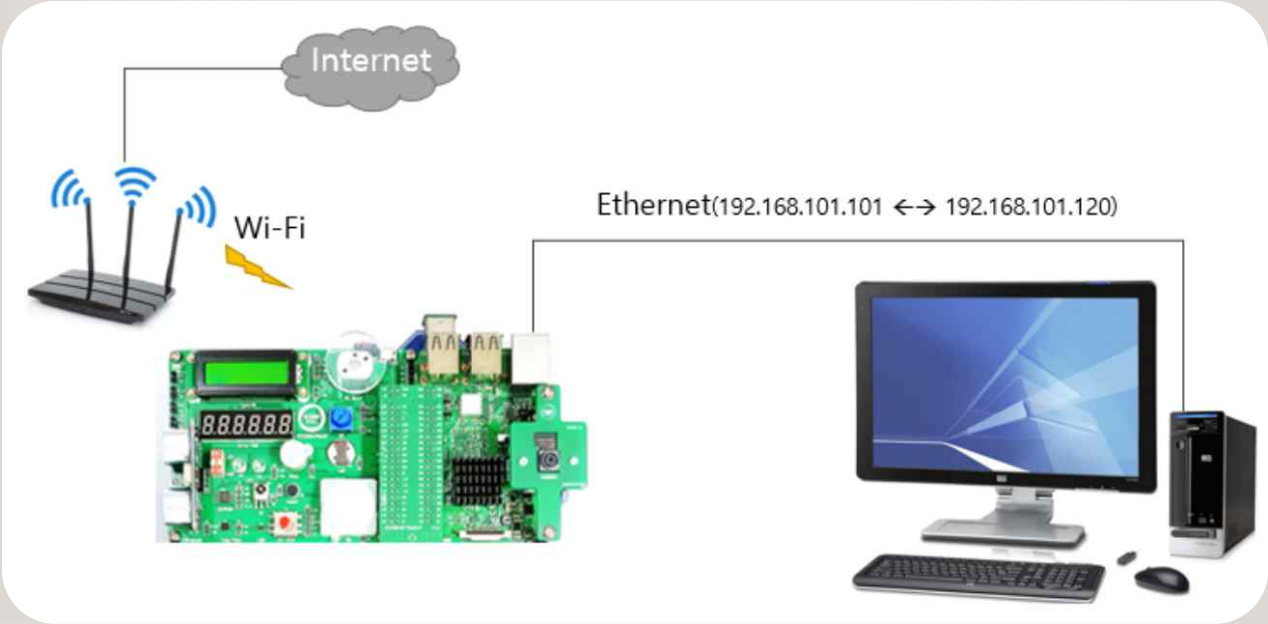
iCORE-SDP와 Python3로 배우는 IoT 센서 제어

실습 환경
iCORE-SDP

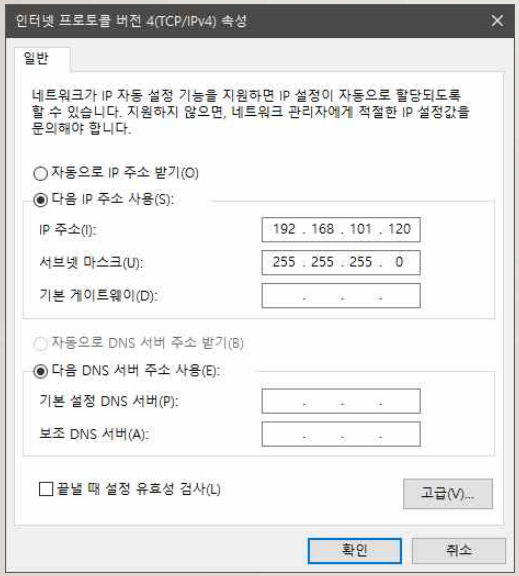
라즈베리파이와 IoT 센서 노드

- 라즈베리파이 기반의 IoT 센서 노드는 리눅스 운영체제를 사용하므로 확장성이 매우 뛰어납니다.
- 파이썬을 기본 지원하므로 C/C++ 외에 파이썬으로도 하드웨어 제어 프로그램을 만들 수 있습니다.
- 리눅스 운영체제가 제공하는 수 많은 기능들을 재사용해 빠른 시간에 원하는 결과를 얻을 수 있습니다.

실습 01

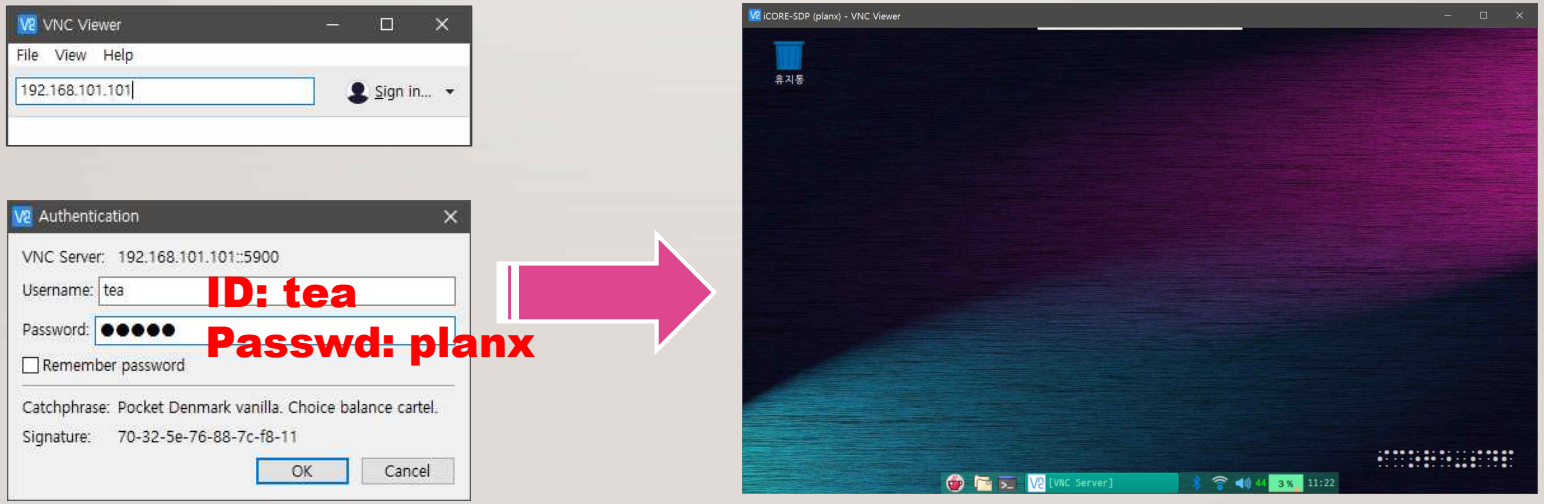


실습 2


인터넷 프로토콜 버전 4(TCP/IPv4) 속성
일반
네트워크가 IP 자동 설정 기능을 지원하면 IP 설정이 자동으로 할당되도록 할 수 있습니다. 지원하지 않으면, 네트워크 관리자에게 적절한 IP 설정값을 문의해야 합니다.
자동으로 IP 주소 받기(O)
다음 IP 주소 사용(S):
IP 주소(I): 192 . 168 . 101 . 120
서브넷 마스크(U): 255 . 255 . 255 . 0
기본 게이트웨이(D):
자동으로 DNS 서버 주소 받기(B)
다음 DNS 서버 주소 사용(E):
기본 설정 DNS 서버(P):
보조 DNS 서버(A):
끝낼 때 설정 유효성 검사(L)
고급(V)...
확인
취소

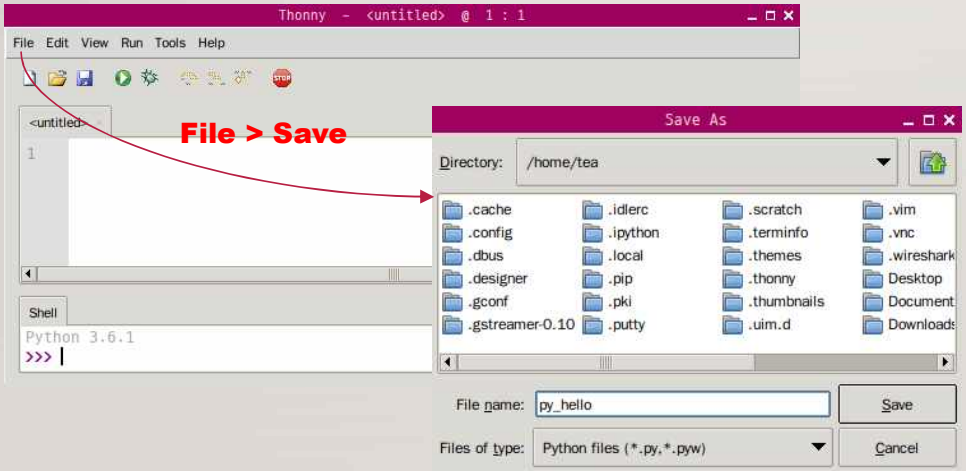
실습 3

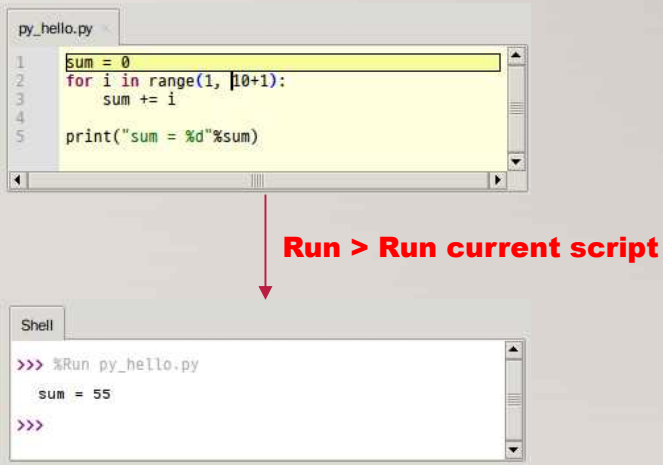
https://www.realvnc.com/download/viewer



실습 4

패널 > 시작 메뉴 > 개발 > 토니





Python3 101

HELLO WORLD!

- 파이썬은 매우 간단한 언어이며 매우 간단한 구문을 사용합니다.
- 보편적으로 준비된 코드 없이 바로 프로그래밍할 수 있습니다.
- 파이썬2와 파이썬3는 완전히 다릅니다. 튜토리얼은 의미론적으로 더 정확하고 새로운 기능을 지원하는 파이썬3를 사용합니다.

실습 05

파이썬3에서 문자열을 출력하려면 다음과 같이 작성합니다.

```
print("This line will be printed.")
```

파이썬은 중괄호 대신 들여 쓰기를 블록으로 사용합니다.

```
x = 1
if x == 1:
    # indented four spaces
    print("x is 1.")
```

변수와 타입

- 파이썬은 완전한 객체지향 언어로 동적인 타입을 사용합니다.
- 변수를 사용하기 전에 선언할 필요가 없습니다.
- 파이썬의 모든 변수는 하나의 객체입니다.
- 숫자 타입은 정수와 실수가 있으며 문자열은 작은 따옴표 또는 큰 따옴표로 묶어 표현합니다.

실습 02

변수가 정수나 실수를 가리키도록 하는 것은 쉽습니다.

```
myint = 7
print(myint)

myfloat = 7.0
print(myfloat)
myfloat = float(7)
print(myfloat)
```

변수는 문자열도 직관적으로 가리킬 수 있습니다.

```
mystring = 'hello'
print(mystring)
mystring = "Dot't worry about ..."
print(mystring)
```

실습 03

아래 코드를 바꾸세요

```
mystring = None
myfloat = None
myint = None
```

테스팅 코드입니다.

```
if mystring == "hello":
    print("String: %s" % mystring)
if isinstance(myfloat, float) and myfloat == 10.0:
    print("Float: %f" % myfloat)
if isinstance(myint, int) and myint == 20:
    print("Integer: %d" % myint)
```

리스트

- 리스트는 배열과 매우 유사합니다.
- 변수는 모든 타입의 변수를 가리킬 수 있으며 원하는 만큼의 변수를 포함할 수 있습니다.
- 리스트 매우 간단한 방식으로 반복 될 수 있습니다.
- 리스트의 요소는 인덱스로 접근하며 존재하지 않은 인덱스를 사용하면 예외가 발생합니다.

실습 04

리스트의 인덱스는 0부터 시작합니다.

```
mylist = []
mylist.append(1)
mylist.append(2)
mylist.append(3)
print(mylist[0], mylist[1], mylist[2])
for x in mylist:
    print(x)
```

존재하지 않는 인덱스를 사용하면 예외가 발생합니다.

```
mylist = [1, 2, 3]
print(mylist[3])
```

실습 05

리스트를 결합해 새로운 리스트를 만들 수 있습니다.

```
number = [1, 2, 3]
strings = ['apple', 'banana', 'pi']

print(number)
print(strings)
print("total = %s"%(number + strings))
```

리스트에 리스트를 포함할 수도 있습니다.

```
table = [[1, 2, 3], [4, 5, 6]]
print(table[0], table[0][1])
table[1][2] = 10
for row in table:
    for column in row:
        print(column, end = ' ')
    print()
```

기본 연산자

- +(덧셈), -(뺄셈), *(곱셈), /(실수 몫), //(정수 몫), %(나머지), **(제곱)
- 우선순위는 수학 모델을 따릅니다.
- +는 문자열이나 리스트를 결합해 새로운 문자열이나 리스트를 만듭니다.
- *는 문자열이나 리스트를 반복합니다.

실습 06

연산자의 우선순위는 수학 모델을 따릅니다.

```
number = 1 + 2 * 3 / 4.0
print(number)
```

나머지 연산자는 정수의 나머지를 반환합니다.

```
remainder = 11 % 3
print(remainder)
```

실습 07

두 개의 곱셈 심볼은 제곱을 반환합니다.

```
squared = 7 ** 2
cubed = 2 ** 3

print(squared)
print(cubed)
```

결과는 얼마일까요?

```
question = 1 / (2 * 4) ** 2 + 7 % 4

print(question)
```

실습 08

덧셈 심볼은 문자열이나 리스트를 결합합니다.

```
helloworld = 'hello' + "world"
print(helloworld)

even_num = [1, 11, 111]
odd_num = [2, 22, 222]
all_num = even_num + odd_num
print(all_num)
```

곱셈 심볼은 문자열이나 리스트를 반복합니다.

```
lotsofhellos = "hello" * 10
print(lotsofhellos)

ref = [1, [2, 3]] * 3
print(ref)
ref[1][0] = 20
ref[1][1] = 30
print(ref)
```

문자열 포매팅

- 파이썬은 C 스타일의 문자열 포맷을 사용해 새로운 형식의 문자열을 만듭니다.
- % 심볼은 튜플(고정 크기 리스트)로 묶인 변수 집합을 형식 문자열과 함께 형식화하는데 사용됩니다.
- 형식 문자열은 “%s”, “%d”, “%f”, “%.<자리수>f”, “%x” 와 같은 특수 기호를 포함합니다.

실습 09

형식 지정자를 이용하면 변수를 원하는 형태로 출력할 수 있습니다.

```
name = "chanmin"
print("Hello, %s!"%name)
```

인수가 2개 이상일 때는 튜플을 사용합니다.

```
name = "chanmin"
age = 27
print("%s is %d years old."%(name, age))
```

실습 10

문자열이 아닌 객체도 "%S" 형식 지정자를 사용해 문자열로 표시합니다.

```
mylist = [1, 2, 3]
print("a list: %s"%mylist)
```

format_string에는 무엇을 대입해야 할까요?

```
data = ("chanmin", 172, 76.4)
format_string = None
print(format_string % data)
```

기본적인 문자열 연산

- 문자열은 문자의 집합이므로 리스트처럼 인덱스로 접근할 수 있습니다.
- 단, 읽기 전용 객체이므로 변경은 불가능합니다.
- 미리 정의한 문자열 객체의 메서드를 이용하면 간단하게 원하는 문자열 연산을 수행할 수 있습니다.
- 인덱스 연산자는 조각 나누기로 부분 집합에 대한 연산을 지원합니다.
- 조각 나누기는 [시작 : 종료 : 스텝]으로 표현됩니다. 이때 종료 위치는 포함되지 않으며 스텝은 생략하면 1입니다.

실습 11

len() 함수는 리스트나 문자열에 포함된 요소의 개수를 반환합니다.

```
wellcom = "Hello World!"
print(len(wellcom))
```

index() 메소드는 문자의 위치를 반환합니다.

```
wellcom = "Hello World!"
print(wellcom.index("W"))
```

실습 12

count() 메소드는 동일 문자가 몇 개 인지를 반환합니다.

```
wellcom = "Hello World!"
Print(wellcom.count("l"))
```

조각 나누기를 이용해 부분만 접근할 수 있습니다.

```
wellcom = "Hello World!"
print(wellcom[6:9])
```

실습 13

스탭은 일정 크기(다음)를 건너 뛰며 부분집합 모으기를 합니다.

```
wellcom = "Hello World!"
print(wellcom[3:7:3])
```

시작과 종료는 생략할 수 있습니다. 스탭이 음수면 역순으로 모음니다.

```
wellcom = "Hello World!"
print(wellcom[::-1])
```

실습 14

upper()와 lower()는 대/소문자로 변환합니다.

```
wellcom = "Hello World!"
print(wellcom.upper())
print(wellcom.lower())
```

startswith()와 endswith()는 해당 문자열로 시작/종료하는지 검사합니다.

```
wellcom = "Hello World!"
print(wellcom.startswith("Hell")
print(wellcom.endswith("World")
```

실습 15

upper()와 lower()는 대/소문자로 변환된 새로운 문자열을 만듭니다.

```
wellcom = "Hello World!"
print(wellcom.upper())
print(wellcom.lower())
```

startswith()와 endswith()는 무언가로 시작하거나 끝나는지 검사합니다.

```
wellcom = "Hello World!"
print(wellcom.startswith("Hell")
print(wellcom.endswith("World")
```

실습 16

split()는 무언가로 문자열을 나눠 리스트로 반환합니다.

```
wellcom = "Hello World!"
print(wellcom.split(" "))
```

인자를 생략하면 화이트스페이스 중 하나로 문자열을 나눕니다.

```
wellcom = "Hello   World! ₩n"

print(wellcom.split(" "))
print(wellcom.split())
```

조건

- 파이썬은 부울 변수를 사용해 조건을 평가합니다.
- 표현식을 비교하거나 평가할 때 부울 값 True 또는 False를 반환합니다.
- 조건 제어를 사용하면 표현식의 결과에 따라 실행할 문장을 선택할 수 있습니다.
- 정수 0은 False로 취급하고 나머지는 True로 취급합니다.
- 빈 객체는 False로 취급합니다.

실습 17

>, <, >=, <=, ==, != 비교 연산자는 표현식을 만듭니다.

```
x = 2
print(x == 2)
print(x == 3)
print(x < 3)
print(x != 3)
print(x <= 3)
```

and, or 부울 연산자를 사용해 복잡한 부울 식을 만들 수 있습니다.

```
name = "chanmin"
age = 27

if name == "chanmin" and age == 27:
    print("perfect!!!")
if name == "chanmin" or name == "mijung":
    print("good!!!")
```

실습 18

in은 대상이 리스트와 같이 반복 가능한 컨테이너 객체 안에 있는지 확인합니다.

```
name = "chanmin"
if name in ["chanmin", "mijung"]:
    print("good")
```

파이썬의 코드 블록은 공백 4개 들여 쓰기이며 종료가 없습니다.

```
score = 90
if score > 90:
    result = "A, best of best!!!"
elif score >= 80:
    result = "B"
    result += ", pass."
else:
    result = "C"
print(result)
```

실습 19

is는 값을 비교하는 ==와 달리 참조 대상이 같은지 확인합니다.

```
x = [1, 2, 3]
y = [1, 2, 3]

print(x == y)
print(x is y)
```

부울 표현식 앞에 not을 사용하면 결과는 반전됩니다.

```
print(not False)
print(not 0 == False)
print(not (10 > 20))
```

실습 20

결과는 어떻게 될까요?

```
number = 10
other = 10
first_array = []
second_array = [1, 2, 3]
result = []
```

```
if number > 15:
    result.append(1)
if first_array:
    result.append(2)
if len(second_array) == 3:
    result.append(3)
if second_array and second_array[0] == 1:
    result.append(4)
print(result)
```

루프

- 파이썬은 for, while 루프만 지원합니다.
- for는 항상 반복가능한 객체의 크기만큼 블록을 반복합니다.
- while은 표현식의 결과가 True인 동안 블록을 반복합니다.
- for, while 문이 항상 True 면 블록은 무한히 반복됩니다.
- break는 루프 블록을 강제로 벗어납니다.
- continue는 실행을 for, while 문으로 옮깁니다.

실습 21

for 루프는 리스트와 같이 반복가능한 시퀀스 객체를 반복합니다.

```
primes = [2, 3, 5, 7]
for prime in primes:
    print(prime)
```

for 루프는 range() 함수를 이용해 숫자 시퀀스를 반복할 수 있습니다.

```
result = []
for x in range(5):
    result.append(x)
for x in range(3, 8)
    result.append(x):
for x in range(3, 8, 2):
    result.append(x)
print(x)
```

실습 22

while 루프는 특정 부울 조건이 충족되는 한 반복됩니다.

```
count = 0
while count < 5:
    print(count)
    count += 1
```

break는 강제로 for, while 루프를 종료합니다.

```
count = 0
while True:
    if count >= 5:
        break;
    print(count)
    count += 1
print("last count is %d" % count)
```

실습 23

continue는 for, while 문으로 돌아갑니다.

```
for x in range(10):
    if x % 2 == 0:
        continue
    print(x)
```

else는 for, while 문이 False 면 실행됩니다.

```
count = 0
while count < 5:
    print(count)
    count += 1
else:
    print("count = %d" % count)
```

실습 24

break는 조건문을 False로 만들지 않으므로 else 문이 실행되지 않습니다.

```
for i in range(1, 10)
    if i % 5 == 0:
        break;
    print(i)
else:
    print("not printed")
```

예외가 발생하는 이유는 무엇 때문일까요?

```
nums = [x for x in range(1, 10 + 1)]
print(nums)
for pos in range(len(nums)):
    if nums[pos] % 2 == 0:
        del nums[pos];
print(nums)
```

함수

- 함수는 코드를 유용한 블록으로 나눌 수 있는 편리한 방법입니다.
- 함수를 코드를 정렬하고, 읽기 쉽게 만들며 재사용을 통해 시간을 절약합니다.
- 함수는 프로그래머가 코드를 공유할 수 있도록 인터페이스를 정의하는 핵심적인 방법입니다.

실습 25

def를 사용해 이름이 있는 함수 블록을 만듭니다.

```
def my_func():
    print("hello from my function!")
```

함수는 사용자가 전달한 인자를 수신할 수 있습니다.

```
def my_func_arg(name, age):
    print("Hi, %s(%d)"%(name, age))
```

실습 26

return 문으로 결과를 호출한 쪽으로 반환할 수 있습니다.

```
def sum_two(a, b):
    return a + b
```

정의한 함수 호출은 이름 뒤에 ()와 인자를 넣으면 됩니다.

```
# 앞에서 정의한 함수 호출

my_func()
my_func_arg("chamin", 27)
x = sum_two(1, 2)
print(x)
```

실습 27

전역 변수를 변경할 때는 global로 만들어야 합니다.

```
x = 1 # 전역 변수
def foo():
    global x
    x += 1

x += 1
foo()
print(x)
```

지역 변수는 함수 내부에서만 사용합니다.

```
my_func()
my_func_arg("chamin", 27)
x = sum_two(1, 2)
print(x)
```

클래스와 객체

- 객체는 변수와 함수를 단일 엔티티로 캡슐화 한 것입니다.
- 객체는 클래스에서 변수와 함수를 가져옵니다.
- 클래스는 기본적으로 객체를 생성하기 위한 템플릿입니다.

실습 28

class 키워드를 이용해 변수와 함수가 포함된 클래스를 정의합니다.

```
class MyClass:
    variable = "nothing"

    def function(self):
        print("inside the class")
```

클래스에 포함된 함수는 첫 번째 인자로 self를 갖습니다.

```
class MyClass:
    def sum_two(self, a, b):
        return a + b

    def show(self, msg):
        print(msg)
```

실습 29

클래스로부터 객체를 만드는 방법은 클래스 이름과 괄호를 이용합니다.

```
class MyClass:
    pass

myobject = MyClass()
```

같은 클래스로 여러 다른 객체를 만들 수 있습니다

```
class MyClass:
    pass

myobject1 = MyClass()
myobject2 = MyClass()
```

실습 30

새로 만들어진 객체에서 변수를 읽을 때는 점(.)을 사용합니다.

```
class MyClass:
    variable = "noting"

myobject = MyClass()
print(myobject.variable)
```

새로 만들어진 객체에서 함수에 접근할 때도 점(.)을 사용합니다

```
class MyClass:
    def sum_two(self, a, b):
        return a + b

myobject = MyClass()
result = myobject.sum_two(1, 2)
print(result)
```

실습 31

self는 같은 클래스에서 서로 다른 객체를 구분합니다.

```
class MyClass:
    def set(self, a):
        self.var = a

    def get(self):
        return self.var
```

객체로 함수에 접근할 때 객체의 이름이 self에 대입됩니다.

```
myobject1 = MyClass()
myobject2 = MyClass()

myobject1.set(1)
print(myobject1.get())
myobject2.set(2)
print(myobject2.get())
```

딕셔너리

- 딕셔너리는 배열과 비슷하지만 키와 값으로 작동합니다.
- 키는 배열의 인덱스처럼 값이 저장된 위치를 나타냅니다.
- 데이터의 양에 상관없이 항상 일정 시간 이내에 데이터에 접근할 수 있습니다.

실습 32

딕셔너리의 다음과 같이 사용합니다.

```
phonebook = {}
phonebook["chanmin"] = "01011112222"
phonebook["mijung"] = "01022223333"
phonebook["gihyun"] = "01033334444"

print(phonebook)
```

초기화 방법은 다음과 같습니다.

```
phonebook =
{"chanmin":"01011112222",
    "mijung":"01022223333",
    "gihyun":"01033334444"
}

print(phonebook)
```

실습 33

for 루프를 사용하면 딕셔너리 전체 요소에 접근할 수 있습니다.

```
phonebook = {"chanmin":"01011112222",
    "mijung":"01022223333",
    "gihyun":"01033334444"
}

for name, number in phonebook.items():
    print("%s is %s"%(name, number))
```

특정 요소를 제거하려면 del 구문을 사용하면 됩니다.

```
phonebook =
{"chanmin":"01011112222",
    "mijung":"01022223333",
    "gihyun":"01033334444"
}

del phonebook["chanmin"]
print(phonebook)
```

모듈과 패키지

- 파이썬의 모듈은 함수 집합 또는 클래스를 구현하는 .py 확장자를 가진 파이썬 파일입니다.
- 한 모듈에서 import 구문을 사용해 다른 모듈을 가져옵니다.
- 모듈이 처음 로드될 때 한 번 모듈의 코드를 실행해 초기화합니다. 따라서 두 번 로드해도 첫 번째 초기화된 결과를 재사용하게 됩니다.
- 패키지는 여러 패키지와 모듈 자체를 포함하는 이름 공간입니다. 하지만 이들의 실체는 디렉터리일 뿐입니다.

실습 34

requests 모듈을 로드하면 쉽게 웹 페이지를 가져올 수 있습니다.

```
import requests

url = "http://www.hani.co.kr"
code = requests.get(url)
print(code.text)
```

모듈에 포함된 함수 이름은 dir()로 얻고, 사용법은 help()를 사용합니다.

```
import requests

print(dir(requests))

help(requests.get)
```

실습 35

새로운 모듈을 만드는 것은 쉽습니다. 그냥 파이썬 파일을 만들면 됩니다.

```
# 파일 이름은 my.py

def function():
    print("it's me")
```

import 문으로 새로 만든 모듈을 가져옵니다.

```
# 파일 이름은 you.py

import my

my.function()
```

실습 36

패키지에 포함된 모듈을 가져오는 방법은 2가지 있습니다.

```
# foo 디렉터리에 my.py 포함

# 첫 번째 방법
import foo.my

# 두 번째 방법
from foo import my
```

패키지에 포함된 모든 모듈을 가져오려면 __init__.py가 제공되어야 합니다.

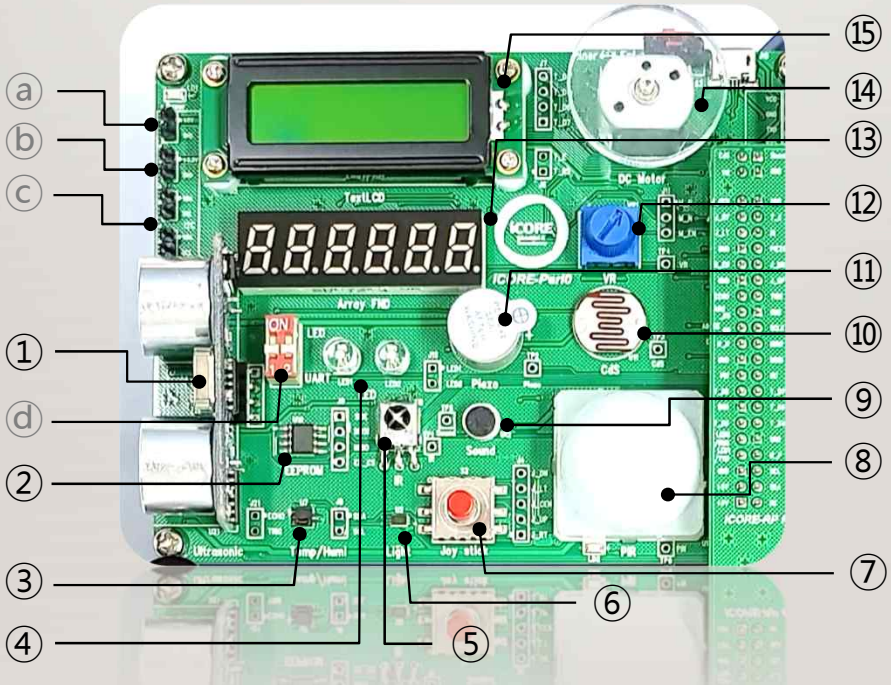
```
# foo 디렉터리에 my.py you.py 포함

from foo import *

# foo 디렉터리는 __init__.py도 포함
__all__ = [“my”, “you”]
```

센서 제어 101

IoT 센서 제어 모듈

- 프로세서는 GPIO, I2C, SPI와 같은 인터페이스를 통해 주변장치를 제어합니다.
- GPIO는 범용 입출력 인터페이스로 입력 또는 출력으로 사용합니다.
- I2C는 칩과 칩을 1:n으로 연결하는 버스 인터페이스로 마스터인 프로세서는 슬레이브 주소를 이용해 통신합니다.
- SPI는 I2C와 유사한 버스 인터페이스로 속도가 빠르며 주소 대신 선택 핀을 사용합니다.

실습 41



실습

센서 제어에 필요한 시간 지연은 time 모듈 적재해 적재합니다.

```
import time
```

sleep() 함수는 초 단위로 다음 명령을 지연시킵니다. (0.1은 100ms)

```
print("Hello World!")
time.sleep(1)            #1초
print("after 1sec")
time.sleep(0.02)     #20ms
print("after 20ms")
```

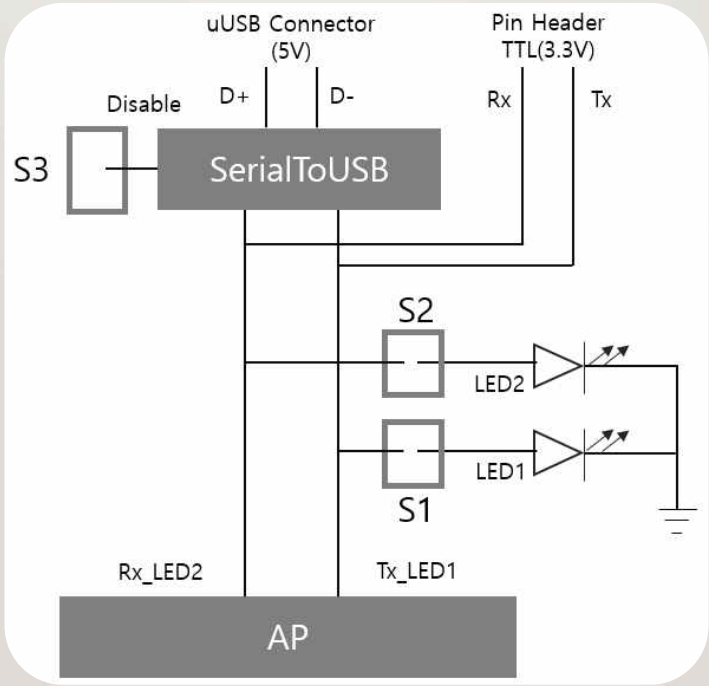
GPIO

- 프로세서 내부에는 외부로부터 입력된 신호를 처리하는 입력 회로와 외부에 신호를 내보내는 출력 회로가 구현되어 있습니다.
- 외부에 노출된 하나의 핀에 입력 회로와 출력 회로를 연결한 후 핀의 기능을 사용자가 선택할 수 있도록 해 하드웨어 설계에 유연성을 부여합니다.
- 파이썬은 RPi.GPIO 라이브러리를 사용해 GPIO를 제어할 수 있습니다.

LED

- LED는 전류의 방향이 전극 방향과 일치하면 빛을 내는 반도체 소자로 대표적인 출력장치 입니다.
- 전압이 낮으면 전류가 거의 흐르지 않아 빛을 내지 않지만 동작 전압 범위에서는 전류량에 비례해 빛이 발생합니다.
- Peri0에는 2개의 LED가 있으며 GPIO14, GPOI15에 연결되어 있습니다.

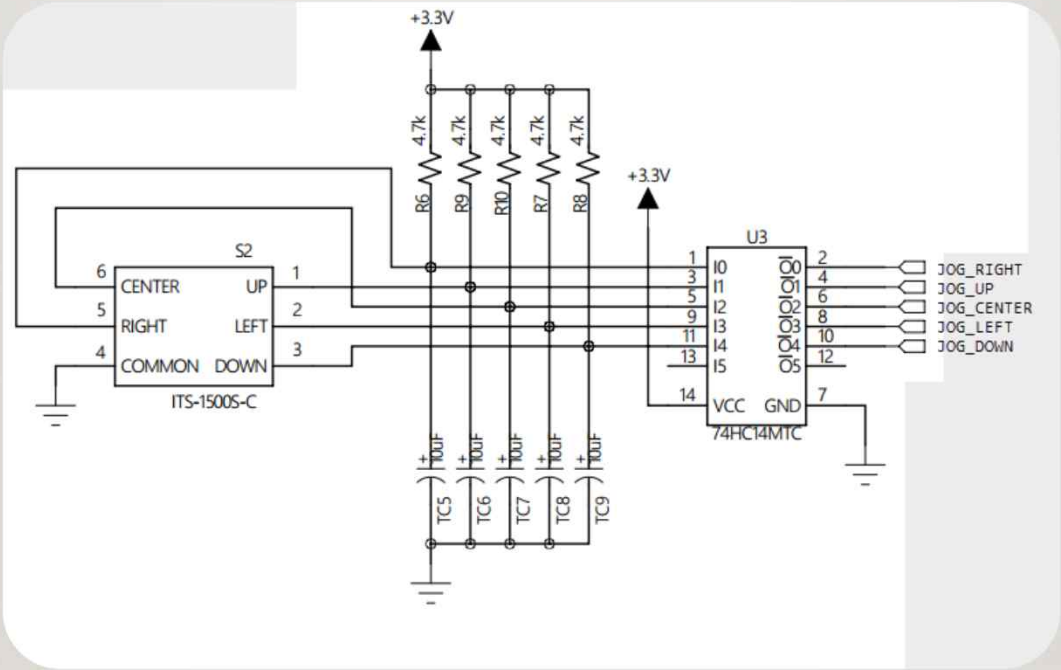
LED


uUSB Connector
(5V)
Pin Header
TTL(3.3V)
Disable
D+
D-
Rx
Tx
S3
SerialToUSB
S2
LED2
LED1
S1
Rx_LED2
Tx_LED1
AP

Joystick

- Joystick은 직관적으로 4개의 방향과 가운데 중 하나를 선택할 때 사용하는데 대표적인 입력장치 입니다.
- Joystick은 5개의 버튼 스위치를 모아 놓은 것과 같습니다.
- Peri0에 있는 Joystick의 left, right, up, down, center는 각각 GPIO16, GPIO20, GPIO5, GPIO6, GPIO21에 연결되어 있습니다.

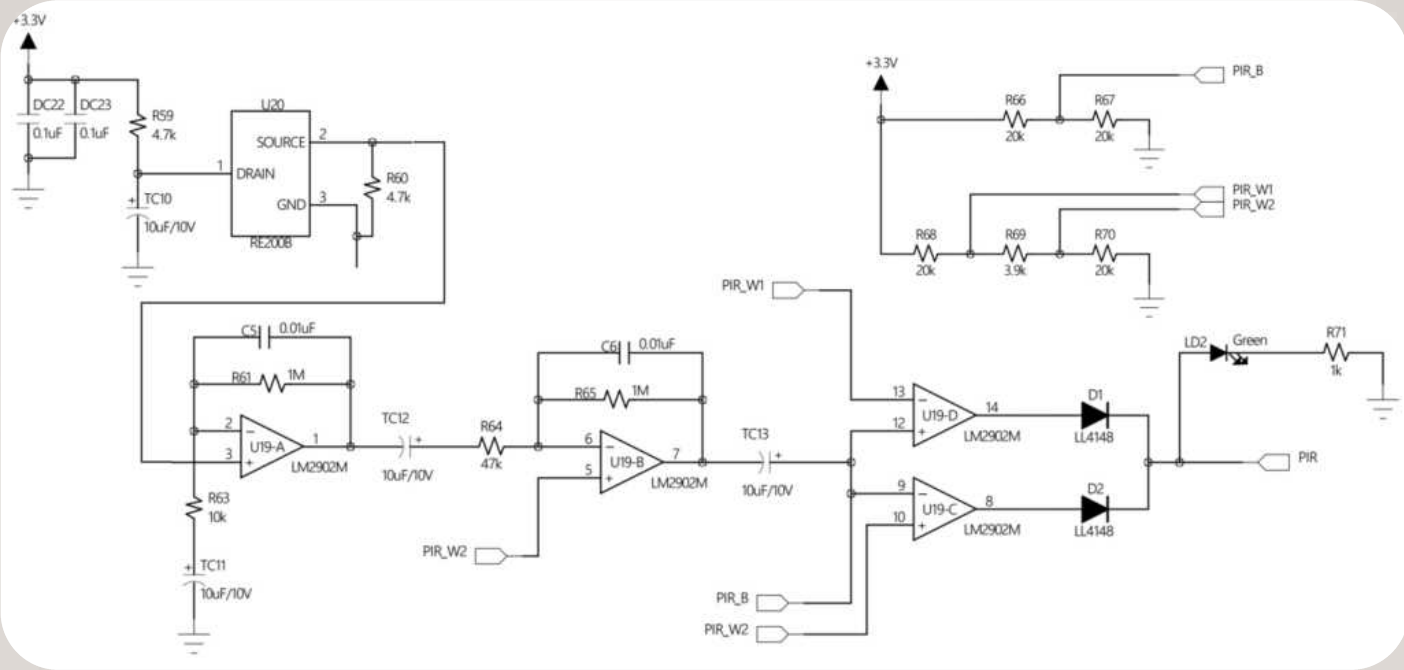
Joystick


+3.3V
4.7k
R6
4.7k
R9
4.7k
R10
4.7k
R7
4.7k
R8
+3.3V
S2
6
CENTER
UP
1
5
RIGHT
LEFT
2
4
COMMON
DOWN
3
ITS-1500S-C
U3
1
I0
O0
2
3
I1
O1
4
5
I2
O2
6
9
I3
O3
8
11
I4
O4
10
13
I5
O5
12
14
VCC
GND
7
74HC14MTC
JOG_RIGHT
JOG_UP
JOG_CENTER
JOG_LEFT
JOG_DOWN
100uF
TC5
100uF
TC6
100uF
TC7
100uF
TC8
100uF
TC9

PIR Sensor

- PIR Sensor는 프리넬 렌즈와 적외선 검출기, 판별 회로부로 구성됩니다.
- 프리넬 렌즈가 적외선만 모아 적외선 검출기로 전달하면 적외선 검출기는 수평면을 기준으로 첫 번째 감지점과 두 번째 감지점을 구분해 판별 회로로 전달합니다.
- 판별 회로는 두 입력 값의 변화를 비교한 후 디지털 결과를 출력합니다.
- Peri0의 PIR은 GPIO24에 연결되어 있으며 움직임이 없으면 LOW, 감지되면 HIGH입니다.

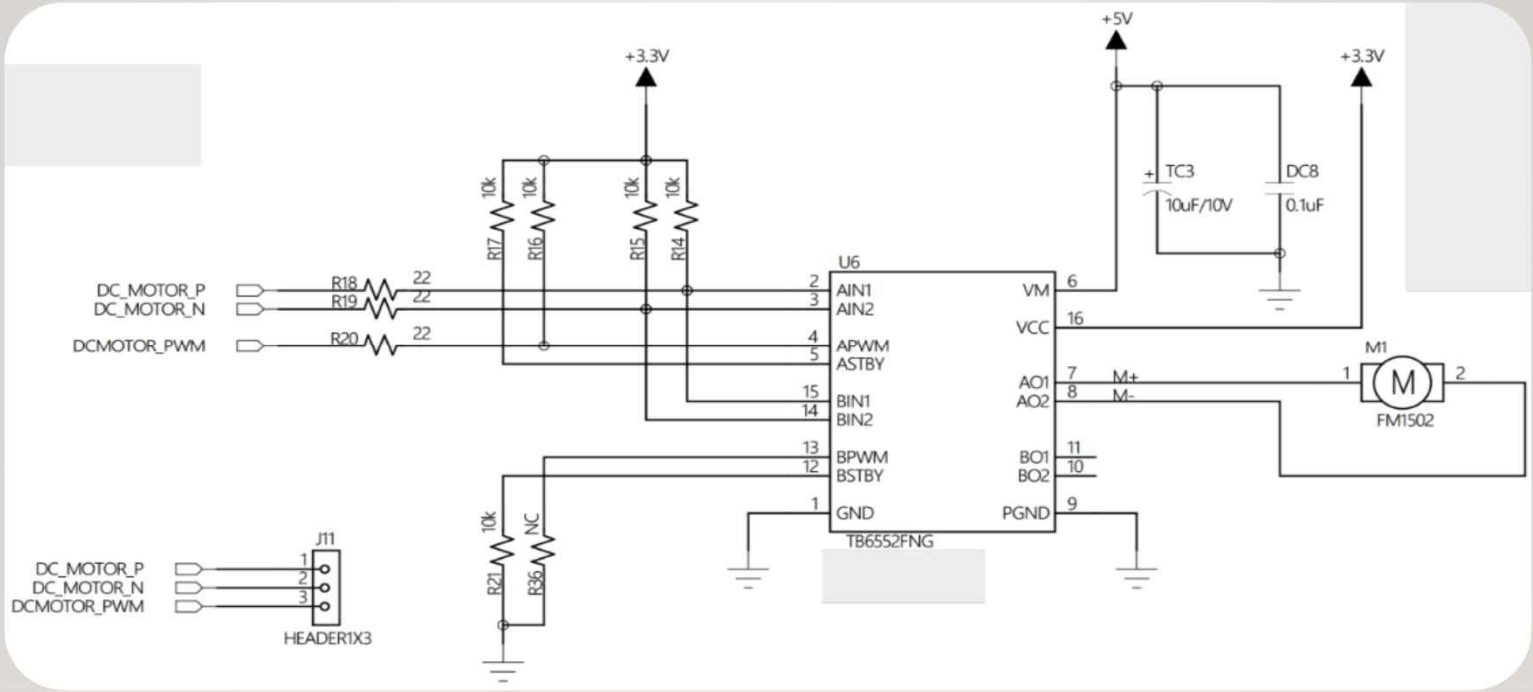
PIR Sensor


+3.3V
DC22
0.1uF
DC23
0.1uF
R59
4.7k
U20
SOURCE
DRAIN
GND
RE200B
R60
4.7k
TC10
10uF/10V
C5
0.01uF
R61
1M
U19-A
LM2902M
TC12
10uF/10V
R64
47k
C6
0.01uF
R65
1M
U19-B
LM2902M
TC13
10uF/10V
R63
10k
TC11
10uF/10V
PIR_W2
PIR_W1
PIR_B
PIR_W2
+3.3V
R66
20k
R67
20k
PIR_B
PIR_W1
PIR_W2
R68
20k
R69
3.9k
R70
20k
U19-D
LM2902M
U19-C
LM2902M
D1
LL4148
D2
LL4148
LD2
Green
R71
1k
PIR

DC Motor

- DC 모터는 N극과 S극 영구 자석을 양쪽에 고정하고 가운데 회전 가능한 코일과 정류자를 배치해 전류에 의한 반발력과 흡입력으로 회전력을 만들어냅니다.
- 모터 드라이버는 PW 입력을 DC 모터의 공급 전압으로 전달하고 P, N 입력에 따라 회전 방향을 바꿉니다.
- Peri0에는 1개의 DC Motor가 있으며 모터 드라이버의 PW는 GPIO12, P, N은 각각 GPIO4, GPIO25에 연결되어 있습니다.

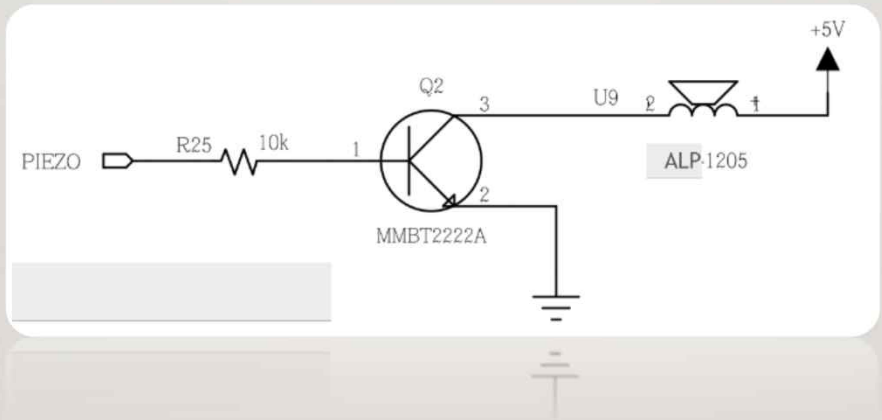
DC Motor


+3.3V
+5V
+3.3V
10k
10k
10k
10k
R17
R16
R15
R14
TC3
10uF/10V
DC8
0.1uF
U6
DC_MOTOR_P
DC_MOTOR_N
DCMOTOR_PWM
R18
R19
R20
22
22
22
AIN1
AIN2
APWM
ASTBY
BIN1
BIN2
BPWM
BSTBY
GND
VM
VCC
AO1
AO2
BO1
BO2
PGND
M+
M-
M1
FM1502
TB6552FNG
J11
HEADER1X3
R21
R36
NC

Piezo Buzzer

- Piezo Buzzer는 피에조 효과를 이용해 소리를 내는 소자로 구동 회로가 포함된 Active 타입과 구동 회로가 빠진 Passive 타입으로 나눠집니다.
- 구동 회로가 없는 Passive 타입은 GPIO를 이용해 원하는 주파수를 PWM으로 전달하면 주파수에 대응하는 소리가 출력됩니다.
- Peri0의 Passive 타입의 Piezo Buzzer는 GPIO13에 연결되어 있습니다.

Piezo



실습 00

GPIO에 연결된 장치를 제어하려면 RPi.GPIO 모듈을 적재합니다.

```
import RPi.GPIO as GPIO

GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
```

setup()에 핀 번호와 함께 GPIO.IN 또는 GPIO.OUT을 인자로 방향을 지정합니다.

```
led1 = 14
led2 = 15

GPIO.setup(led1, GPIO.OUT)
GPIO.setup(led2, GPIO.OUT)
```

실습

output() 함수는 출력 상태를 변경합니다.

```
GPIO.output(led1, GPIO.HIGH)
time.sleep(3)
GPIO.output(led1, GPIO.LOW)
```

듀플로 묶어서 한 번에 처리할 수도 있습니다.

```
led = (14, 15) #led1, led2
GPIO.setup(led, GPIO.OUT)

GPIO.output(led, GPIO.HIGH)
time.sleep(3)
GPIO.output(led, GPIO.LOW)
```

실습

input() 함수는 현재 입력 상태를 반환합니다.

```
joy_center = 21
GPIO.setup(joy_center, GPIO.IN)

print("press Joystick::Center")
while True:
    stat = GPIO.input(joy_center)
    if stat == GPIO.HIGH:
        break;
    time.sleep(0.1)
```

input() 함수는 현재 입력 상태를 반환합니다.

```
joy = (16, 20, 5, 6)
name = ["left", "right", "up", "down"]
GPIO.setup(joy, GPIO.IN)

while True:
    for i in range(4):
        if GPIO.input(joy[i]):
            print("select = %s"%name[i])
    time.sleep(0.02)
```

실습

add_event_detect()는 상승 또는 하강 에지와 같은 입력 레벨의 변화를 감지해 사용자 함수를 호출합니다.

```
pir = 24
GPIO.setup(pir, GPIO.IN)

def on_pir(gpio):
    print("pir detection!")

GPIO.add_event_detect(pir, GPIO.RISING, callback=on_pir)

while True:
    time.sleep(0.01)
```

실습

PWM 객체를 만들어 일정 주기로 출력 전압을 조절하는 것도 가능합니다.

```
motor = (4, 25, 12) # p, n, pw
GPIO.setup(motor, GPIO.OUT)

#50Hz → 1/50 == 0.02 == 20ms
p = GPIO.PWM(motor[2], 50)
p.start(0)
```

ChangeFrequency()는 주기 비율을 변경합니다.

```
GPIO.output(motor[0], GPIO.HIGH)
GPIO.output(motor[1], GPIO.LOW)

p.ChangeDutyCycle(30)      #30% HIGH
time.sleep(5)
p.ChangeDutyCycle(100)     #100% HIGH
time.sleep(3)
p.stop()
```

실습

ChangeFrequency()는 주파수를 변경합니다.

```
buzzer = 13
GPIO.setup(buzzer, GPIO.OUT)

p = GPIO.PWM(buzzer, 1.0)
p.start(50)  #50%
```

```
hz = (261.6256, 293.6648, 329.6276, 349.2282,
      391.9954, 440.0000, 493.8833, 523.2511)

for n in hz:
    p.ChangeFrequency(n)
    time.sleep(1)

p.stop()
```

I2C

- 터치스크린을 비롯해 많은 센서들이 I2C로 제어합니다.
- 마스터, 슬레이브 구조로 프로세서인 마스터가 슬레이브인 센서를 제어합니다.
- I2C 버스에 연결된 센서들은 고유 주소를 가지고 있습니다.
- 파이썬은 smbus2 라이브러리를 사용해 I2C에 연결된 주변장치를 제어할 수 있습니다.

실습

I2C에 연결된 장치를 제어하려면 smbus2 모듈을 적재합니다.

```
import smbus2 as smbus
```

SMBus() 함수에 인자로 채널 번호를 전달하면 I2C 객체가 반환됩니다.

```
# 채널 번호는 하드웨어 의존적
# peri0는 1번 사용

i2c = smbus.SMBus(1)
```

실습

read_byte()와 write_byte()는 주소를 인자로 한 바이트를 읽거나 씁니다.

.

```
th_addr = 0x40
temp = 0xf3
humi = 0xf5
reset = 0xfe

i2c.write_byte(th_addr, reset)
time.sleep(0.02)

i2c.write_byte(th_addr, temp)
time.sleep(0.1)

data = i2c.read_byte(th_addr) << 8  #상위 바이트
Data |= i2c.read_byte(th_addr)      #하위 바이트

# 계산 공식은 온습도 센서 의존적
print("Temp = %.2f"%(-46.85+175.72/65536 * data))
```

실습

read_i2c_block_data()와 write_i2c_block_data()는 주소를 인자로 연속적인 바이트를 읽거나 씁니다. .

```
light_addr = 0x23
mode = 0x13

for _ in range(50):
    data = i2c.read_i2c_block_data(light_addr, mode, 2)
    time.sleep(0.02)
    print("light = %d lx"%round((data[0] << 8 | data[1]) / 1.2))
    time.sleep(0.1)
```

SPI

- SPI는 I2C에 비해 고속 통신이 가능해 저해상도 LCD나 EEPROM 등이 SPI로 연결됩니다.
- 칩 선택 핀에 의해 주변 장치에 선택됩니다.
- 파이썬은 spidev 라이브러리를 사용해 SPI에 연결된 주변장치를 제어할 수 있습니다.

실습

SPI에 연결된 장치를 제어하려면
spidev 모듈을 적재합니다.

```
import spidev
```

SpiDev() 함수로 SPI 객체를 만든 후
open()으로 버스와 장치를 선택합니다.

```
# 채널 번호와 장치 선택은 하드웨어 의존적
# peri0는 채널 0번 사용. ADC 장치는 1번

adc_dev = 1

spi = spidev.SpiDev()
spi.open(0, adc_dev)
```

실습

xfer2() 함수로 데이터를 전달하고 받습니다.

```
cds_cmd = [0x06, 0x00, 0x00]  #ADC에 연결된 CDS 센서로부터 밝기 단계를 읽음

for _ in range(100):
    tmp = spi.xfer2(cds_cmd)
    data = ((tmp[1] & 0x0F) << 8) | tmp[2]  #data[1]의 하위 4비트가 상위 바이트
    print("cds = %d"%data)
    time.sleep(0.05)
```

실습

(계속) Potentiometer의 노브 레벨 읽기

출력 범위 0 ~ 4095를 0 ~ 40로 변경해 보세요.

```
pm_cmd = [0x06, 0x40, 0x00]  #ADC에 연결된 가변저항에서 노부 상태를 레벨로 읽음

for _ in range(100):
    tmp = spi.xfer2(pm_cmd)
    data = ((tmp[1] & 0x0F) << 8) | tmp[2]  #data[1]의 하위 4비트가 상위 바이트
    print("potentiometer = %d"%data)
    time.sleep(0.05)
```

실습

(계속) Sound Sensor의 소리 레벨 읽기

중심값은 2046 ~ 2048이나 하드웨어에 따라 다를 수 있습니다. 조용한 곳에서 중심값을 먼저 확인하세요.

```
sd_cmd = [0x06, 0x80, 0x00]  #ADC에 연결된 Sound 센서에서 현재 소리 레벨을 읽음

for _ in range(200):
    tmp = spi.xfer2(sd_cmd)
    data = ((tmp[1] & 0x0F) << 8) | tmp[2]  #data[1]의 하위 4비트가 상위 바이트
    print("sound = %d"%data)
    time.sleep(0.05)
```