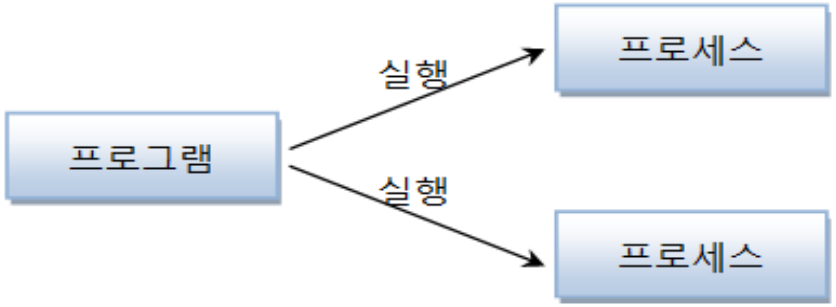
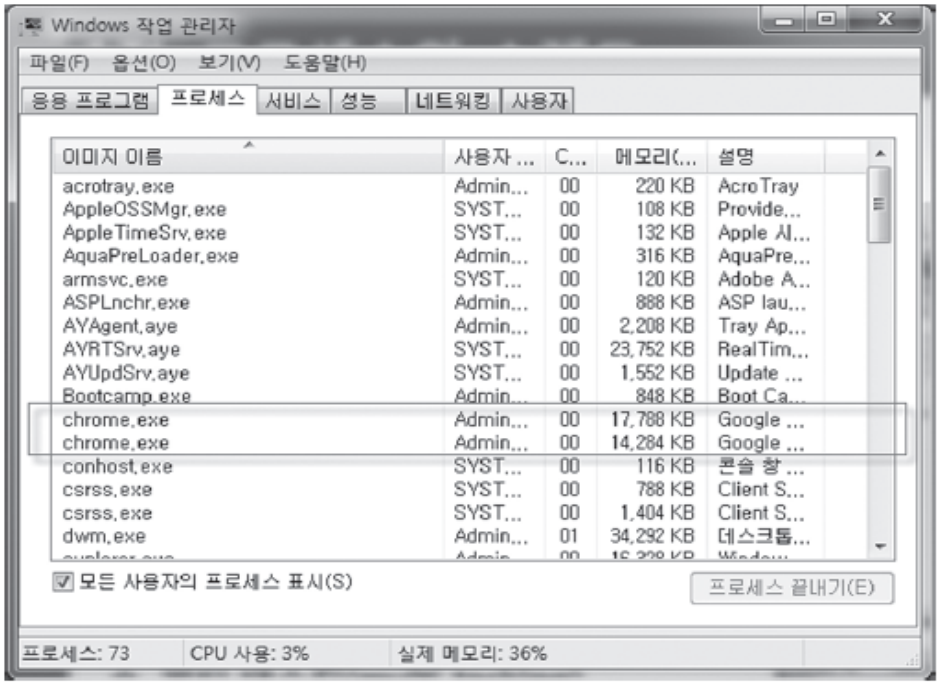
스레드

멀티 스레드

❖ 프로세스(process)

- 실행 중인 하나의 프로그램
- 하나의 프로그램이 여러 프로세스로 만들어짐

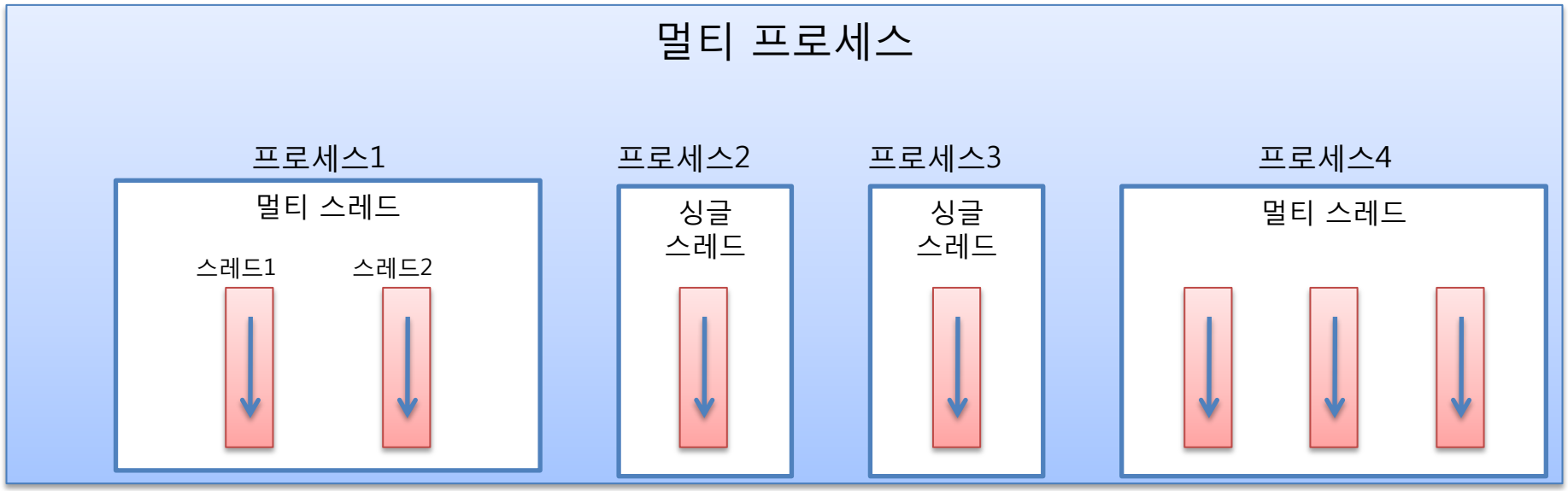




멀티 스레드

❖ 멀티 태스킹(multi tasking)

- 두 가지 이상의 작업을 동시에 처리하는 것
- 멀티 프로세스
  - 독립적으로 프로그램들을 실행하고 여러 가지 작업 처리
- 멀티 스레드
  - 한 개의 프로그램을 실행하고 내부적으로 여러 가지 작업 처리



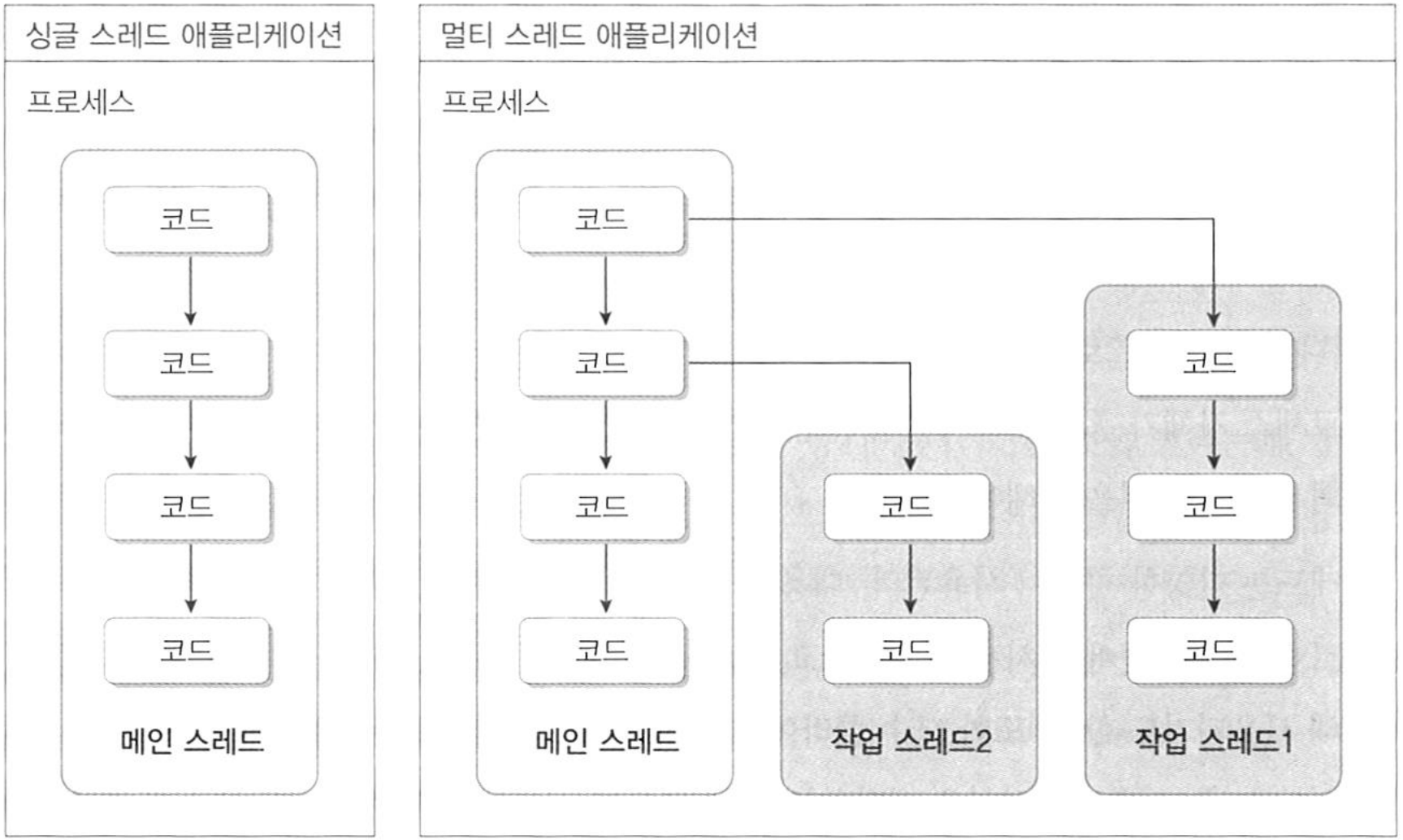
멀티 스레드

❖ 메인(main) 스레드

- 모든 프로그램은 메인 스레드가 실행하며 시작
- 실행 파일의 첫 코드부터 아래로 순차적으로 실행
- 더 이상 실행할 코드가 없는 경우 종료

멀티 스레드

❖ 메인(main) 스레드



멀티 스레드

❖ 메인(main) 스레드

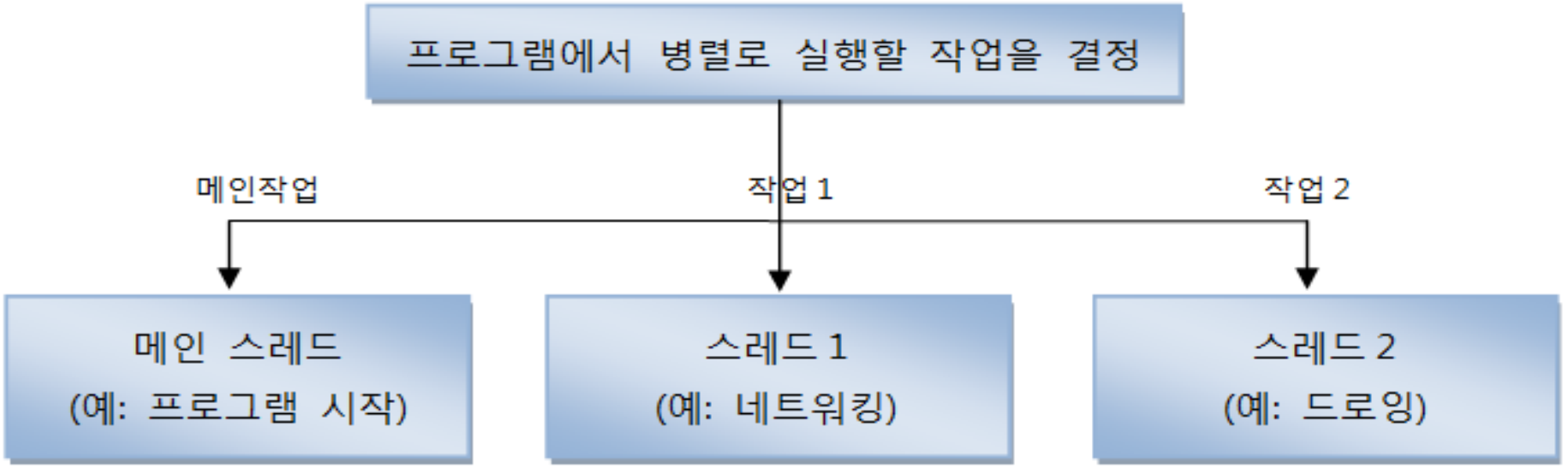
- 실행 종료 조건
  - 마지막 코드 실행
  - return 문을 만나면

- main 스레드는 작업 스레드들을 만들어 병렬로 코드들 실행
  - 멀티 스레드 생성해 멀티 태스킹 수행

- 프로세스의 종료
  - 싱글 스레드: 메인 스레드가 종료하면 프로세스도 종료
  - 멀티 스레드: 실행 중인 스레드가 하나라도 있다면, 프로세스 미종료

❖ 멀티 스레드로 실행하는 어플리케이션 개발

- 몇 개의 작업을 병렬로 실행할지 결정하는 것이 선행되어야



❖ threading 모듈

- threading.Thread() 함수를 호출하여 Thread 객체 생성
  - 생성자에 실행 함수와 인자를 전달
- Thread 객체의 start() 메서드 호출

```
import threading

def sum(low, high):
    total = 0
    for i in range(low, high):
        total += i
    print("Subthread", total)

t = threading.Thread(target=sum, args=(1, 100000))
t.start()

print("Main Thread")
```

멀티 스레드

❖ threading 모듈

```
import threading, requests, time


def getHtml(url):
    resp = requests.get(url)
    time.sleep(1)
    print(url, len(resp.text), resp.text)


t1 = threading.Thread(target=getHtml, args=('https://google.com',))
t1.start()

print("### End ###")
```

❖ threading 모듈

- Thread 클래스 상속 방법
- run() 메서드 구현

```
import threading, requests, time

class HtmlGetter (threading.Thread):
    def __init__(self, url):
        threading.Thread.__init__(self)
        self.url = url

    def run(self):
        resp = requests.get(self.url)
        time.sleep(1)
        print(self.url, len(resp.text), resp.text)

t = HtmlGetter('https://google.com')
t.start()

print("### End ###")
```

가상 센서

가상 센서

❖ 센서 패키지

- sensor 패키지 추가

❖ 온도 센서 값 제너레이터 정의

- sensor.temperature.py

```
import time

def temperature(value, displacement=None):
    if not displacement:
        displacement = (0, 1, 1, 2, -1, -1, -2, 0, -1, -1, -2, 2, 2, 0)
    current = 0;

    while True:
        value += displacement[current]
        current = (current+1)%len(displacement)
        yield value

if __name__ == '__main__':
    for value in temperature(5):
        print(value)
        time.sleep(1)
```

❖ 온도 센서 클래스 정의 : sensor.temperature.py에 정의

```
from threading import Thread
:
class TemperatureSensor(Thread):
    def __init__(self, value=0, displacement=None, interval=1,
                 on_change=None):
        Thread.__init__(self)

        self.sensor = temperature(value, displacement)
        self.value = value
        self.interval = interval
        self.on_change = on_change

    def measure(self):
        return self.value

    def run(self):  # interval 간격으로 센서 값 갱신
        while True:
            time.sleep(self.interval)
            value = self.sensor.__next__()
            if self.on_change:
                self.on_change(value)
```

가상 센서

❖ 온도 센서 클래스 정의 : sensor.temperature.py에 정의

```
if __name__ == '__main__':
    # test 코드
     ts = TemperatureSensor(on_change=lambda  v:print(v) )
     ts.start()
     print('센서 기동')
```

❖ 명령행 인자 처리

- python SensorFrame.py

```
import sys

print(sys.argv)
```

- 실행결과

```
['C:/python-work/iot/SensorFrame.py']
```

❖ 명령행 인자 처리

- python SensorFrame.py 5

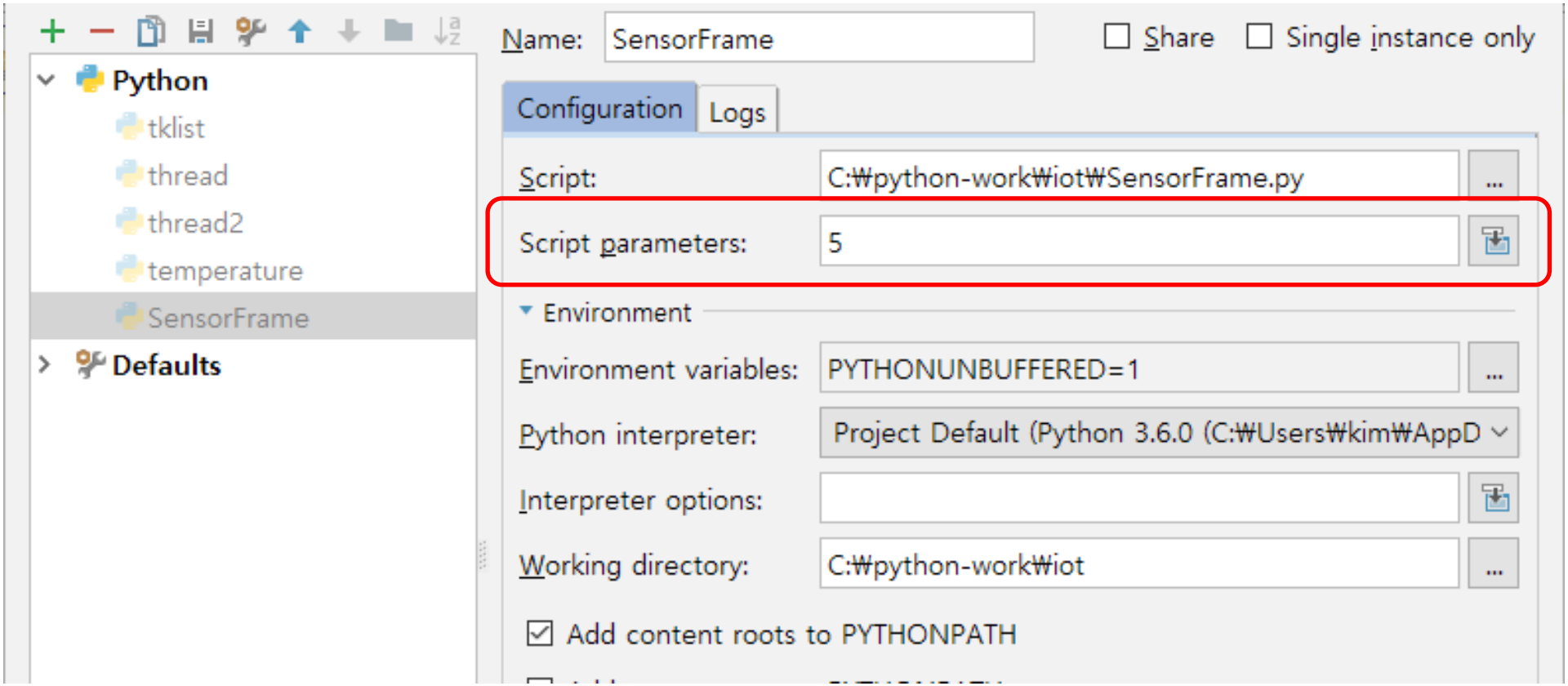
```
import sys

print(sys.argv)
```

- 실행결과

```
['C:/python-work/iot/SensorFrame.py', '5']
```

가상 센서

❖ 실행환경 구성에 명령행 인자 설정



가상 센서

❖ 가상 센서 만들기

❖ 가상 센서 만들기 : SensorFrame.py - 형태 만들기

```
import sys
from tkinter import *
from tkinter.ttk import *
from sensor.temperature import TemperatureSensor

class SensorFrame(Frame):
    def __init__(self, master, category ='', location='', value=0):
        Frame.__init__(self, master)  # master는 부모 윈도우

        self.master = master
        self.master.title('센서 : ' + name)
        self.pack(fill=BOTH, expand=True)  # 부모 윈도우 크기에 맞게 크기 조정

        self.scale = Scale(self, from_=0, to=100, orient=VERTICAL )
        self.scale.pack(ipadx=10, ipady=0, side=LEFT)

        self.lblValue = Label(self)
        self.lblValue.pack(side=LEFT, fill=X, padx=10, expand=True)
```

❖ 가상 센서 만들기 : SensorFrame.py - 형태 만들기

```
def main():
    value = 10       # 디폴트 값
    if len(sys.argv) > 1 :
        value = int(sys.argv[1])

    root = Tk()                             # 메인 윈도우
    root.geometry("200x100+100+100")   # 가로x세로+X위치+Y위치
    SensorFrame(root, 'temperature', 'livingroom', value)
    root.mainloop()

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

❖ 가상 센서 만들기 : SensorFrame.py - 센서 운영하기

```
class SensorFrame(Frame):
    def __init__(self, master, name='', location='', value=0):
        :

        self.sensor = TemperatureSensor(value,
                                on_change=lambda v:self.on_change(v))
        self.sensor.start()
        self.set_value(self.sensor.measure())


    def set_value(self, value):
        self.lblValue.config(text="온도 : " + str(value))
        # print(value, 50-value)
        self.scale.set(50-value)

    def on_change(self, value):
        self.set_value(value)
```

가상 조명

가상 조명

❖ 가상 조명 만들기

가상 조명

❖ 가상 조명 이미지 준비

- light_on.png
- light_off.png

❖ actor 패키지 생성

❖ 가상 조명

- actor/light.py

```
from tkinter import *


class Light(Label):
    def __init__(self, parent,
                 on_file='light_on.png', off_file='light_off.png'):
        self.on_img = PhotoImage(file=on_file)
        self.off_img = PhotoImage(file=off_file)

        super().__init__(parent, image=self.off_img)

        self.place(x=0, y=0)

        self.status = OFF

    def get_status(self):
        return self.status
```

❖ 가상 조명

- actor/light.py

```
    def turn_on(self):
        self.config(image=self.on_img)
        self.status = ON

    def turn_off(self):
        self.config(image=self.off_img)
        self.status = OFF

def main():
    root = Tk()
    root.title('이미지 보기')
    root.geometry('500x400+10+10')
    light = Light(root, '../light_on.png', '../light_off.png')
    light.turn_on()
    root.mainloop()

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

❖ 가상 조명 장치

○ LightFrame.py

```
import sys
from tkinter import *
from tkinter.ttk import *from actor.light import Light

class LightFrame(Frame):
    def __init__(self, master, category='', location='', value=0):
        Frame.__init__(self, master)  # master는 부모 윈도우

        self.master = master
        self.master.title('전등 : ' + category)
        self.pack(fill=BOTH, expand=True)  # 부모 윈도우 크기에 맞게 크기 조정

        # 전등
        self.light = Light(self)
        self.light.pack(side=LEFT, expand=True)

        # 전등 제어 버튼
        self.lightButton = Button(self, text="전등 켜기",
                                  command=lambda: self.on_light_btn_click())
        self.lightButton.pack(side=LEFT, expand=True)
```

❖ 가상 조명 장치

- LightFrame.py

```
    def on_light_btn_click(self):
        self.light.status = not self.light.status
        if self.light.status:
            self.turn_on()
        else:
            self.turn_off()

    def turn_on(self):
        self.light.turn_on()
        self.lightButton.config(text='전등 끄기')

    def turn_off(self):
        self.light.turn_off()
        self.lightButton.config(text='전등 켜기')
```

❖ 가상 조명 장치

- LightFrame.py

```
def main():
    root = Tk()                             # 메인 윈도우
    root.geometry("300x100+100+100")   # 가로x세로+X위치+Y위치
    location = 'livingroom'        # 디폴트 값

    if len(sys.argv) > 1 :
        location = sys.argv[1]

    app = LightFrame(root, location)
    root.mainloop()

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