면부 상단 돌출된 3곳 중 가운데이며 마이크로비트를 재시작할 때 사용합니다. 배터리 소켓은 후면부 상단 돌

출된 3곳 중 가장 오른쪽에 있으며 외장 배터리 팩을 연결하는데 사용됩니다. USB 소켓과 재시작 버튼 사이에는 상태 표시 램프가 있으며 이 램프의 역할은 시스템에서 사용자에게 무언가 알릴 상황이 있으면 노란색 LED 램프가 깜빡이게 됩니다. 후면부 상단 왼쪽에 있는 저전력 블루투스 안테나의 바로 밑에 조금 돌출된 곳이 있는데 이곳은 마이크로컨트롤러로 256KB의 플래시 메모리와 16KB의 RAM을 내장하고 있고, 온도 센싱이 가능합니다.

- 마지막으로 마이크로컨트롤러의 아래쪽에 위치하고 있으며 하단의 왼쪽 첫 번째 홀(GDN) 위에 있는 작은 칩은 가속도 및 지자기 센서입니다. 이를 통해 속도 변화, 자기장 감지, 방향 검출 등이 가능합니다.

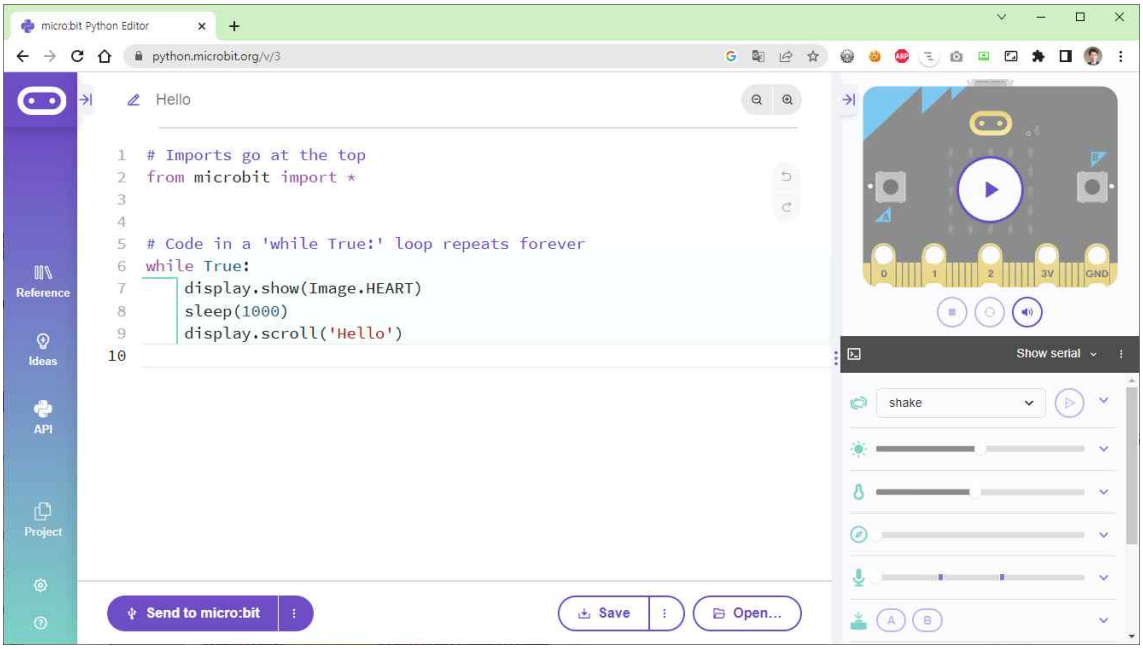
5) 마이크로비트 버전 비교



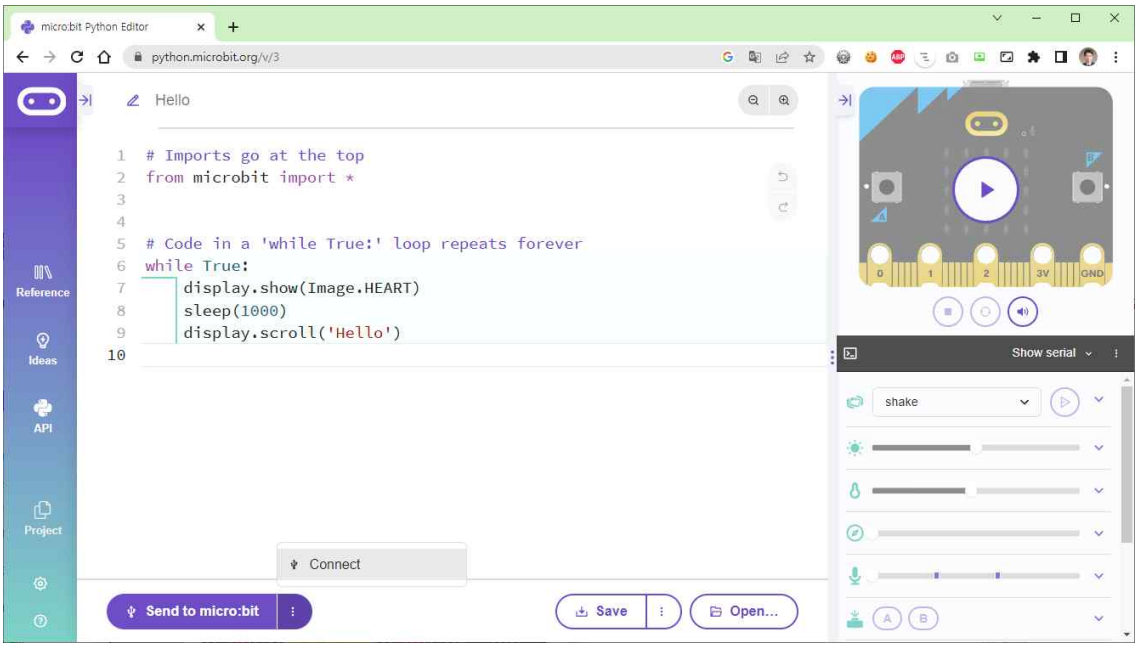
2. 파이썬 에디터 사용

1) 파이썬 에디터(Python editor) 열기

https://python.microbit.org/v/3



2) 장치 페어링

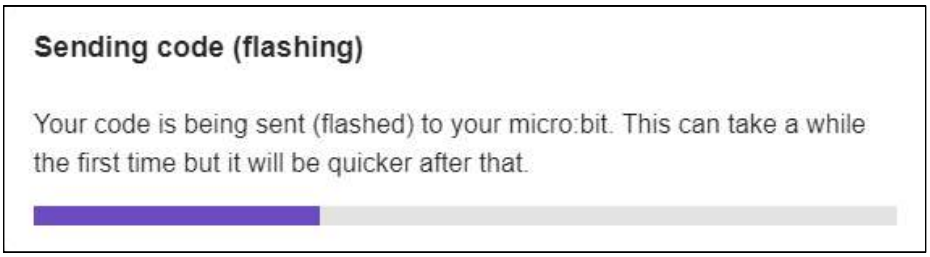


왼쪽 하단 (Send to micro:bit) 버튼 오른쪽의 … 을 클릭하고 'Connect'을 선택한 뒤 아래 그림과 같이 장치를 선택하고 '연결' 버튼을 누릅니다.

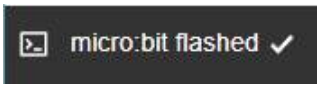
위 그림에서와 같이 'micro:bit ready to flash ↯'라는 메시지가 보이면 성공입니다.

3) 테스트 프로그램 실행

이제 바로 업로드 기능이 정상작동하는지 확인하기 위해 (Send to micro:bit) 버튼을 눌러보겠습니다. 최초 업로드 시에는 아래의 메시지가 나오면서 약간의 시간이 필요할 수도 있습니다.



잠시 후 아래의 메시지가 나타나면 프로그램 업로드가 완료된 것입니다.



이제 마이크로비트의 LED 매트릭스에 하트 아이콘이 표시되고 1초 뒤, 'Hello'라는 문자열이 스크롤 될 것입니다. 이것은 파이썬 에디터가 자동으로 만들어 준 기본프로그램입니다.

3. 파이썬 기초

1) 개요

```
Life is short, You need Python.
인생은 짧다. 당신에겐 Python이 필요하다.
```

위 문장은 Python의 엄청나게 빠른 개발 속도와 생산성을 두고 개발자들 사이에서 유행처럼 퍼진 말입니다. 하지만 프로그램을 만드는 속도가 빠르다는 뜻일 뿐, 실행속도는 오히려 느린 편이고 실행 파일 사이즈는 쓸데없이 거대한 단점도 있으니 무조건적인 파이썬 찬양은 곤란합니다. 모든 면에서 만능인 언어는 아직 없는 셈이지요.

2) 탄생

파이썬의 로고

파이썬(Python)은 1991년 네덜란드계 소프트웨어 엔지니어인 귀도 반 로섬이 발표한 고급 프로그래밍 언어로, 플랫폼에 독립적이며 인터프리터식, 객체지향적, 동적 타이핑(dynamically typed)[1] 대화형 언어이다. 파이썬이라는 이름은 귀도가 좋아하는 코미디인〈Monty Python's Flying Circus〉에서 따온 것입니다. 이름에서 고대신화에 나오는 커다란 뱀을 연상하는 경우도 있겠지만 이와는 무관합니다.

3) 특징

파이썬은 초보자부터 전문가까지 사용자층을 보유하고 있다. 동적 타이핑(dynamic typing) 범용 프로그래밍 언어로, 펄 및 루비와 자주 비교됩니다. 다양한 플랫폼에서 쓸 수 있고, 라이브러리(모듈)가 풍부하여, 대학을 비롯한 여러 교육 기관, 연구 기관 및 산업계에서 이용이 증가하고 있습니다. 또 파이썬은 순수한 프로그램 언어로서의 기능 외에도 다른 언어로 쓰인 모듈들을 연결하는 접착제 언어로써 자주 이용됩니다. 실제 파이썬은 많은 상용 응용 프로그램에서 스크립트언어[2]로 채용되고 있습니다. 도움말 문서도 정리가 잘 되어 있으며, 유니코드 문자열을 지원해서 다양한 언어의 문자 처리도 가능합니다.

---

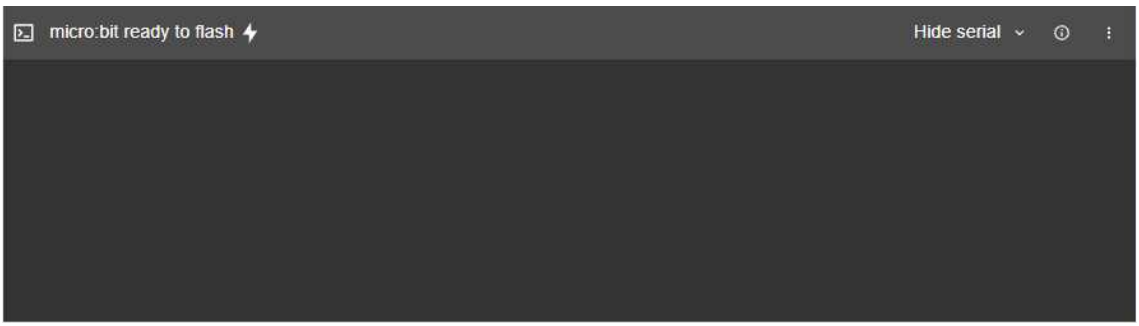
1) 동적 타입 언어의 자료형은 컴파일 시 자료형을 정하는 것이 아니고 실행 시에 결정합니다.
2) 프로그래밍 언어의 한 종류로, 기존에 이미 존재하는 소프트웨어(애플리케이션)를 제어하기 위한 용도로 쓰이는 언어입니다.

4. 파이썬 기초 코딩

1) print() 함수

파이썬에서는 데이터를 화면에 출력하고 싶을 때 표준 출력 함수인 print() 함수를 사용합니다. 마이크로비트에서 print() 함수를 사용하면 시리얼 포트를 통해 컴퓨터로 내용이 전송됩니다. 전송된 내용을 확인하려면 파이썬 에디터 화면 하단부의 'Show serial' 링크를 클릭하세요.

그러면 아래와 같이 창이 넓어지면서 시리얼 모니가 나타납니다. 이곳에서 우리는 마이크로비트가 보내주는 출력을 확인할 수 있습니다.



다음은 다양한 타입의 데이터를 print() 함수를 사용하여 출력하는 예제입니다.

print.py

```
1 print(38)
2 print("문자열")
3 print([1, 2, 3])
```

2) 변수

대부분의 프로그램은 데이터를 활용하여 다양한 동작을 수행하며, 이러한 데이터는 컴퓨터의 메모리 공간에 저장될 수 있습니다. 이때 데이터를 저장한 메모리 공간에 이름을 할당받아야만 나중에 다시 해당 데이터에 접근할 수 있습니다. 이처럼 프로그램에서 사용되는 데이터를 저장해 놓는 일종의 저장 공간을 변수(variable)라고 부릅니다. 즉, 변수란 데이터를 저장할 수 있도록 이름을 할당받은 메모리 공간을 의미하며, 이렇게 저장된 데이터에는 언제든지 다시 접근하거나 그 값을 변경할 수 있습니다.

variable.py

```
1 korea = "동해 물과 백두산이 마르고 닳도록..."
2 print(korea)
3 print(korea + "\n")
4 korea = "남산 위에 저 소나무 철갑을 두른 듯"
5 print(korea)
```

3) 변수명

다음은 파이썬에서 변수명을 지을 때 지켜야하는 규칙들입니다.

① 변수명은 영문자(대소문자), 숫자, 언더스코어(_)로만 작성할 수 있습니다.

② 변수명은 숫자로 시작할 수 없습니다. 즉, 반드시 영문자나 언더스코어(_)로 시작해야 합니다.

③ 변수명은 대소문자를 구분합니다.

④ 변수명에는 파이썬에서 미리 정의된 예약어(reserved words)는 사용할 수 없습니다.

4) if-else문

if-else 문은 if 키워드 뒤에 위치한 조건식의 결과에 따라 실행되는 명령문이 달라집니다.

조건식의 결과가 참(True)인 경우에는 if 문 바로 다음의 명령문들이 실행되며, else 문 다음의 명령문들은 실행되지 않습니다.

반대로 조건식의 결과가 거짓(False)인 경우에는 else 문 바로 다음의 명령문들이 실행되며, if 문 바로 다음의 명령문들은 실행되지 않습니다.

파이썬에서 블록(block)은 콜론(:)으로 시작하여 동일한 들여쓰기(indentation) 구간을 의미하므로, if 키워드와 else 키워드의 맨 끝에도 반드시 콜론(:)을 삽입해야 합니다.

variable.py

```
1 con = "sweet"
2 if con == "sweet":
3     print("삼키다")
4 else:
5     print("뱉는다")
```

5) if-elif-else 문

if-else 문만으로는 우리가 실제로 구현해야 하는 복잡한 조건들을 제대로 표현하는 것이 매우 힘들고 어려울 수 있습니다. 여러 가지 조건에 따른 분기를 표현하려면 if-elif-else 문이 적합할 수 있습니다.

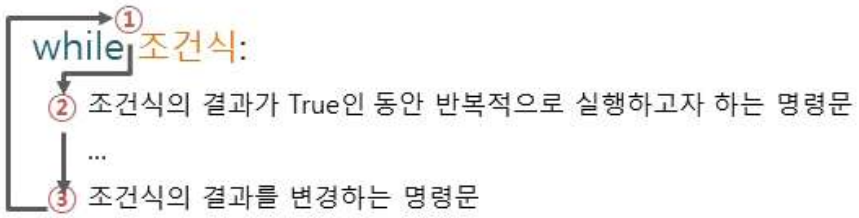
variable.py

```
1  season = "winter"
2
3  if season == "spring":
4      print("봄이 왔네요!")
5  elif season == "summer":
6      print("여름에는 더워요~")
7  elif season == "fall":
8      print("가을은 독서의 계절!")
9  elif season == "winter":
10     print("겨울에는 눈이 와요~")
```

위 예제는 네 계절에 따라 서로 다른 문구가 출력되어야 하는 상황을 if-elif-else문을 사용하여 하나의 제어구조로 표현하였습니다.

6) while 문

while 문은 조건식이 특정 조건을 만족할 때까지 계속해서 주어진 명령문을 반복 실행합니다.



while 문을 만난 프로그램은 가장 먼저 조건식(①)의 결과가 참(True)인지를 검사합니다. 만약 조건식의 결과가 참(True)이라면 프로그램의 흐름은 while 문 내부로 진입하며, 만약 결과가 거짓(False)이라면 while 문에 진입하지 않고 건너뛰게 됩니다.

while 문 내부로 진입한 프로그램은 내부에 포함된 모든 명령문(②)을 실행하고 나서 또다시 조건식을 검사합니다. 이렇게 조건식을 검사하고 명령문을 모

두 수행한 후 또다시 조건식을 검사하는 모양이 마치 고리와 같다고 하여 반복문을 루프(loop)라고도 부릅니다.

variable.py

```
1 i = 1
2 while i < 11: # 조건식
3     print("파이썬 " + str(i))
4     i = i + 1 # 탈출 조건
```

7) for 문

while 문은 구조상 while 문 내부에 조건식의 결과를 변경하는 명령문을 삽입해야하기 때문에 종종 실수를 하게 됩니다. 하지만 for 문은 이러한 조건식의 결과를 변경하는 명령문을 따로 사용하지 않고도 튜플이나 리스트, 문자열 등을 원하는 횟수만큼 반복할 수 있도록 해줍니다.

문법	for 변수 in 문자열(or 튜플 or 리스트): 반복적으로 실행하고자 하는 명령문 ⋮

for 문을 만난 프로그램은 우선 in 키워드가 가리키는 문자열, 튜플 또는 리스트(①)의 첫 번째 요소를 꺼내 변수(②)에 대입합니다. 이렇게 대입된 변수는 for 문 내부의 명령문(③)에서 자유롭게 사용할 수 있습니다.

for 문 내부의 모든 명령문을 수행하고 나면 또다시 해당 문자열, 튜플 또는 리스트(①)로 돌아가 다음 요소가 있는지를 검사합니다. 만약 다음 요소가 존재한다면 해당 요소를 변수에 대입하고 또다시 루프를 실행하며, 만약 다음 요소가 존재하지 않는다면 for 문은 종료됩니다.

8) range 함수

파이썬에서 for 문과 함께 가장 많이 사용되는 함수는 바로 range() 함수일 것입니다. range() 함수는 전달된 인수에 따라 연속된 정수들을 반환하는 함수로, 전달되는 인수의 수에 따라 다음과 같이 두 가지 방식으로 사용할 수 있습니다.

9x9.py

```
1 for col in range(2, 10, 1):
2     for row in range(1, 10):
3         print (col, " x ", row, " = ", col * row)
```

8) break 문

반복문을 통해 명령문을 반복해서 수행하다보면 프로그램의 흐름상 특정 조건을 만족할 때 더 이상 반복문을 수행하지 않고 그 즉시 해당 반복문을 빠져나가야 할 경우가 생깁니다. 이러한 경우에는 break 키워드를 사용하여 반복 조건에 상관없이 가장 가까운 반복문을 즉시 탈출할 수 있습니다.

다음 예제는 구구단을 5단까지만 출력하도록 한 예제입니다.

break.py

```
1 for col in range(2, 10):
2     if col > 5:
3         break
4     for row in range(1, 10):
5         print (col, " x ", row, " = ", col * row)
```

9) continue 문

break 키워드가 해당 반복문 전체를 빠져나가게 해준다면, continue 키워드는 해당 루프만을 즉시 종료하고 다음 루프를 실행시킵니다.

즉, continue 키워드는 해당 키워드 바로 다음 명령문부터 해당 반복문의 마지막 명령문까지를 모두 건너뛰고 바로 다음 반복을 실행하는 것입니다.

다음은 1부터 10까지의 정수 중 홀수만을 출력하는 예제입니다.

continue.py

```
1 for n in range(1, 11):
2     if n % 2 == 0:
3         continue
4     print(n, "은 홀수입니다.")
```

10) 리스트(list)

파이썬에서는 기본 데이터 타입인 숫자형 타입, 불리언 타입, 문자열 타입과는 별도로 이들로 구성되는 다양한 컨테이너 형태의 데이터 구조를 제공합니다. 그 중에서도 가장 많이 사용되는 것이 바로 리스트(list) 타입입니다.

리스트(list)는 간단히 순서대로 늘어선 박스로 이해할 수 있습니다. 각 박스에는 다양한 타입의 데이터를 저장할 수 있으며, 이 박스들을 통틀어서 리스트라고 부르게 됩니다.

① 리스트 선언

파이썬에서 리스트는 대괄호([])로 감싸서 선언할 수 있으며, 리스트 안의 요소

(element)들은 쉼표(,)로 구분합니다.

```
리스트명 = [요소1, 요소2, 요소3, ...]
```

list.1py

```
1 primes = [2, 3.14, 'A', "string"]
2 for p in primes:
3     print(p)
4 print(len(primes))
```

위의 예제처럼 리스트는 같은 타입의 데이터뿐만 아니라 서로 다른 타입의 데이터도 함께 저장할 수 있으며, for 문을 사용하면 저장된 모든 요소에 손쉽게 접근할 수 있습니다.
또한, 파이썬 내장 함수인 len() 함수를 사용하면 리스트의 길이, 즉 리스트에 저장된 모든 요소의 개수를 손쉽게 알 수 있습니다.

② 리스트 요소

for 문을 사용하면 리스트의 모든 요소에 순차적으로 접근할 수 있습니다. 하지만 리스트의 특정 요소에만 접근하고 싶을 때에는 0부터 시작하는 인덱스(index)를 사용해야만 합니다.
리스트의 첫 번째 요소는 0, 두 번째 요소는 1, 세 번째 요소는 각각 2의 인덱스를 가지며 이처럼 리스트의 각 요소는 저장될 때 순서대로 1씩 증가되는 인덱스를 자동으로 부여받습니다.

list2.py

```
1 primes = [2, 3, 5, 7]
2 print(primes[0])
3 print(primes[-1])
4 print(primes[1] + primes[3])
```

list2.py

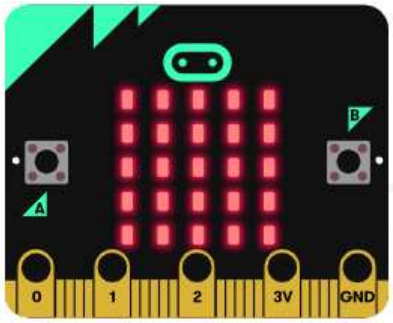
```
1 list1 = [1, 2, 3, 4, 5]
2 list1.append(2)
3 print(list1)
4
5 list1.remove(3)
6 print(list1)
7 list1.remove(2)
8 print(list1)
```

Part 02

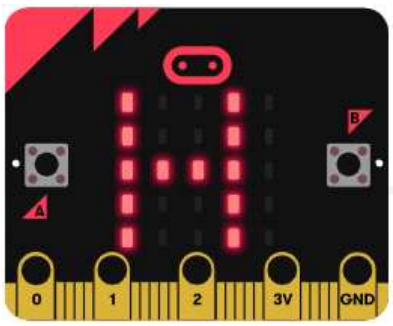
LED 출력

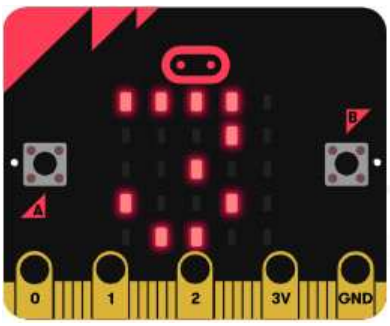
LED 매트릭스

발광 다이오드를 의미하는 LED는 빛을 출력하는 장치입니다. 마이크로비트에는 프로그램 가능한 25개의 LED가 내장되어 있습니다. 프로그램 가능하다는 뜻은 프로그램을 만드는 사람이 원하는 대로 켜거나 끌 수 있다는 뜻입니다.

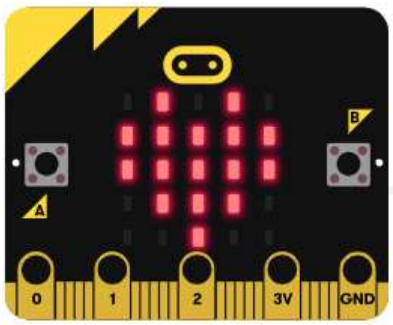


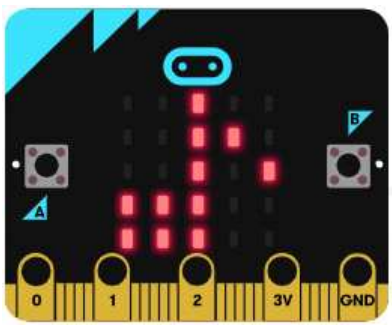
25개의 LED는 왼쪽 그림과 같이 5행 5열로 배치되어 있는데 이러한 형태의 배열을 매트릭스라고 합니다. 여기서는 LED로 매트릭스를 만들었으니 'LED 매트릭스'가 되는 것이지요.
LED 매트릭스는 각 점들을 켜거나 끔으로써 여러 가지 의미를 만들어 낼 수 있습니다.





위쪽 그림과 같이 알파벳이나 숫자를 표현할 수도 있고 스크롤 기능을 통해 'Hello'와 같은 긴 문자열을 표현할 수 있습니다.
또한 아래 그림과 같이 그림이나 아이콘을 그리는 것도 가능합니다.





1. 문자 출력

1) 문자와 문자열의 개념

먼저 코딩의 세계에서 '문자열'이 무엇인지 알아봅시다. 여러분이 키보드로 타이핑 할 수 있는 글자 하나하나를 '문자'라 부릅니다. 그리고 이러한 문자가 하나 이상 모인 집합을 '문자열'이라고 합니다. "Hello", "World"와 같은 문장을 문자열이라 할 수 있지요. 파이썬에서 문자열은 반드시 작은따옴표 또는 큰따옴표로 묶어서 표기해야 합니다.

2) 파이썬 에디터의 기본 프로그램 분석

hello.py

```
1  # Imports go at the top
2  from microbit import *
3
4  # Code in a 'while True:' loop repeats forever
5  while True:
6      display.show(Image.HEART)
7      sleep(1000)
8      display.scroll('Hello')
```

먼저 파이썬 에디터를 시작하면 자동으로 만들어지는 기본프로그램을 살펴보겠습니다. 소스코드는 위와 같습니다.

1번과 4번 라인은 '#' 문자로 시작됩니다. '#' 문자는 특수한 기능을 하는 문자로 '#' 문자 이후 문자열이 주석문이라는 것을 의미합니다. 주석문은 컴퓨터가 실행해야 할 명령어 아니라 사람을 위한 설명문이라는 뜻이기 때문에 컴퓨터는 주석문을 무시합니다.

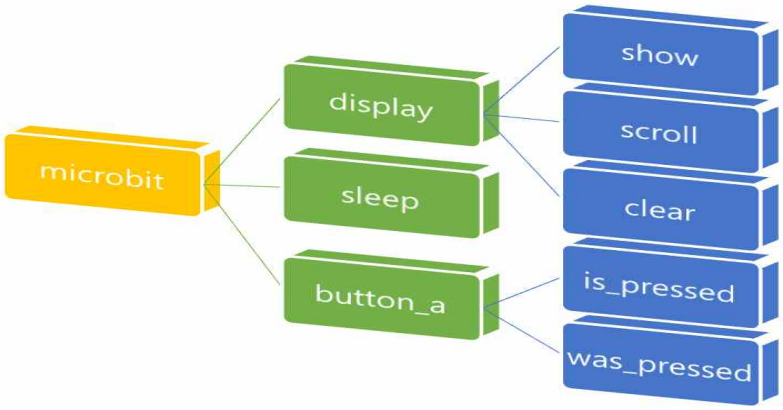
2번 라인은 'microbit' 라는 모듈의 전체(*)를 가져온다는 의미입니다. 모듈은 어떤 특정한 기능들을 수행하기 위해 필요한 코드(변수와 클래스)를 모아놓은 파이썬 파일을 의미합니다. 마이크로비트에서 뭔가를 만드려면 microbit 모듈은 필수로 필요하기에 2번 라인은 매번 필요한 문장이라 할 수 있겠습니다.

5번 라인의 while 문장은 python 대표적인 반복문으로 어떠한 명령어를 반복적으로 수행해야 할 때 사용합니다. 형식은 오른쪽 그림과 같으며 <조건문>이 들어갈 자리에 True 를 적게 되면 조건이 항상 참이기 때문에 무한 반복을 하겠다는 의미가 됩니다. 반복 수행할 명령어는 들여쓰기로 구분된 블록에 속하는 문장들이 됩니다.

```
while <조건문> :
    <수행할 문장1>
    <수행할 문장2>
    <수행할 문장3>
    ...
```

[while문의 기본구조]

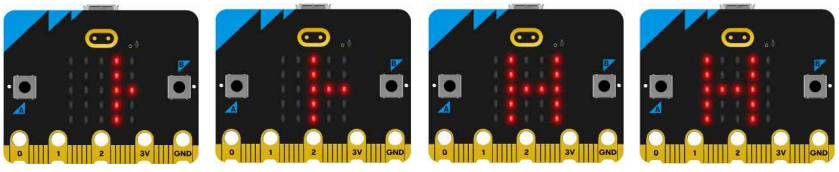
6번 라인의 display.show(Image.HEART)라는 문장은 display 객체에 속하는 show 라는 함수(메서드)에 Image.HEART 라는 인수를 전달하여 호출하겠다는 뜻입니다. 이렇게 설명하면 프로그래밍 초심자에게는 이해하기 힘든 어려운 말일 수 있으니 조금 자세히 살펴보자면...,



2번 라인에서 import 한 microbit 모듈의 내부는 기능의 사용을 일목요연하게 하기 위해 내부 구조가 위와 같이 되어 있습니다. 즉 관련 있는 것끼리 계층적으로 묶여 있다는 뜻입니다. 화면(LED 매트릭스)에 보여주고(show), 화면을 스크롤 하고(scorll), 화면을 지우는(clear) 기능(=함수)을 display에 묶어 놓은 것이지요. 그래서 화면에 보여주는 기능을 사용한다는 코딩의 표현이 display 묶음의 show 라는 기능을 사용하겠다는 뜻에서 display.show() 가 됩니다. 마지막의 '()' 는 함수의 호출, 즉 'show 기능을 실행하라!'라는 의미입니다. 그런데 show는 화면에 보여주는 기능인데 무엇을 보여줄 것인지 목적어를 지정해야 말이 됩니다. 바로 이 목적어에 해당하는 것이 Image 묶음(객체)의 HEART라는 뜻이고 코드로 표현하면 Image.HEART가 되는 것입니다. 그래서 6번 라인은 화면에 HEART 이미지를 출력하는 코드가 됩니다.

7번 라인은 sleep 함수는 마이크로비트를 일시 정지시키는 기능입니다. 괄호 안에 원하는 일시 정지 시간 값을 넣어주는데 ms(밀리 초) 단위이므로 1초를 정지시키려면 1000을 0.5초를 정지시키려면 500을 넣어야 합니다.

마지막 8번 문장의 display.scroll('Hello')는 display 묶음(객체)의 scroll 기능(함수)을 호출하겠다는 뜻이며, scroll은 LED에 지정한 글자를 아래 그림과 같이 슬라이드 시키면서 보여주는 기능입니다.



3) 실습예제

display.py

```
1 from microbit import *
2 while True:
3     display.scroll('count down', delay=50)
4     display.show('9876543210', delay=1000, loop=False, wait=True)
5     display.clear()
```

3번 라인에는 이전에 살펴봤던 scroll 함수가 사용되었습니다. 하지만 추가로 delay=50 이라는 인수가 추가되었는데 이것의 의미는 스크롤 지연 시간을 0.05초로 지정하겠다는 뜻입니다. 그래서 count down 이라는 문자열을 아주 빠르게 스크롤 하게 됩니다. delay를 지정하지 않을 시 기본값은 400 (0.4초) 입니다.
4번 라인의 show라는 함수는 문자열를 출력하는 함수입니다. scroll 함수와는 달리 스크롤 없이 문자를 한 개씩 즉시 출력합니다. 여기에서는 delay가 1000으로 지정되었으므로 1초에 한 개씩 순서대로 다음 문자를 출력하게 됩니다. loop 인수는 반복 출력 여부를 결정하고, wait 인수는 출력을 완료할 때까지 프로그램의 제어가 해당라인에 멈춰 있을지를 결정합니다. 만약 wait=False를 지정하면 화면에 출력을 진행하는 중에 다음 줄의 코드를 실행하러 제어가 이동됩니다.
그리고 5번 라인의 display.clear()는 모든 LED를 끄는 기능(함수)입니다.
마지막으로 3, 4, 5번 라인은 while 반복문의 블록으로 들여쓰기 되어 있으므로 3, 4, 5번 라인은 계속하여 반복 실행되게 됩니다. 파이썬에서는 블록을 들여쓰기로 구분하므로 동일 블록이라면 아래 그림과 같이 모두 같은 위치에 들여쓰기 되어야 함에 유의하세요.

```
while True:
    btn_a_prev = btn_a_curr
    btn_a_curr = button_a.is_pressed()

    if btn_a_prev==RELEASE and btn_a_curr==RELEASE:
        if(running_time() - btn_a_release_time >= TICK*3):
            display.clear()
            cur_y = 0

    if btn_a_prev==RELEASE and btn_a_curr==PRESSED:
        btn_a_pressed_time = running_time()
        speaker.on()
        music.pitch(880)
```

[블록 구분 예시]

2. 내장 이미지 출력

1) 개요

micro:bit에는 디스플레이에 표시할 수 있는 많은 내장 이미지가 있습니다. 이미지를 출력하기 위해 display 객체의 show 함수를 사용하며, 내장 이미지는 Image 객체를 통해 접근할 수 있습니다.

2) 내장 이미지 실습 예제

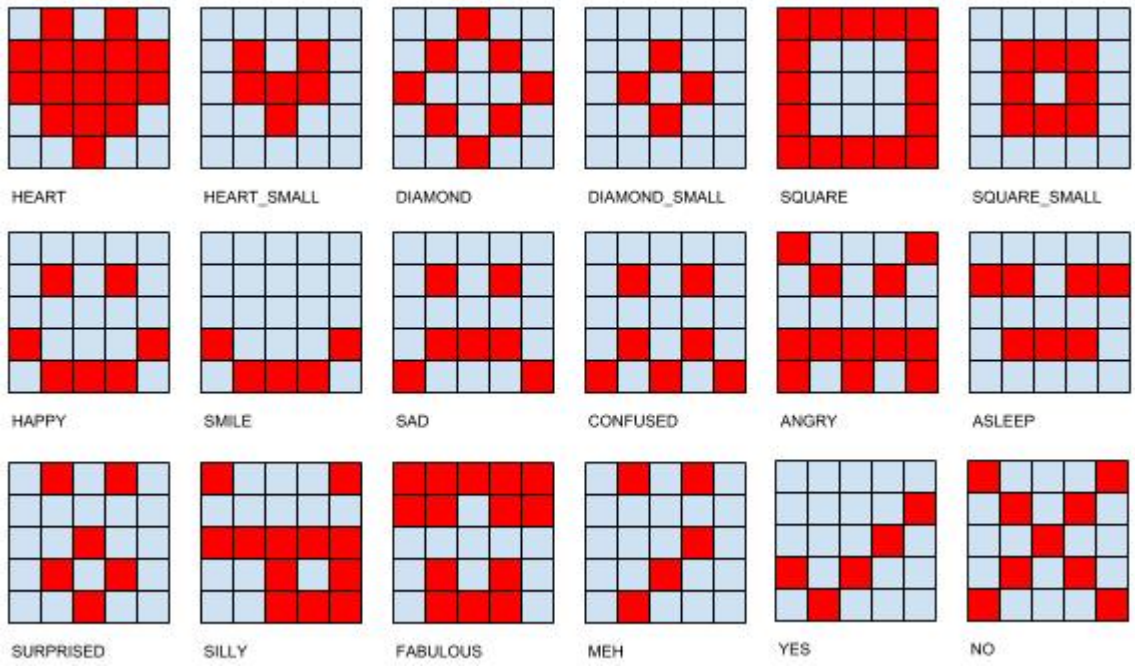
display_image.py

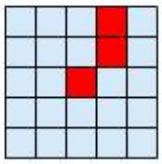
```
from microbit import *

while True:
    display.show(Image.HEART)
    sleep(1000)
    display.show(Image.DIAMOND)
    sleep(1000)
    display.show(Image.SMILE)
    sleep(1000)
    display.show(Image.ANGRY)
    sleep(1000)
```

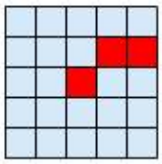
display_arrow.py

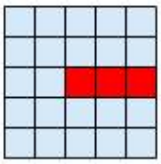
```
from microbit import *

while True:
    display.show(Image.ARROW_E)
    sleep(1000)
    display.show(Image.ARROW_N)
    sleep(1000)
    display.show(Image.ARROW_W)
    sleep(1000)
    display.show(Image.ARROW_S)
    sleep(1000)
```

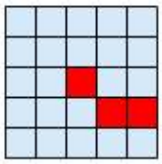
3) 출력 가능한 내장 이미지

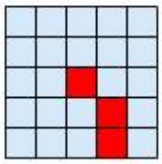


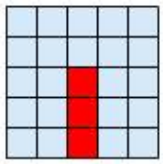

CLOCK1

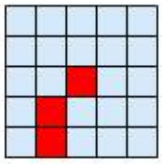

CLOCK2

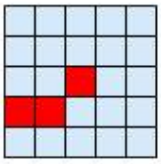

CLOCK3

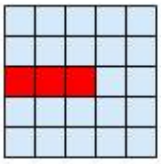

CLOCK4

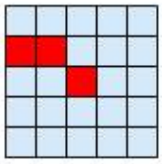

CLOCK5

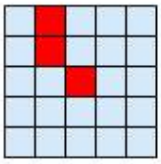

CLOCK6

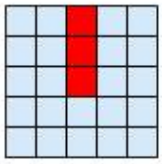

CLOCK7

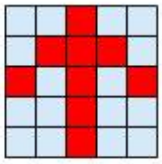

CLOCK8

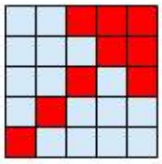

CLOCK9

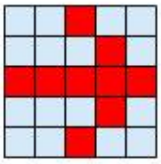

CLOCK10

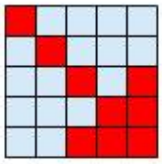

CLOCK11

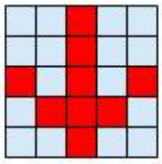

CLOCK12

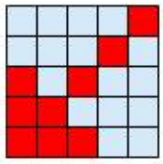

ARROW_N

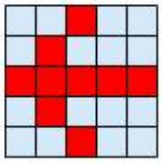

ARROW_NE

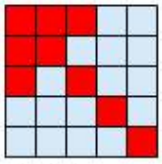

ARROW_E

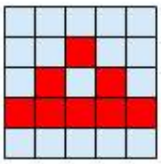

ARROW_SE

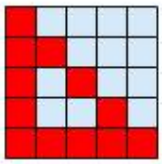

ARROW_S

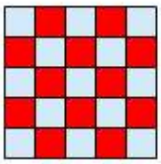

ARROW_SW

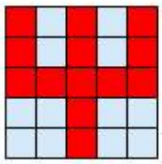

ARROW_W

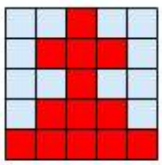

ARROW_NW

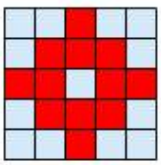

TRIANGLE

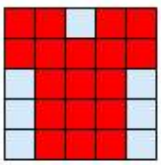

TRIANGLE_LEFT

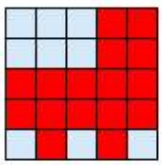

CHESSBOARD

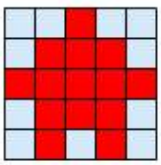

PITCHFORK

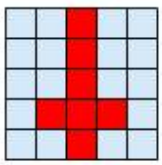

XMAS

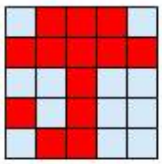

TARGET

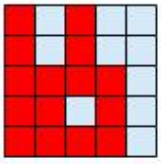

TSHIRT

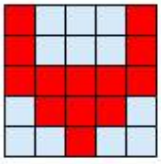

ROLLERSKATE

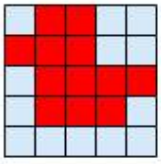

HOUSE

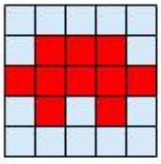

SWORD

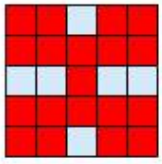

UMBRELLA

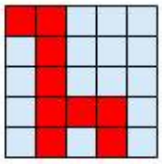

RABBIT

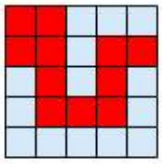

COW

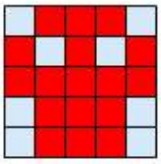

DUCK

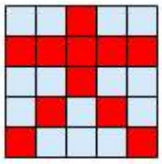

TORTOISE


BUTTERFLY


GIRAFFE


SNAKE


SKULL


STICKFIGURE

PACMAN

GHOST

MUSIC_CROTCHET

MUSIC_QUAVER

MUSIC_QUAVERS

3. 커스텀 이미지 출력

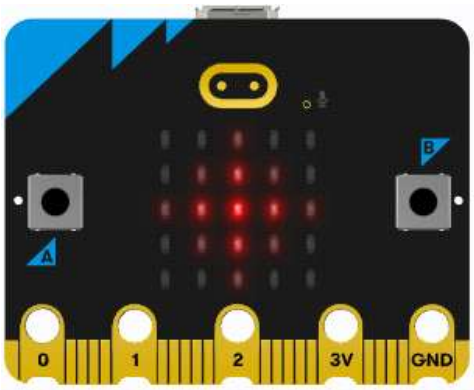
1) 개요

LED 매트릭스에 여러분이 직접 디자인 한 이미지를 출력하는 것도 가능합니다. 심지어 각 화소의 밝기를 지정하여 더욱 디테일하게 이미지를 표현할 수 있습니다.

2) 실습 예제: Image() 함수로 이미지 생성

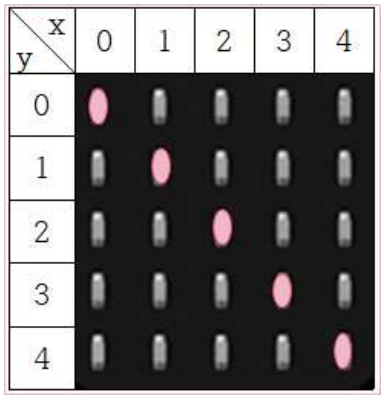
다음 예제는 십자 모양의 과녁을 출력하게 됩니다. 이미지를 구성하는 각 화소의 밝기를 0~9의 숫자로 표현하여 지정하고 있습니다.

display.custom_image.py	실행 결과

`from microbit import *`
 	
`display.show(Image('00300:'`
`'03630:'`
`'36963:'`
`'03630:'`
`'00300'))`
	

4. LED 개별 제어 – 변수의 활용

1) LED 좌표계

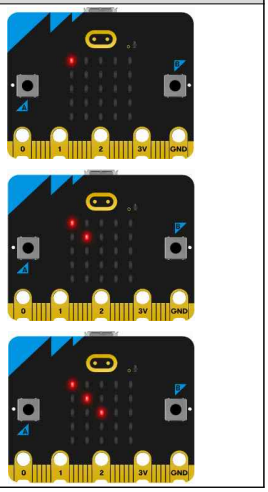


가로 방향을 x축, 세로 방향을 y축이라 하고 좌표의 시작 숫자는 0부터 시작합니다.
따라서 왼쪽 상단의 좌표는 x:0, y:0,
오른쪽 하단의 좌표는 x:4, y:4 라고 할 수 있습니다.
자 그렇다면, 왼쪽 상단(0,0)에서 오른쪽 하단(4,4) 대각선 방향으로 LED를 차례로 켰다가 끄는 애니메이션을 만들려면 어떻게 코딩을 하면 될까요?

2) 변수의 사용

변수는 '변하는 수'라는 의미로 코딩의 세계에서 컴퓨터에게 무언가를 기억시켜야 할 때 사용합니다. 예를 들어, 1부터 10까지 차례로 숫자를 출력하여야 한다면 현재 출력 중인 숫자가 무엇인지 기억하고 있어야 다음 숫자로 넘어갈 수 있기 때문에 현재 출력 숫자를 변수에 기억시키는 방식으로 코딩을 합니다. 변수를 사용하기 위해서는 이름을 붙여줘야 하는데 변수의 이름이 중복되어서는 안됩니다. 이름이 겹치면 서로 구분이 되지 않기 때문이지요. 그리고 한 가지 더! 파이썬은 식별자의 대소문자를 구분합니다. 그렇기에 'Apple'과 'apple'은 서로 다른 변수로 인식됩니다. 초심자들이 많이 실수하는 부분이지요.

3) 실습예제: ↘ 방향으로 반전하는 LED

display.pixel.py		실행 결과
1	`from microbit import *`	
2		
3	`y=0`	
4		
5	`while True:`	
6	`    for x in range(5):`	
7	`        y = x`	
8	`        b = abs(display.get_pixel(x, y)-9)`	
9	`        display.set_pixel(x, y, b)`	
10	`        sleep(100)`	
11		

- **변수**

3번 라인 `y=0` 의 의미는 변수 y에 0을 기억시키겠다는 뜻입니다. '=' 기호가 프로그래밍에서는 '같다'의 의미가 아니라 '대입(기억)'의 의미라는 것에 주의하세요. 파이썬에서 '같다'라는 뜻은 '==' 로 표현해야 합니다. 3번 라인에서 처음 y 변수가 등장하는데, 여기서와 같이 변수를 사용하기 전에는 반드시 초기화(시작 값을 정함)를 해 주어야 합니다.

- **for 반복문**

6번 라인에서는 for 반복문이 사용되었습니다. 여기서 '`range(n)`'은 0부터 n-1 사이의 정수를 원소로 갖는 숫자들의 목록을 만들어 냅니다. 즉, '`range(5)`'는 목록 [0, 1, 2, 3, 4] 를 만들어 냅니다. 다시 적으면 '`for x in [0, 1, 2, 3, 4]`'라고 적은 것과 동일하지요. 7번 라인의 `for` 문은 `in` 오른쪽에 있는 목록의 모든 원소에 대해서 변수 x에 값을 대입하여 7~10번

라인의 문장을 반복 수행합니다. 즉, x에 0을 집어넣고 반복, x에 1을 집어넣고 반복, x가 4일 때까지 반복을 수행합니다.

• **대입**

7번 라인은 `y=x` 이므로 변수 y에 변수 x값을 대입한다는 뜻입니다. 그러면 x값과 y값이 동일하게 지정되겠군요.

• **get_pixel, set_pixel**

이제 이어서 특정 좌표 LED의 켜짐과 꺼짐 상태를 알아내거나, 반대로 켜짐과 꺼짐 상태를 결정하는 함수를 알아봅시다.

- `display.get_pixel(x, y)` : LED 매트릭스에서 (x, y) 좌표에 해당하는 LED의 밝기를 0 ~ 9 사이 숫자로 알려준다.
- `display.set_pixel(x, y, b)` : LED 매트릭스에서 (x, y) 좌표에 해당하는 LED의 밝기를 b로 지정한다. (0 ≤ b ≤9)

• **LED 반전 계산**

8번 라인은 밝기는 정하는 코드인데, 0은 LED를 끄는 것이고 9는 LED를 최대 밝기로 켜는 것입니다. 우리는 0 아니면 9 (즉, 끄기 아니면 켜기) 이 두 가지 숫자만 사용해야 합니다. 현재 LED 밝기가 0인지 아니면 9인지는 get_pixel로 알아낼 수 있습니다. 그렇다면 어떻게 계산을 해야 LED를 반전(0일 때는 9로 바꾸고, 9일 때는 0으로 바꾸는 것)시킬 수 있을까요? 그래서 abs() 함수가 등장하게 됩니다.

```
abs(LED - 9)
```

abs()는 괄호안 수식의 절댓값을 계산해 줍니다. LED가 9이면 0이 되고, LED가 9면 0이 됩니다. 그래서 밝기 b를 저장하는 8번 라인이 아래와 같이 코딩된 것입니다.

```
b = abs(display.get_pixel(x, y)-9)
```

• **밝기 지정**

9번 라인에서는 반전된 밝기를 LED에 지정하기 위해 set_pixel() 함수를 사용하고 있습니다.

• **일시 정지**

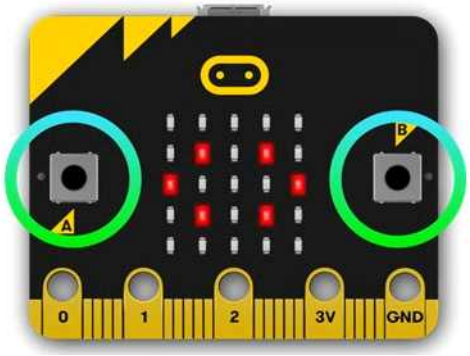
10번 라인에서는 이 모든 작업이 끝나고 다음 반복으로 넘어가기 직전 100밀리초(0.1초) 만큼 프로그램을 일시 정지시킵니다. 그래야 애니메이션 효과가 나타날 테니까요. 숫자를 작게 하면 당연히 더 빠르게 움직이게 됩니다.

Part 03

버튼 입력

버튼은 기계나 장치를 누르기 위한 스위치를 가리키는 말입니다. 오른쪽 그림과 같이 많은 종류의 버튼이 존재합니다.

마이크로비트에는 총 두 개의 버튼이 내장되어 있습니다.



왼쪽의 버튼을 A 버튼이라 하고 오른쪽의 버튼을 B 버튼이라고 합니다.

버튼의 누름 상태는 아래와 같이 총 세 가지 상태가 존재합니다.

첫째, A 버튼 누름.

둘째, B 버튼 누름.

셋째, A · B버튼 모두 누름.

그리고, 버튼의 이벤트는 아래와 같이 두 가지로 구분할 수 있습니다.

첫째, 버튼이 눌림.

둘째, 눌렸던 버튼이 떨어짐.

이제부터 이 다양한 버튼의 상태와 이벤트를 프로그램에서 어떻게 다루는지 살펴보겠습니다.

1. 버튼에 반응하기

1) button_x.was_pressed()

was_pressed() 함수는 마이크로비트의 전원이 켜진 이후 또는 마지막으로 호출된 이후 버튼이 눌렸는지 확인하는 용도로 사용합니다. 버튼이 눌리는 순간 인식되므로, 눌렀던 버튼에서 손을 떼지 않아도 눌린 것으로 판정합니다.

만약 반복하여 호출한다면 이전 was_pressed() 호출 시점 이후에 새롭게 다시 버튼을 누른 것이 아니라면 False가 반환됩니다.

2) button_x.is_pressed()

is_pressed() 함수는 현재 버튼이 눌린 상태인지 확인하는 용도로 사용합니다. was_pressed() 와는 달리 반복하여 호출하여도 현재 버튼이 눌려 있으면 계속 True가 반환됩니다.

3) 실습 예제

button.was_pressed.py

```
1  from microbit import *
2
3  while True:
4      if button_a.was_pressed():
5          display.show('A')
6      if button_b.was_pressed():
7          display.show('B')
8
9      sleep(200)
10     display.clear()
```

button.is_pressed.py

```
1  from microbit import *
2
3  while True:
4      if button_a.is_pressed():
5          display.show('A')
6      if button_b.is_pressed():
7          display.show('B')
8
9      sleep(200)
10     display.clear()
```

was_pressed() 함수를 사용한 왼쪽 프로그램은 버튼을 계속 누르고 있으면 1회만 인식되지만 is_pressed() 함수를 사용한 오른쪽 프로그램은 버튼을 누르고 있는 동안 계속하여 인식됩니다.

4번 라인의 if 문은 어떤 조건이 참인 경우에만 해야 할 일이 있을 때 사용합니다. if문의 기본 구조는 오른쪽과 같으며 <조건>이 참일 때 들여쓰기 된 블록의 명령들을 실행하게 됩니다.

```
if <조건>:
    참인경우 실행할 명령 1
    참인경우 실행할 명령 2
    참인경우 실행할 명령 3
    ....
다음 명령
```

[if문의 기본구조]

추가하여 **if**문을 확장한 **if ~ else**문, **if ~ elif ~ else** 문에 대해서 살펴보겠습니다. 형식은 아래와 같습니다.

if ~ else 문	**if ~ elif ~ else 문**

if <조건>:
  참인 경우 실행할 명령 1
  참인 경우 실행할 명령 2
else:
  거짓인 경우 실행할 명령 1
  거짓인 경우 실행할 명령 2
다음 명령
	if <조건1>:
	  조건1이 참인 경우 실행할 명령...
	elif <조건2>:
	  조건2가 참인 경우 실행할 명령...
	else:
	  모든 조건이 거짓인 경우 실행할 명령...
	다음 명령

if ~ else 문은 <조건>이 참인 경우와 거짓인 경우 해야 할 일이 따로따로 존재하는 경우 사용합니다. 예를 들어, 60점 이상이면 '합격'을 출력하고 그 밖의 경우에는 '불합격'을 출력해야 하는 경우 if ~ else 문을 사용하는 것이 최선입니다. 추가하여 if ~ elif ~ else 문은 조건을 여러번 비교해야 하는 경우 사용하는데, 예를 들어 90점 이상은 'A', 80점 이상은 'B' 70점이상은 'C', 그 밖의 경우에는 'F'를 출력해야 하는 경우 적합합니다.

4) 실습 퀴즈

A버튼을 누르면 LED에 매트릭스에 'A'를 출력하고, B버튼이 눌리면 'B'를 출력하고, A와 B버튼을 동시에 누르면 '&'문자를 출력하는 프로그램을 제작하시오.

정답 예시 프로그램은 아래와 같습니다.

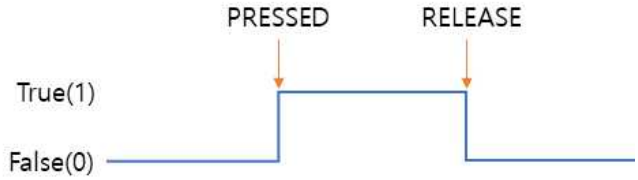
button.a_b_ab.py

```
1  from microbit import *
2  while True:
3      if button_a.is_pressed() and button_b.is_pressed():
4          display.show('&')
5      elif button_a.is_pressed():
6          display.show('A')
7      elif button_b.is_pressed():
8          display.show('B')
9      else:
10         display.clear()
11
12     sleep(100)
```

2. 버튼 눌린 시간 측정하기

1) 개요

버튼이 눌렸다 떨어지면 물리적으로 어떤 일들이 일어나는지 알아보겠습니다. 버튼은 눌린 상태(PRESSED)와 떨어진 상태(RELEASE)를 구분하기 위해 이 두 가지 상태에서 서로 다른 전압이 검출되도록 설계됩니다. 예를 들어, 버튼을 눌렀을 때 HIGH(3.3V), 눌렸던 버튼이 떨어졌을 때 LOW(0V)가 되는 식입니다. (버튼의 회로 설계를 어떻게 했느냐에 따라 반대가 될 수도 있음)



이제 이와 같은 사전 지식을 바탕으로 버튼이 눌린 시간을 측정하는 방법을 고안해 봅시다. 가장 간단하면서 쉬운 아이디어는 아래와 같습니다.

버튼이 눌린 시간 = 버튼이 떨어진 시간 – 버튼이 눌린 시간

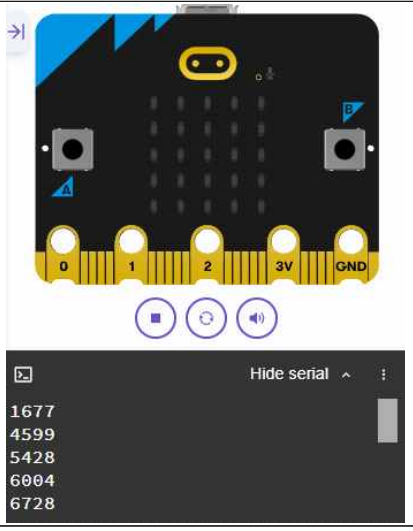
2) 실습예제1: running_time()

위 계산을 하려면 기본적으로 시간을 알아내야 하는데 이 때 사용하는 함수가 running_time()이라는 함수입니다. 이 함수는 마이크로비트가 켜진 이후 몇 밀리초가 흘렀는지 알려줍니다. 아래 실습예제로 테스트 해 봅시다.

button_running_time.py

```
1  from microbit import *
2
3  while True:
4      if button_a.was_pressed():
5          time = running_time()
6          print(time)
```

실행결과



이제 모든 지식을 조합하여 버튼이 눌린 시간을 계산하는 프로그램을 만들어 봅시다.

3) 실습예제2: 버튼 눌린 시간 알아내기

button_press_time.py

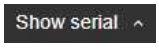
```
1  from microbit import *
2
3  PRESSED = True
4  RELEASE = False
5  btn_a_prev = RELEASE  # button A previous = 떨어짐
6  btn_a_curr = RELEASE  # button A current = 떨어짐
7  btn_a_pressed_time=0  # 버튼 누른 시간
8  btn_a_release_time=0  # 버튼 떨어진 시간
9
10 while True:
11     btn_a_prev = btn_a_curr
12     btn_a_curr = button_a.is_pressed()
13
14     if btn_a_prev==RELEASE and btn_a_curr==PRESSED: #버튼 누르면,
15         btn_a_pressed_time = running_time()
16
17     if btn_a_prev==PRESSED and btn_a_curr==RELEASE: #버튼 떨어지면,
18         btn_a_release_time = running_time()
19         pressed_ms = btn_a_release_time - btn_a_pressed_time
20         print(pressed_ms)  # 눌린 시간을 시리얼 출력
```

3번 라인에서 PRESSED 를 True 값으로, 4번 라인에서 RELEASE 를 False 값으로 정의하였습니다. 5번 라인에서 A 버튼의 이전 상태(button A previous)라는 의미의 btn_a_prev 라는 변수를 만들었고, 6번 라인에서는 A 버튼의 현재 상태(button A current)라는 의미의 btn_a_curr 라는 변수를 만들어 두 변수 모두 RELEASE(떨어짐) 으로 초기화하였습니다.

7번 라인에서는 A 버튼이 눌린 시간이라는 의미의 btn_a_pressed_time 이라는 변수를 만들었고, 8번 라인에서는 A 버튼이 떨어진 시간이라는 의미의 btn_a_release_time 이라는 변수를 만들어 0으로 초기화 하였습니다.

12번 라인에서는 btn_a_curr 변수에 A버튼이 눌렸는지를 저장하는데, 이 작업을 하기 직전 11번 라인에서 btn_a_prev 에 bun_a_curr 값을 백업합니다. 그 이유는 이전 상태와 현재 상태를 모두 보관하기 위함입니다.

14번 라인은 A버튼이 RELEASE 였다가 PRESSED로 바뀌었는지 확인하는 if문이고, 15번 라인은 반대로 PRESSED 였다가 RELEASE로 바뀌었는지 확인하는 if문입니다.

이제 프로그램을 실행시킨 뒤 에디터 하단의 

버튼을 눌러서 시리얼 창을 열어주세요. 이곳에 버튼이 눌린 시간이 출력됩니다.

3. 모스 부호 연습 기계 만들기

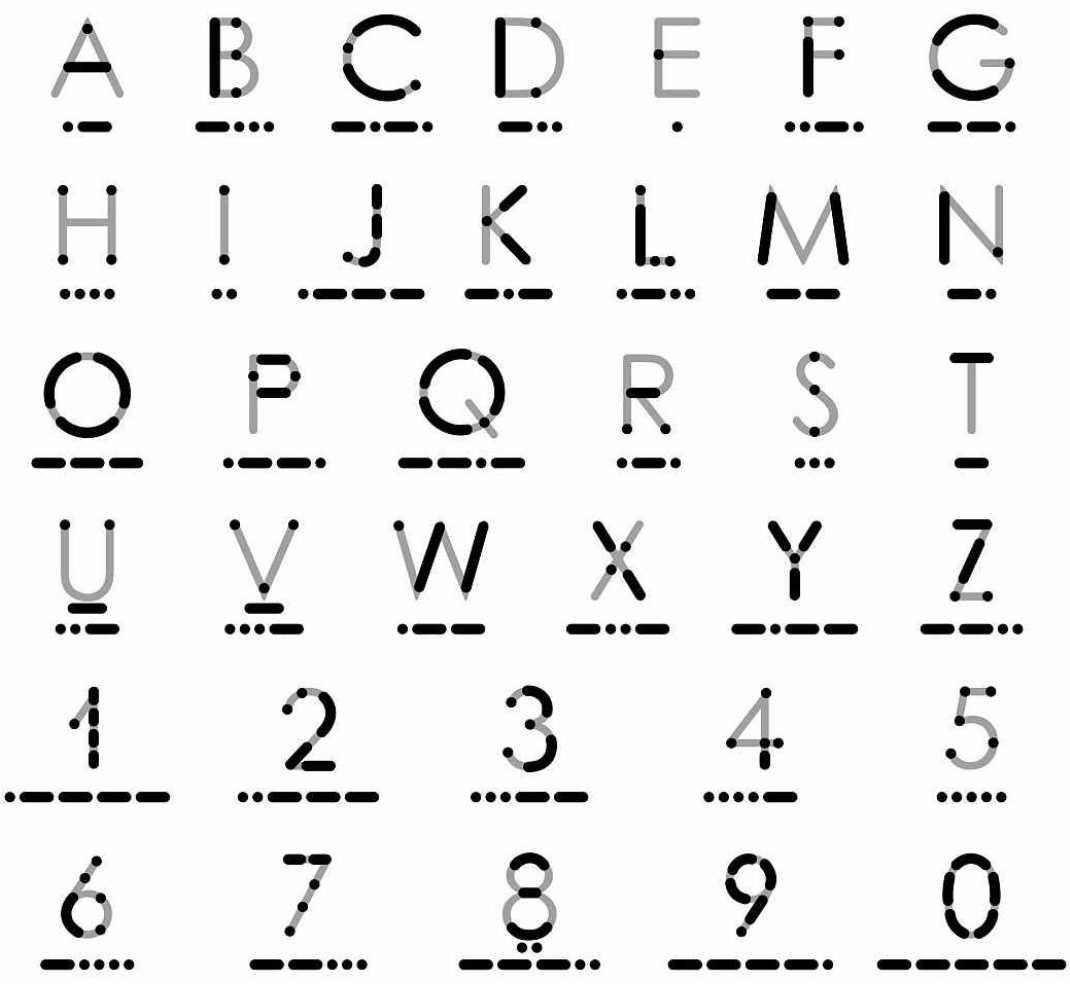
1) 모스 부호 개요

모스 부호(Morse code)는 한 종류의 신호 발생장치로 짧은 신호(·, 점 또는 단점)와 긴 신호(-, 선 또는 장점)를 적절히 조합하여 문자 기호를 표기하는 방식이다. 발명가 새뮤얼 핀리 브리즈 모스가 고안하였으며, 1844년 최초로 미국의 볼티모어와 워싱턴 D.C. 사이 전신 연락에 사용되었다.

2) 모스 부호(모스 코드)

신호는 소리, 빛, 전류 등으로 나타낼 수 있으며, 읽을 때에는 점/선이나 돈/쓰, 딧(dit)/다(dah)라고 부르기도 한다. 알파벳 A의 모스 부호는 · — (dit dah)인데, 막연하게 외우는건 너무 어렵기 때문에 아래와 같은 비주얼 가이드가 만들어졌다.

<모스 코드 비주얼 가이드>



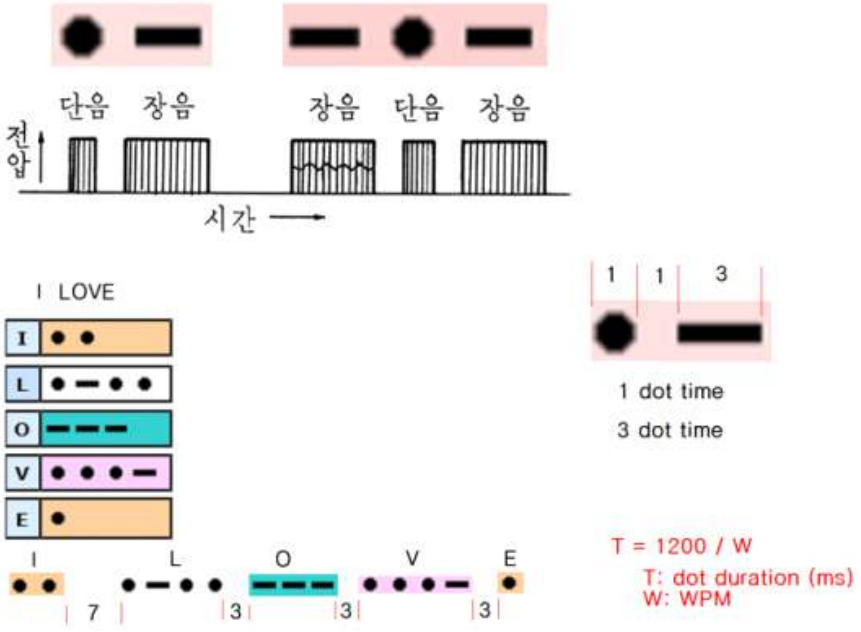
- 유튜브 URL: https://youtu.be/GeVdnd4_vyk

영문	모스부호	국문	모스부호	기호	모스부호
A	·–	ㄱ	·–··	1	·————
B	–···	ㄴ	··–·	2	··–––
C	–·–·	ㄷ	–···	3	···––
D	–··	ㄹ	···–·	4	····–
E	·	ㅁ	––	5	·····
F	··–·	ㅂ	·––	6	–····
G	––·	ㅅ	––·	7	––···
H	····	ㅇ	–·–	8	–––··
I	··	ㅈ	·––·	9	––––·
J	·–––	ㅊ	–·–·	0	–––––
K	–·–	ㅋ	––··	.	·–·–·–
L	·–··	ㅌ	––··	,	––··––
M	––	ㅍ	–––	:	–––···
N	–·	ㅎ	·–––	?	··––··
O	–––	ㅏ	·	-	–····–
P	·––·	ㅑ	··	_	··––·–
Q	––·–	ㅓ	–	(	–·––·
R	·–·	ㅕ	···	)	–·––·–
S	···	ㅗ	·–	/	–···–·
T	–	ㅛ	–·	"	·–··–
U	··–	ㅜ	····	!	··–··
V	···–	ㅠ	·–·	×	··–·–
W	·––	ㅡ	–··	+	·–·–·
X	–··–	ㅣ	··–	=	–···–
Y	–·––	ㅐ	–·––	;	–·–·–·
Z	––··	ㅒ	––·–	'	·––––––·

모스 부호는 점과 선의 시간 길이에 대해 아래와 같이 명확한 정의가 내려져 있습니다.

- 선(dash) 길이는 점(dot)의 3배일 것,
- 한 자(로마자 또는 한글자모 문자)를 형성하는 선과 점 사이 간격은 1점과 같을 것,
- 문자와 문자, 문자와 기호 사이 간격은 3점과 같을 것,
- 단어와 단어 사이는 7점과 같을 것.

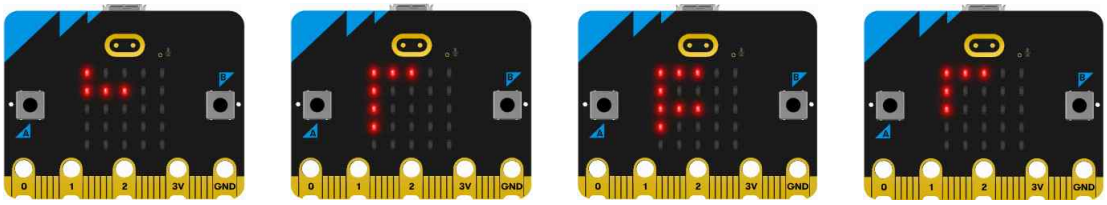
아래 그림에서 'I LOVE' 문자열을 모스 부호로 변환한 예시를 살펴보세요.



[신호의 시간 길이를 고려한 모스 코드 실제 사용 예]

3) 설계 목표

A: • —　　B: — • • •　　C: — • — •　　D: — • •



㈎ 0.1초(100ms)를 단음의 기준으로 정의한다. (장음은 단음의 3배이므로 300ms)

㈏ A 버튼으로 단음 또는 장음을 누르면 위 그림과 같이 LED 매트릭스에 표시되도록 한다.

㈐ 문자와 문자 사이의 길이를 0.7(단음의 7배이므로 700ms)초로 정하고 이 시간이 지나면 자동으로 LED화면을 지운다.

4) 실습예제1

button_morse_trainer.py

```
1  from microbit import *
2
3  PRESSED = True
4  RELEASE = False
5  btn_a_prev = RELEASE  # button A previous = 떨어짐
6  btn_a_curr = RELEASE  # button A current = 떨어짐
7  btn_a_pressed_time=0  # 버튼 누른 시간
8  btn_a_release_time=0  # 버튼 떨어진 시간
9  TICK = 100   # 단음(dot time)을 100ms로 정의함
10 cur_y = 0    # 단음 또는 장음을 표시할 y 좌표
11
12 while True:
13     btn_a_prev = btn_a_curr
14     btn_a_curr = button_a.is_pressed()
15
16     # 버튼을 계속 안누르고 있는 상태이면,
17     if btn_a_prev==RELEASE and btn_a_curr==RELEASE:
18         # TICK의 7배 시간(문자와 문자 사이의 길이)이 흐르면,
19         if(running_time() - btn_a_release_time >= TICK*7):
20             display.clear()  # 화면 모두 지움
21             cur_y = 0        # y좌표 0으로 리셋
22
23     if btn_a_prev==RELEASE and btn_a_curr==PRESSED: #버튼 누르면,
24         btn_a_pressed_time = running_time()
25
```

```
26      if btn_a_prev==PRESSED and btn_a_curr==RELEASE: #버튼 떨어지면,
27          btn_a_release_time = running_time()
28          pressed_ms = btn_a_release_time - btn_a_pressed_time
29          print(pressed_ms)
30
31          if cur_y<5:  # y좌표가 5미만일 때만 그린다.
32              dot_cnt = round(pressed_ms/TICK)  # 계산결과를 반올림
33              for x in range(dot_cnt):
34                  display.set_pixel(x, cur_y, 9)
35              cur_y = cur_y + 1
```

㈎ 기능을 9번 라인에서 정의하였습니다. 그리고 31~35번 라인이 ㈏ 기능을 구현하는 내용입니다. 31번 라인의 **cur_y**라는 변수는 마이크로비트의 LED 매트릭스에 출력을 해야 할 y좌표를 저장하는 변수입니다. 처음에는 0에서 시작하여 버튼이 입력될 때마다 다음 줄로 넘어가게 됩니다. 32번 라인의 round(x) 함수는 x의 반올림 결과는 구하는 함수입니다. 버튼을 오래 누를수록 dot_cnt는 큰 수가 됩니다. 33~34번 라인에서는 **display.set_pixel()** 함수로 dot_cnt 숫자만큼 점을 찍게 됩니다. 그 결과 A 버튼을 0.1초 가량 누르면 점이 출력되고, 0.3초 가량 누르면 선(점3개)가 출력됩니다.

㈐ 기능은 19~21번 라인에서 구현하였습니다.

5) 실습예제2: dot, dash 자동구분

dot_cnt 만큼 점을 찍으면, A 버튼을 0.2초, 0.4초, 0.5초를 누르는 경우 단음도 장음도 아닌 애매한 길이가 출력됩니다. (심지어 0.6초 이상을 누르면 에러가 남) 그래서 아래와 같은 규칙으로 무조건 단음(점1개) 또는 장음(점3개)로만 출력되도록 프로그램을 수정하고 싶습니다. 어떻게 하면 될까요? 여러분이 직접 만들어 보세요.

계산된 dot_cnt가 2이하이면 단음으로 처리하고 그 밖의 경우에는 장음으로 처리한다.

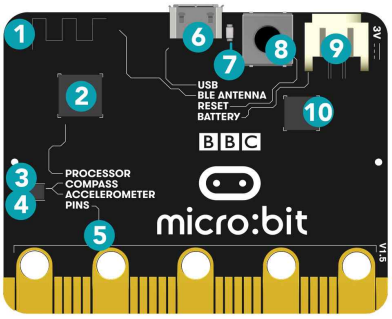
button_morse_trainer.py

```
31          if cur_y<5:  # y좌표가 5미만일 때만 그린다.
32              dot_cnt = round(pressed_ms/TICK)  # 계산결과를 반올림
33              dot_cnt = min(dot_cnt, 3)  # 최대 길이를 3개로 제한
34              if dot_cnt<=2:   # 만약 길이가 2라면,
35                  dot_cnt=1    # 길이를 1로 간주
36
37              for x in range(dot_cnt):
38                  display.set_pixel(x, cur_y, 9)
39              cur_y = cur_y + 1
```

정답은 위와 같습니다. 33~35번 라인만 추가하면 됩니다.

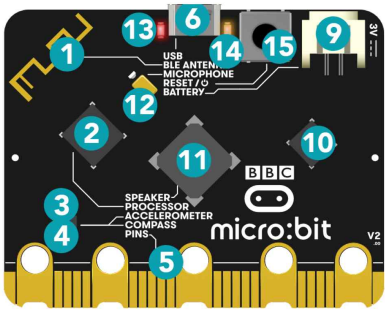
Part 04

응용 프로젝트



micro:bit V1 - Back

1. Radio & Bluetooth Antenna
2. Temperature Sensor & Processor
3. Compass
4. Accelerometer
5. Pins
6. Micro USB Socket
7. Single Red LED
8. Reset Button
9. Battery Socket
10. USB Interface Chip



micro:bit V2 - Back

1. Radio & Bluetooth Antenna
2. Temperature Sensor & Processor
3. Compass
4. Accelerometer
5. Pins
6. Micro USB Socket
9. Battery Socket
10. USB Interface Chip
11. Speaker
12. Microphone
13. Red Power LED
14. Yellow USB LED
15. Reset & Power button

<버전 별 마이크로비트 내장 모듈>

마이크로비트에는 여러 가지 모듈이 이미 내장(built in)된 상태에서 출시되었습니다. 내장 모듈을 활용한 마이크로비트의 능력을 알아봅시다.

가. 라디오 & 블루투스 안테나

- 무선으로 정보를 송·수신 할 수 있습니다.

나. 온도센서

- 프로세서에 내장된 온도센서를 통하여 주변 온도를 측정할 수 있습니다.

다. 자기 센서

- 자기력을 감지하는 홀센서가 내장되어 나침반 등을 만들 수 있습니다.

라. 가속도 센서

- 가속도 센서로 기울어짐, 회전, 제스처를 감지할 수 있습니다.

마. 스피커

- 내장 스피커로 소리를 만들어 낼 수 있습니다.

바. 마이크

- 마이크로 주변 소리를 감지할 수 있습니다.

사. 터치센서

- 로고 그림에 터치센서가 내장되어 로고의 터치를 감지할 수 있습니다.

1. 무선으로 모스 부호 송·수신하기

1) 설계 목표

3단원에서 만들었던 '모스 부호 연습 기계'를 업그레이드하여 아래 기능을 추가해 봅시다.

> ㈎ A버튼을 누르기 시작하는 순간부터 버튼이 떨어질까지 880Hz 음을 출력한다.
> ㈏ B 버튼을 누를 때마다가 송신 모드와 수신 모드가 변경되는데, 송신모드로 전환되면 '↑'를 출력하고, 수신모드로 전환되면 '↓'를 출력한다.
> ㈐ 송신모드에서는 입력되는 모스 부호를 다른 마이크로비트로 실시간 무선 전송한다.
> ㈑ 수신모드에서는 전송 받은 모스 부호를 출력한다.

2) 소리 출력 기능 추가

먼저 ㈎ 기능부터 추가해 보겠습니다. 아래 소스코드의 빨간색으로 표시된 2번, 26번, 30번 라인만 추가하면 소리 출력기능이 완성됩니다.

26번 라인의 music.pitch(n) 함수는 내장 스피커로 n Hz(헤르츠)의 음파를 만들어 음을 출력하는 함수입니다. 소리의 출력은 30번 라인의 music.stop() 함수를 만날 때 까지 지속됩니다.

project_morse_sound.py

```
1  from microbit import *
2  import music
3
4  PRESSED = True
5  RELEASE = False
6  btn_a_prev = RELEASE  # button A previous = 떨어짐
7  btn_a_curr = RELEASE  # button A current = 떨어짐
8  btn_a_pressed_time=0  # 버튼 누른 시간
9  btn_a_release_time=0  # 버튼 떨어진 시간
10 TICK = 100  # 단음(dot time)을 100ms로 정의함
11 cur_y = 0   # 단음 또는 장음을 표시할 y 좌표
12
13 while True:
14     btn_a_prev = btn_a_curr
15     btn_a_curr = button_a.is_pressed()
16
17     # 버튼을 계속 안누르고 있는 상태이면,
18     if btn_a_prev==RELEASE and btn_a_curr==RELEASE:
```

```
        # TICK의 7배 시간(문자와 문자 사이의 길이)이 흐르면,
        if(running_time() - btn_a_release_time >= TICK*7):
            display.clear()  # 화면 모두 지움
            cur_y = 0        # y좌표 리셋

    if btn_a_prev==RELEASE and btn_a_curr==PRESSED: #버튼 누르면,
        btn_a_pressed_time = running_time()
        music.pitch(880)  # 5옥타브 '라'음 출력 시작

    if btn_a_prev==PRESSED and btn_a_curr==RELEASE: #버튼 떨어지면,
        btn_a_release_time = running_time()
        music.stop()      # 소리 출력 중지
        pressed_ms = btn_a_release_time - btn_a_pressed_time
        print(pressed_ms)

        if cur_y<5:  # y좌표가 5미만일 때만 그린다.
            dot_cnt = round(pressed_ms/TICK)
            for x in range(dot_cnt):
                display.set_pixel(x, cur_y, 9)
            cur_y = cur_y + 1
```

3) 송 · 수신 모드 추가하기

이제 (나) 기능을 추가해봅시다. 13번 라인부터 시작하여 빨간색으로 표시된 코드를 추가하면 완성됩니다.

project_morse_radio.py

```
is_send_mode = True   # 이 변수가 True 이면 송신모드임

while True:
    if button_b.was_pressed(): # B버튼이 눌렸으면,
        is_send_mode = not is_send_mode  # 모드를 반전
        if is_send_mode:   # 송신모드이면,
            display.show(Image.ARROW_N)   # ↑ 이미지 출력
        else:
            display.show(Image.ARROW_S)   # ↓ 이미지 출력
        sleep(TICK*2)
```

4) 무선 송수신 기능 구현하기

(다), (라) 기능을 추가한 최종 완성 코드는 아래와 같습니다. 빨색으로 표시된 코드가 추가된 부분이며 나머지 부분은 동일합니다.

project_morse_radio_updated.py

```
1  from microbit import *
2  import music
3  import radio   # 무선 송수신 모듈 추가
4
5  PRESSED = True
6  RELEASE = False
7  btn_a_prev = RELEASE  # button A previous = 떨어짐
8  btn_a_curr = RELEASE  # button A current = 떨어짐
9  btn_a_pressed_time=0  # 버튼 누른 시간
10 btn_a_release_time=0  # 버튼 떨어진 시간
11 TICK = 100            # 단음(dot time)을 100ms로 정의함
12 cur_y = 0             # 단음 또는 장음을 표시할 y 좌표
13 is_send_mode = True   # 이 변수가 True 이면 송신모드임
14
15 radio.config(group=1)  # 라디오그룹을 1로 설정
16 radio.on()             # 무선 송수신 기능 켜기
17
18 while True:
19     if button_b.was_pressed(): # B버튼이 눌렸으면,
20         is_send_mode = not is_send_mode  # 모드를 반전
21         if is_send_mode:   # 송신모드이면,
22             display.show(Image.ARROW_N)   # ↑ 이미지 출력
23         else:
24             display.show(Image.ARROW_S)   # ↓ 이미지 출력
25         sleep(TICK*2)
26
27     btn_a_prev = btn_a_curr
28     if is_send_mode:   # 송신모드이면,
29         btn_a_curr = button_a.is_pressed()
30     else:   # 수신모드이면,
31         message = radio.receive()   # 무선으로 전송된 내용 수신
32         if message != None:     # 메시지가 없는게 아니라면
33             print(message)
34             if message=='pressed':    # 수신메시지가 'pressed'이면,
35                 btn_a_curr = PRESSED  # 버튼 현재 상태 눌림으로 변경
36             elif message=='release':  # 수신메시지가 'release'이면,
37                 btn_a_curr = RELEASE  # 버튼 현재 상태 떨어짐으로 변경
38
39     if btn_a_prev==RELEASE and btn_a_curr==RELEASE:
40         if(running_time() - btn_a_release_time >= TICK*7):
```

```
41              display.clear()  # 화면 모두 지움
42              cur_y = 0        # y좌표 0으로 리셋
43
44      if btn_a_prev==RELEASE and btn_a_curr==PRESSED:
45          btn_a_pressed_time = running_time()
46          music.pitch(880)
47          if is_send_mode:  # 송신모드이면,
48              radio.send('pressed')   # 'pressed'라는 문자열 송신
49
50      if btn_a_prev==PRESSED and btn_a_curr==RELEASE:
51          btn_a_release_time = running_time()
52          music.stop()
53          if is_send_mode:  # 송신모드이면,
54              radio.send('release')  # 'release'라는 문자열 송신
55
56          pressed_ms = btn_a_release_time - btn_a_pressed_time
57          print(pressed_ms)
58
59          if cur_y<5:  # y좌표가 5미만일 때만 그린다.
60              dot_cnt = round(pressed_ms/TICK)
61              dot_cnt = min(dot_cnt, 3)
62              if dot_cnt<=2:
63                  dot_cnt=1
64
65              for x in range(dot_cnt):
66                  display.set_pixel(x, cur_y, 9)
67              cur_y = cur_y + 1
```

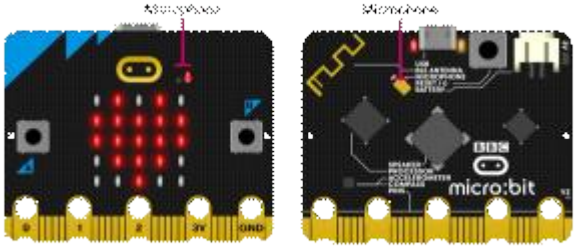
15번 라인에서 `radio.config(group=1)`로 radio 그룹을 설정하고 있습니다. 무선 송수신 기능은 radio 그룹이 동일한 기기들 사이에서만 효과가 있습니다. 그렇기 때문에 송신기와 수신기에는 동일한 radio 그룹이 설정되어야 합니다. group 번호는 0부터 255사이의 숫자를 사용할 수 있습니다. 교실의 전체 인원이 모두 group=1을 사용한다면 다수가 송신자가 되어 몹시 혼란스러울 수 있으니 2인 1조가 되어 둘이서만 같은 group 번호를 설정하여 실습을 진행하는 것이 좋습니다.

송신모드에서는 '모스 부호 연습기계' 때와 동일하게 작동하므로 47~48번, 53~54번 라인의 무선 송신 코드만 추가되었습니다. 그리고 30~37번 라인은 수신모드를 위한 코드들입니다. 코드 내용은 무선으로 수신한 메시지가 있다면 메시지 내용에 따라 A 버튼을 누르거나 땐 것으로 에뮬레이션[3]하는 것이지요.

3) 마치 버튼이 눌리거나 떨어진 것처럼 소프트웨어적으로 흉내냄.

2. 마이크로 소리에 반응하기

1) 개요



마이크로비트 v2에는 마이크가 내장되어 있습니다. 뒷면에 위치하고 있으며, 전면 터치 로고 오른쪽에 소리가 들어올 수 있는 구멍이 있습니다. 마이크 사용 시에는 전면 마이크 LED에 불이 들어옵니다.

2) 실습예제: 큰소리에 반응하기

박수를 칠 때 마다 ↑, ↓ 화살표를 번갈아 출력하는 프로그램을 만들어 봅시다.
파이썬 에디터에서 새로운 프로젝트를 생성한 뒤 아래와 같이 코딩합니다.

project_mic_loud.py

```
1  from microbit import *
2  image = Image.ARROW_N   # ↑로 초기화
3  while True:
4      if image == Image.ARROW_N:  # ↑ 이면
5          image = Image.ARROW_S   # ↓ 대입
6      else:
7          image = Image.ARROW_N   # ↑ 대입
8
9      if microphone.current_event() == SoundEvent.LOUD:
10         display.show(image)
11         sleep(100)
```

3) 실습예제: 사운드 레벨 측정

project_mic_level.py

```
1  from microbit import *
2  image = Image.ARROW_N
3  levels = [0,0,0,0,0]
4  while True:
5      for x in range(5):
6          levels[x] = round(microphone.sound_level()/51)
7
8          for y in range(levels[x]):  #사운드 레벨을 막대그래프로 출력
9              display.set_pixel(x, 4-y, 9)
10         sleep(20)
11     display.clear()
```

3. 스피커로 출력하기

1) 효과음 출력

마이크로비트에는 다음과 같이 Sound 객체의 상수로 정의된 여러 가지 종류의 효과음이 내장되어 있습니다. 소리을 글로 표현하기 어렵기 때문에 어떤 효과음인지 아래 프로그램을 만들어 직접 들어보도록 하겠습니다.

project_output_sound.py

```
1  from microbit import *
2  import music
3
4  sound_list = [Sound.GIGGLE, Sound.HAPPY, Sound.HELLO,
5      Sound.MYSTERIOUS, Sound.SAD, Sound.SLIDE, Sound.SOARING,
6      Sound.SPRING, Sound.TWINKLE, Sound.YAWN]
7
8  sound_index = 0
9
10 while True:
11     if button_a.was_pressed():  # A버튼이 눌렸으면,
12         print(sound_index, sound_list[sound_index])
13         audio.play(sound_list[sound_index], wait=False)
14         # 마지막 인덱스에 도달하면 다시 0번으로 되돌리기 위함
15         sound_index = (sound_index+1) % len(sound_list)
```

A 버튼을 누를 때마다 새로운 다음 효과음을 출력하게 됩니다.

2) 내장 멜로디

마이크로비트에는 다음과 같이 music 객체의 상수로 정의된 여러 가지 종류의 음악이 내장되어 있습니다.

- DADADADUM- 베토벤 교향곡 5번 C단조의 오프닝.
- ENTERTAINER- Scott Joplin의 Ragtime 클래식 "The Entertainer"의 오프닝 단편.
- PRELUDE- JSBach의 48개 전주곡과 푸가 중 첫 번째 전주곡 C장조 오프닝.
- ODE- 베토벤 교향곡 9번 D단조의 "환희의 송가" 주제.
- NYAN- 냥 고양이 테마(http://www.nyan.cat/). 작곡가는 알 수 없습니다. 이것은 교육용 돌고래를 위한 공정 사용입니다(뉴욕에서 말하는 것처럼).
- RINGTONE- 휴대폰 벨소리. 들어오는 메시지를 나타내는 데 사용됩니다.
- FUNK- 비밀 요원과 범죄 배후를 위한 펑키한 베이스 라인.
- BLUES- 부기우기 12마디 블루스 워킹 베이스.

- BIRTHDAY- "Happy Birthday to You…"
- WEDDING- 바그너의 오페라 "로엔그린"의 신부 합창.
- FUNERAL- 프레데릭 쇼팽의 피아노 소나타 2번 B♭ 단조 Op. 35.
- PUNCHLINE- 농담을 의미하는 재미있는 조각이 만들어졌습니다.
- PYTHON- John Philip Sousa의 행진 "Liberty Bell" 일명 "Monty Python's Flying Circus"의 주제
- BADDY- 악역의 무성영화 시대 등장.
- CHASE- 무성 영화 시대 추격 장면.
- BA_DING- 어떤 일이 발생했음을 나타내는 짧은 신호.
- WAWAWAWAA- 매우 슬픈 트롬본.
- JUMP_UP- 게임에서 사용하기 위해 상향 이동을 나타냅니다.
- JUMP_DOWN- 게임에서 사용하기 위해 하향 이동을 나타냅니다.
- POWER_UP- 잠금 해제된 업적을 알리는 팡파르.
- POWER_DOWN- 업적을 잃었음을 알리는 슬픈 팡파르.

음악을 글로 표현하기 어렵기 때문에 어떤 음악인지 아래 프로그램을 만들어 직접 들어보도록 하겠습니다.

project_output_sound.py

```
1  from microbit import *
2  import music
3
4  music_list = [music.BA_DING, music.BADDY, music.BIRTHDAY,
5      music.BLUES, music.CHASE, music.DADADADUM, music.ENTERTAINER,
6      music.FUNERAL, music.FUNK, music.JUMP_DOWN, music.JUMP_UP,
7      music.NYAN, music.ODE, music.POWER_DOWN, music.POWER_UP,
8      music.PUNCHLINE, music.PYTHON, music.RINGTONE,
9      music.WAWAWAWAA, music.WEDDING]
10
11 music_index = 0
12
13 while True:
14     if button_b.was_pressed():  # B버튼이 눌렸으면,
15         print(music_index, music_list[music_index])
16         music.play(music_list[music_index], wait=False)
17         music_index = (music_index+1) % len(music_list)
```

B 버튼을 누를 때마다 새로운 다음 음악을 출력하게 됩니다.

3) 악보 연주하기

우리가 직접 연주하고 싶은 음악을 악보로 표현하여 연주하는 것도 가능합니다.
악보 표기에 사용하는 음표를 다음과 형식으로 지정할 수 있습니다.

```
음[옥타브][:길이]
```

예를 들어 'A1:4' 는 4틱 동안 지속되는 1옥타브 A(라)음표를 나타냅니다(틱은 템포 설정 기능에 의해 정의된 임의의 시간 길이입니다). 음표이 'R'이 사용되면 쉼표(묵음)로 처리됩니다. 음표는 대소문자를 구분하지 않습니다. 옥타브 및 지속시간은 후속 음표로 전달이 됩니다. 기본 값은 4옥타브, 지속시간은 4틱(4분음표)입니다.
예를 들어 베토벤 교향곡 5번의 오프닝을 다음과 같이 나타낼 수 있습니다.

```
['r4:2', 'g', 'g', 'g', 'eb:8', 'r:2', 'f', 'f', 'f', 'd:8']
```

project_output_music_prelude.py

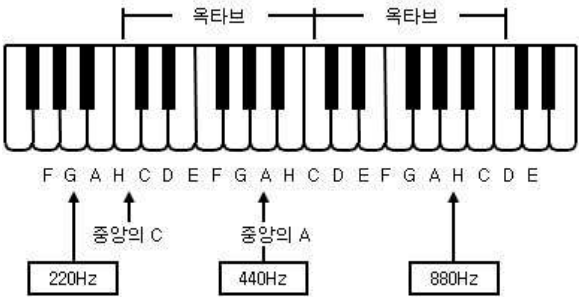
```
notes = [
    'c4:1', 'e', 'g', 'c5', 'e5', 'g4', 'c5', 'e5',
    'c4', 'e', 'g', 'c5', 'e5', 'g4', 'c5', 'e5',
    'c4', 'd', 'a', 'd5', 'f5', 'a4', 'd5', 'f5',
    'c4', 'd', 'a', 'd5', 'f5', 'a4', 'd5', 'f5',
    'b3', 'd4', 'g', 'd5', 'f5', 'g4', 'd5',
    'f5', 'b3', 'd4', 'g', 'd5', 'f5', 'g4', 'd5', 'f5',
    'c4', 'e', 'g', 'c5', 'e5', 'g4', 'c5', 'e5',
    'c4', 'e', 'g', 'c5', 'e5', 'g4', 'c5', 'e5',
    'c4', 'e', 'a', 'e5', 'a5', 'a4', 'e5', 'a5',
    'c4', 'e', 'a', 'e5', 'a5', 'a4', 'e5', 'a5',
    'c4', 'd', 'f#', 'a', 'd5', 'f#4', 'a', 'd5',
    'c4', 'd', 'f#', 'a', 'd5', 'f#4', 'a', 'd5',
    'b3', 'd4', 'g', 'd5', 'g5', 'g4', 'd5', 'g5',
    'b3', 'd4', 'g', 'd5', 'g5', 'g4', 'd5', 'g5',
    'b3', 'c4', 'e', 'g', 'c5', 'e4', 'g', 'c5',
    'b3', 'c4', 'e', 'g', 'c5', 'e4', 'g', 'c5',
    'a3', 'c4', 'e', 'g', 'c5', 'e4', 'g', 'c5',
    'a3', 'c4', 'e', 'g', 'c5', 'e4', 'g', 'c5',
    'd3', 'a', 'd4', 'f#', 'c5', 'd4', 'f#', 'c5',
    'd3', 'a', 'd4', 'f#', 'c5', 'd4', 'f#', 'c5',
    'g3', 'b', 'd4', 'g', 'b', 'd', 'g', 'b',
    'g3', 'b3', 'd4', 'g', 'b', 'd', 'g', 'b'
]

music.play(notes)
```

4. 피아노 만들기

1) 음파

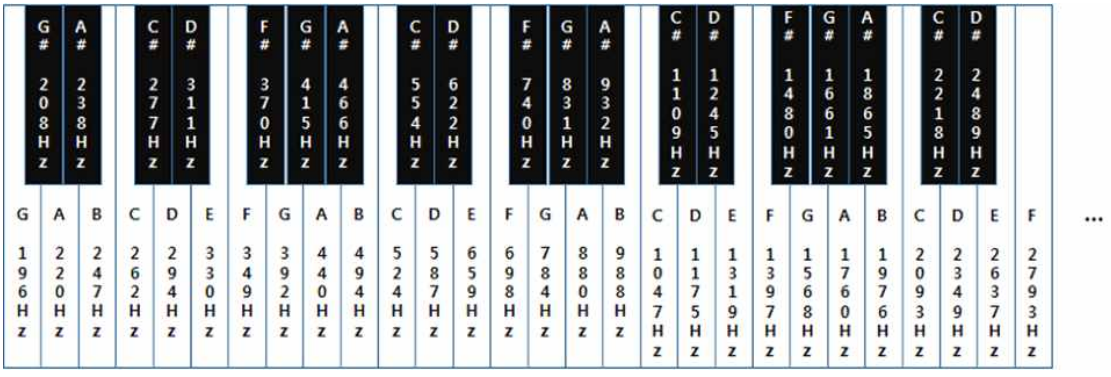
우리들의 귀에 끊임없이 들려오는 소리는 공기 속을 전해오는 파동입니다. 사람이 듣는 소리는 공기의 진동을 감지하는 것이고 진동수(주파수)에 따라 낮은음과 높은 음으로 구별되는 것입니다. 그렇다면, 만약 우리가 특정 주파수의 진동을 스피커로 만들어 낼 수 있다면 원하는 음을 출력할 수 있지 않을까요? 이번 프로젝트는 이 아이디어에서 출발합니다. 실제로 마이크로비트의 `music.pitch()` 함수로 특정 주파수의 음을 만들어 낼 수 있습니다.

2) 음과 주파수



옥타브	음계	주파수	계산
4	라	440	$440 * 2^{0/12}$
4	라#	466	$440 * 2^{1/12}$
4	시	494	$440 * 2^{2/12}$
4	도	523	$440 * 2^{3/12}$
4	도#	554	$440 * 2^{4/12}$
4	레	587	$440 * 2^{5/12}$
4	레#	622	$440 * 2^{6/12}$
4	미	659	$440 * 2^{7/12}$
4	파	698	$440 * 2^{8/12}$
4	파#	740	$440 * 2^{9/12}$
4	솔	784	$440 * 2^{10/12}$
4	솔#	831	$440 * 2^{11/12}$
5	라	880	$440 * 2^{12/12}$

국제 표준화 기구(ISO)에서는 4옥타브 '라' 음을 440Hz로, 주파수가 2배인 음의 폭을 '옥타브'로 정의하고 있습니다. 그래서 5옥타브 '라'음의 주파수는 440의 두 배인 880Hz입니다. 추가하여 한 옥타브 구간의 주파수를 동일 비율 간격으로 12분할하여 12음계에 배당하고 있습니다. 그 결과 위 그림과 같은 계산을 통해 음계별 주파수를 계산할 수 있으며 계산 결과는 아래 그림과 같습니다.



4) 설계 목표

우리는 위에서 살펴본 음계별 주파수 계산 방법을 바탕으로 피아노를 만들어 보고자 합니다. 입력 장치로 키보드를 사용하는데 아래 그림과 같이 키와 피아노 건반을 매칭 합니다. 키를 누르면 최대 1초 동안 해당 음을 출력하고, 그 전에 다른 키를 누르면 지금 출력하던 소리를 정지시키고 누른 키에 해당하는 음을 출력합니다.

5) 실습 예제

project_piano.py

```
1  from microbit import *
2  import music
3
4  note_pitch = []       # 음별 주파수를 저장하는 리스트
5  last_play_time = 0    # 마지막 소리 출력 시각
6  key_sequence = "q2we4r5ty7u8i9op-[=]"  # 건반에 해당하는 키 순서
7
8  for n in range(len(key_sequence)):
9      pitch = int(440*pow(2, n/12))  # 음 계산 공식에 따라 주파수 계산
10     print(pitch)                   # 계산 결과를 시리얼로 출력
11     note_pitch.append(pitch)       # 계산 결과를 리스트에 삽입
12
13 while(True):
14     byte = uart.read(1)      # 시리얼 장치에서 한 글자 읽기
15     if byte is not None:     # 읽은 내용이 None(없음) 아니라면,
16         key = byte.decode()  # 읽은 내용을 파이썬 문자로 변환
17         key = key.lower()    # 내용을 소문자로 변환
18
19         position = key_sequence.find(key) # 키가 몇 번째 인지 탐색
20         if position != -1:   # -1 이 아니라면(즉 찾았다면),
21             print(note_pitch[position], end=' ')
```

```
22              music.stop() # 이전에 출력하던 소리 정리
23              sleep(50)    # 50ms 잠시 정지
24              music.pitch(note_pitch[position]) # 키에 해당하는 소리
25              last_play_time = running_time()   # 현재 시간 저장
26          else:               # 건반이 아닌 키 입력되면,
27              music.stop()    #소리 출력 정지
28      else:  # 시리얼 장치에서 입력되는 내용이 없다면,
29          play_time = running_time()-last_play_time
30          if(play_time > 1000):  # 1초 이상 소리를 출력했다면,
31              music.stop()       # 소리 출력 정지
```

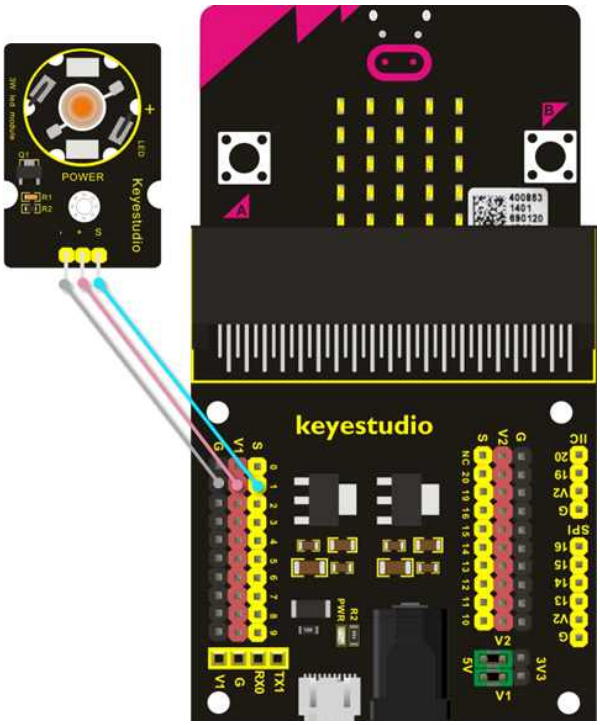
4번 라인의 note_pitch 는 음별 주파수를 저장하는 리스트입니다.
6번 라인의 key_sequence 변수는 피아노 건반에 해당하는 키들을 담는 변수입니다. 파이노의 12음계가 '라', '라#', '시', '도', '도#', '레', '레#', '미', '파', '파#', '솔', '솔#' 와 같은 순서로 배치되므로 그에 해당하는 문자들을 순서대로 연결하여 `"q2we4r5ty7u89op-[=]"`로 지정하였습니다.
9번 라인에서 음에 해당하는 주파수를 계산합니다. `pow(a, b)` 함수는 a^b를 계산하는 함수이고, `int(n)` 함수는 n을 소수점 내림하여 정수로 만들어 줍니다. 최종 계산 결과는 pitch 변수에 저장되고, 10번 라인에서 그 값을 출력한 뒤 11번 라인에서는 결과값을 note_pitch 리스트에 추가해 넣습니다.
14번 라인의 uart는 시리얼 인터페이스를 통한 통신에 사용하는 객체입니다. 시리얼 인터페이스란 컴퓨터와 연결된 마이크로비트의 통신장치를 말합니다. 이 통신장치를 통해 컴퓨터는 마이크로비트와 신호를 주고받는 것이 가능합니다. `uart.read()` 함수로 시리얼 인터페이스에서 데이터를 읽어 올 수 있습니다.
15번 라인의 if문으로 인해 읽은 내용(키보드 입력)이 있다면 16번 라인으로 진행되고 없다면 28번 라인으로 제어가 이동합니다.
16번 라인에서는 시리얼 장치에서 입력된 내용을 파이썬에서 처리 가능한 문자로 변환하고, 17번 라인에서 그 문자를 소문자로 바꿉니다. key_sequence 변수에 소문자 기준으로 입력해 놓았기 때문에 대문자가 입력된 경우 소문자로 바꿔주는 것입니다.
19번 라인에서는 입력된 글자가 key_sequence 변수의 몇 번째 글자인지 알아냅니다. 이렇게 함으로써 `note_pitch[position]` 으로 해당 키(건반)에 대한 주파수를 간편하게 알아낼 수 있습니다. 만약 key_sequence 에 없는 글자라면 26번 라인으로 이동하게 되고 소리의 출력을 정지시킵니다.
30번 라인에서는 소리 출력시간이 1000ms(1초) 이상인지를 확인하고 그 경우에는 소리 출력을 그만 중단하도록 합니다. 이로서 모든 설계 목표가 달성되었습니다.^^

5. 자동 ON/OFF 가로등 만들기

1) 설계 목표

주변 밝기를 측정하여 어두우면 불을 켜고 밝으면 불을 켜는 자동 ON/OFF를 가로등을 만들어 보자.

2) 제작 실습



앞 단원에서 우리는 LED 매트릭스에 글자를 적거나 그림을 그리는 등 출력의 용도로 사용하였다. 하지만 이 LED 장치에는 예상과는 달리 주변 밝기를 감지하는 기능이 내장되어 있다.

`display.read_light_level()` 함수를 이용하여 주변 밝기를 0~255 값으로 읽어 올 수 있다. 아래 예제를 실행시킨 뒤, 휴대폰의 손전등 기능을 이용하여 마이크로비트를 비추어 시리얼 창에 출력되는 변화를 관찰해 보자. 물론 왼쪽과 같이 결선까지 완료하면 가로등이 완성된다.

project_output_music_prelude.py

```
1  from microbit import *
2  import music
3
4  while(True):
5      brightness = display.read_light_level()
6
7      print(brightness)
8      if(brightness < 100):
9          pin1.write_digital(1)
10     else:
11         pin1.write_digital(0)
12     sleep(500);
```

5. 문 열림 경보기 만들기

1) 설계 목표

문 열림 경보기는 닫혀 있어야 문이 열리게 되면 알람을 울리는 장치입니다. 여러분의 예상대로 방범 용도로 많이 활용되고 있습니다. 문이 열리게 되면 싸이렌을 울린다든지 자동으로 신고 전화가 걸린다든지 여러 가지 대응 방안이 있을 수 있겠습니다만 우리는 시끄러운 알람 소리를 만들어 내 보도록 해 보겠습니다

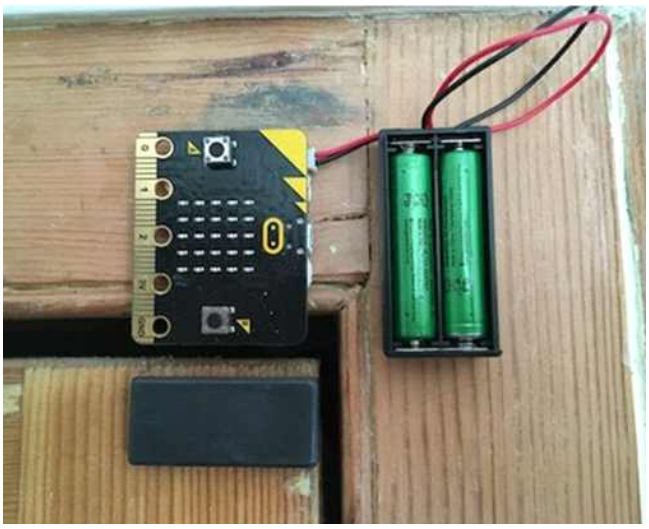
2) 제작 실습

강력자석 옆에서 자기장이 얼마나 측정되는지 그리고 자석 밖으로 벗어나면 자기장 수치가 어떻게 변하는지 먼저 확인할 필요가 있습니다.

project_output_security_lock.py

```
1  from microbit import *
2  import music
3  while(True):
4      magnet_strength_x = compass.get_x()
5      print(magnet_strength_x)
6      if(magnet_strength_x < 0):
7          music.play(['c4', 'c5'], wait=True, loop=False);
8      else:
9          sleep(500);
```

3) 설치 예시



<설치 예1>

<설치 예2>

2번 설치 방법은 문이 계속 열고 닫힐 때마다 마이크로비트가 충격을 받게 되므로 1번 설치 방법이 더 좋겠네요.