

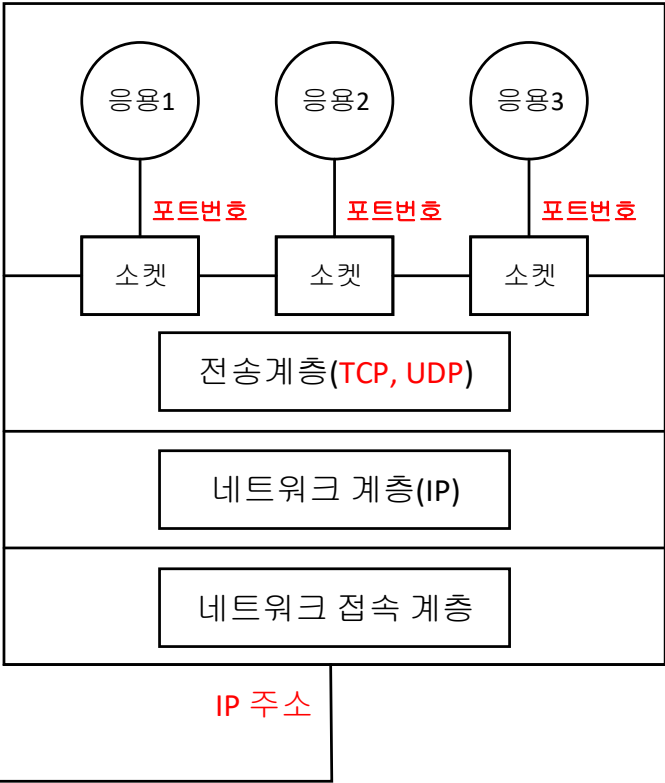
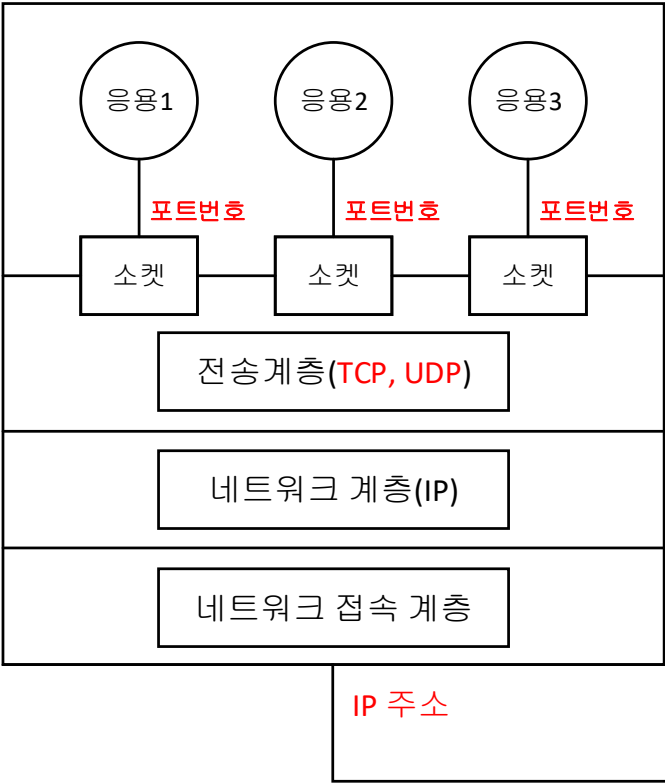
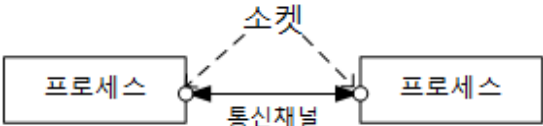
# 네트워크 프로그래밍

## - 소켓 프로그래밍 (TCP) -

순천향대학교 사물인터넷학과

# 응용 프로그램의 식별

## ■ 응용 프로그램과 소켓



네트워크

응용 프로그램이 연결된 소켓을 식별하기 위해서는 IP주소와 함께 포트 번호가 필요함

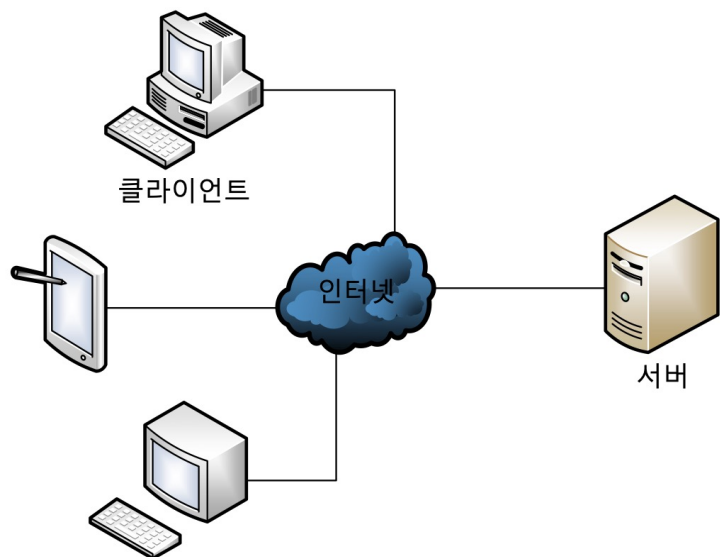
# 네트워크 구조 모델

## ■ 피어-투-피어(peer-to-peer) 구조

- 모든 컴퓨터가 동등하게 요청과 응답이 가능한 구조
- 각 노드가 자원을 분산해서 관리

## ■ 클라이언트-서버 구조

- 서비스 제공자를 제공하는 서버와 서비스를 요청하는 클라이언트로 구성
- 모든 자원이 서버에 집중
- 가장 일반적인 네트워크 구조



# 클라이언트-서버 모델과 파이썬 소켓 모듈

## ■ 클라이언트-서버 모델

- 서버는 클라이언트의 요청을 받아 처리하고 응답을 전송하는 방식
- 하나의 서버와 다수의 클라이언트 통신 가능

## ■ 서버

- 클라이언트의 요청이 접수되면 처리하여 응답을 전송
- 클라이언트와 통신을 위해 **소켓(socket)**을 사용
- 소켓
  - ✓ 서버와 클라이언트의 논리적 전송 통로
  - ✓ 식별: (**호스트 주소 + 서비스 식별 포트번호**)

## ■ 클라이언트

- 소켓을 사용하여 서버와 연결 설정
  - ✓ 서버의 존재와 주소를 알고 있어야 함
- 데이터 송수신
  - ✓ 요청 송신과 응답 수신

# 주소 유형 및 소켓 타입

## ■ TCP/IP 프로토콜 스택

OSI 모델	프로토콜
응용계층	DNS, DHCP, FTP, HTTP, HTTPS, IMAP, NTP, POP3, RTP, RTSP, SSH, SIP, SMTP, SNMP, Telnet, TFTP
표현계층	JPEG, MIDI, MPEG, PICT, TIFF
세션계층	NetBIOS, NFS, PAP, SCP, SQL, ZIP
트랜스포트계층	TCP, UDP
네트워크계층	ICMP, IGMP, IPsec, IPv4, IPv6, IPX, RIP
데이터링크계층	ARP, ATM, CDP, FDDI, Frame Relay, HDLC, MPLS, PPP, STP, Token Ring
물리계층	Bluetooth, Ethernet, DSL, ISDN, Wi-Fi

## ■ 데이터 전송을 위해 소켓의 주소 유형 및 소켓 타입 지정 필요

- 주소 유형address family: 네트워크 계층의 프로토콜 지정
- 소켓 타입socket type: 트랜스포트 계층의 프로토콜 지정

# 주소 유형 및 소켓 타입

## ■ 소켓(객체) 생성 방법

```
import socket
```

```
s = socket.socket(addr_family, type)
```

## ■ 주소 유형

- socket.AF\_INET                      IPv4
- socket.AF\_INET6                    IPv6

## ■ 소켓 타입

- socket.SOCK\_STREAM                TCP (Connection based stream)
- socket.SOCK\_DGRAM                UDP (Datagram)

## ■ 예)

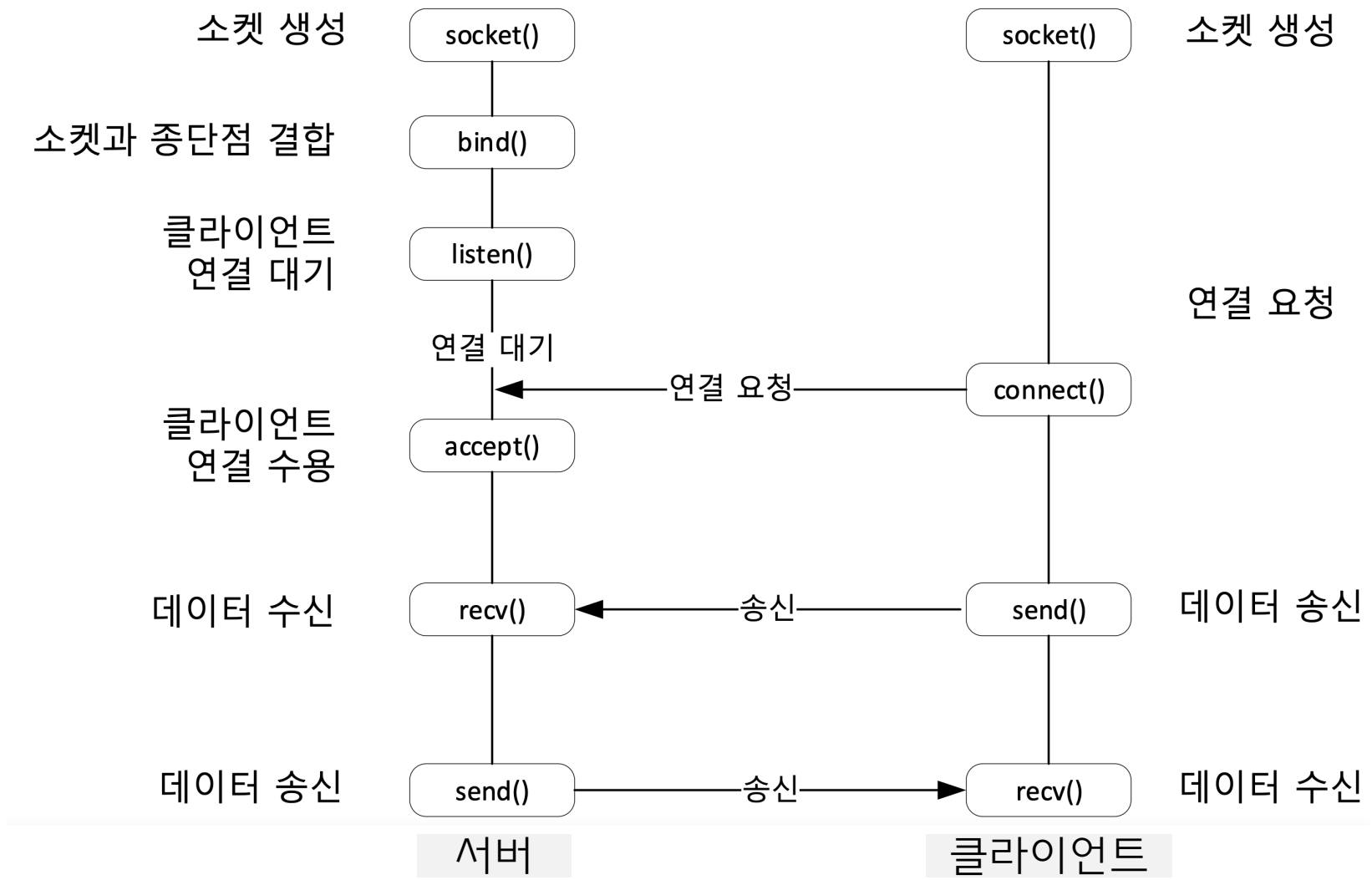
```
from socket import *
```

```
s1 = socket(AF_INET, SOCK_STREAM)    #IPv4, TCP 사용
```

```
s2 = socket(AF_INET, SOCK_DGRAM)    #IPv4, UDP 사용
```

# TCP 소켓 프로그래밍

## ■ TCP 소켓 프로그래밍 순서



# socket 메소드

## ■ 서버 측 메소드

모듈	내용
sock.bind()	종단점(address, port)을 소켓과 결합
sock.listen()	클라이언트 연결 대기
sock.accept()	클라이언트 연결 수용

## ■ 클라이언트 측 메소드

모듈	내용
sock.connect()	서버에 연결 요청

## ■ 데이터 송수신 메소드

모듈	내용
sock.recv()	TCP 소켓을 통해 메시지를 수신. 수신 데이터 반환
sock.recv_into()	TCP 소켓을 통해 메시지를 수신하여 버퍼에 저장
sock.send()	TCP 소켓으로 메시지를 전송. 송신 바이트 수 반환
sock.sendall()	TCP 소켓으로 메시지를 버퍼에 남기지 않고 모두 전송



# TCP 서버

## ■ TCP 서버

- 서버는 정해진 포트 번호로 들어오는 연결(요청)을 기다려야 함
- 서버는 다수의 클라이언트를 동시에 서비스할 수 있어야 함
- 서버는 일반적으로 무한 루프에서 수행됨

## ■ TCP 서버 예제

first\_server.py

```
import socket

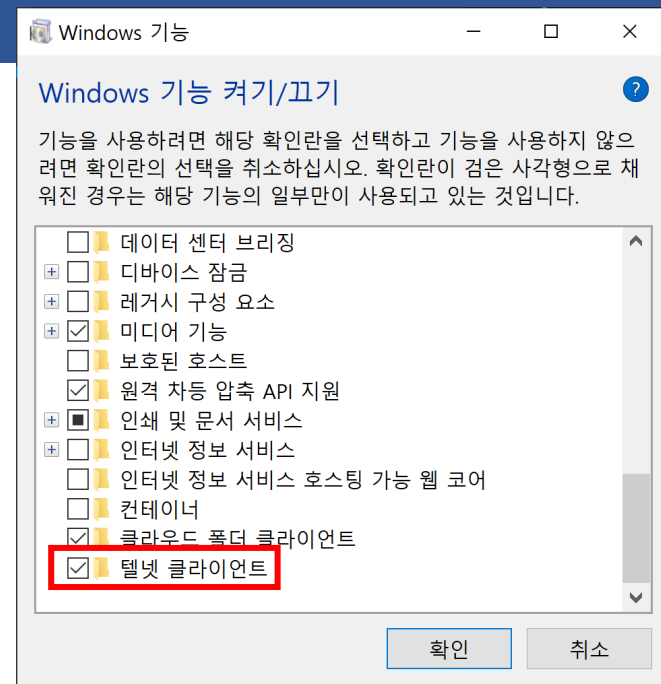
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind(('', 9000))
s.listen(2)

while True:
    client, addr = s.accept()
    print('Connection from ', addr)
    client.send(b'Hello ' + addr[0].encode())
    client.close()
```

# TCP 서버

## ■ 테스트 하기

- 앞에서 작성한 서버 프로그램 실행
- 클라이언트 프로그램으로 telnet 사용
  - ✓ telnet은 “Windows 기능 켜기/끄기”에서 켤 수 있음
- cmd 창을 열고 아래 명령어 수행
  - ✓ `telnet localhost 9000`
- 결과 확인



```
Connection from ('127.0.0.1', 63727)
```

서버

```
C:\Windows\system32\cmd.exe
```

```
Hello 127.0.0.1
```

```
호스트에 대한 연결을 잃었습니다.
```

클라이언트

# TCP 서버

## ■ 주소 바인딩 address binding

- 서버는 소켓을 특정 (주소, 포트)에 바인드시켜야 함

```
s.bind('', 9000)
```

### • bind 메소드

- ✓ 생성된 소켓을 특정 (주소, 포트)에 바인드시키는 메소드
  - 포트 번호는 16비트 정수형으로 1~65535까지 사용 가능
  - 앞쪽 번호는 well-known port로 사용 중이므로, 2000 이후 포트 번호 사용

### ✓ 일반적으로 주소는 입력하지 않음

- 이 경우에, 자동적으로 PC내의 모든 IP 주소가 바인딩됨

```
s.bind('', 9000)
```

- ✓ IP 주소가 여러 개인 경우, 특정 IP 주소를 지정할 수도 있음

```
s.bind('localhost', 9000)
```

```
s.bind('114.71.220.95', 9000)
```

```
s.bind('192.168.56.1', 9000)
```

# TCP 서버

## ■ 연결 대기

- 바인딩 후, 서버는 클라이언트의 연결을 기다림
- `s.listen(backlog)`
  - ✓ backlog: 동시에 연결가능한 소켓 수
  - ✓ 예) `s.listen(2)`
    - 서버는 클라이언트의 연결을 기다리고, 동시에 2개의 연결만 가능함

## ■ 서버 루프 시작 및 새로운 연결 허용

- 서버는 클라이언트의 요청을 처리하기 위해 루프(보통 **무한루프**)를 시작
- 새로운 연결을 허용하기 위해 `accept()` 메소드 호출
  - `while True:`
    - `client, addr = s.accept()`
- `s.accept()`는 연결 요청이 올 때까지 **블로킹** 됨
- 연결 요청이 오지 않으면, 서버는 아무 것도 하지 않음 (`sleep` 상태)

# TCP 서버

## ■ 클라이언트 소켓 & 주소

- `accept()`는 (클라이언트 소켓, 주소) 쌍을 반환
- 새로운 클라이언트 소켓은 이후에 클라이언트와 통신하기 위해 사용됨
  - ✓ "클라이언트 소켓"은 "접속한 클라이언트의 소켓"이 아니라, "접속한 클라이언트와 통신하기 위해 서버가 사용하는 소켓"을 의미함
  - ✓ 서버에서 최초에 생성한 (클라이언트의 연결을 기다리는) 소켓과는 다른 소켓임!!!
- 주소는 클라이언트의 IP 주소와 포트 번호를 포함

연결된 클라이언트와 실제  
통신을 수행하는 소켓

클라이언트의 연결을  
기다리는 소켓

`client, addr = s.accept()`

`<socket._socketobject  
object at 0x3be30>`

`('114.71.220.95', 55999)`

이후에 클라이언트와  
통신하기 위해 사용되는 소켓

연결된 클라이언트의 IP  
주소와 포트 번호

# TCP 서버

## ■ 데이터 송신

- 클라이언트 소켓의 `send()` 메소드를 이용해서 클라이언트로 데이터 송신
- 문자열(유니코드) 전송 시 **bytes 객체(utf-8)**로 변환하여 전송해야 함
- 문자열 → bytes 객체 변환 방법
  - ✓ 문자열 앞에 **b**를 붙이거나, `encode()` 함수 사용
  - ✓ 예) 문자열 'hello'

```
>>> b'hello'
b'hello'
>>> 'hello'.encode()
b'hello'
```
- 데이터 송신 예제

```
client.send(b'Hello ' + addr[0].encode())
```

# TCP 서버

## ■ 연결 종료

- 클라이언트 요청 처리를 완료하면, 자원 해제를 위해 `close()` 호출
- `client.close()`
  - ✓ 소켓을 닫아서, 클라이언트와의 연결을 종료
- 서버는 원하는 만큼 클라이언트와의 연결을 유지할 수 있으며, 반복적으로 데이터를 송수신할 수 있음

## ■ 다음 연결 대기

- 클라이언트와의 연결이 종료된 후, 다음 클라이언트의 연결을 기다림

```
while True:
    client, addr = s.accept()
    print('Connection from ', addr)
    client.send(b'Hello ' + addr[0].encode())
    client.close()
```

- 원래의 서버 측 소켓이 재사용됨
- 서버는 일반적으로 무한 루프를 수행하면서 다음 클라이언트의 연결을 기다림

# TCP 클라이언트

## ■ TCP 클라이언트

- 접속할 서버의 IP 주소와 포트 번호를 알고 있어야 함
- `connect()` 메소드를 이용하여 접속 후, 데이터 송수신

## ■ TCP 클라이언트 예제

first\_client.py

```
import socket

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
addr = ('localhost', 9000)
sock.connect(addr)
msg = sock.recv(1024)
print(msg.decode())
sock.close()
```



# TCP 클라이언트

## ■ 서버 접속

- `addr = ('114.71.220.95', 9000)`
- `sock.connect(addr)`
  - ✓ `addr` 주소의 서버로 접속

## ■ 데이터 수신

- `msg = sock.recv(1024)`
  - ✓ 최대 1024바이트의 데이터를 읽어와서 반환 (bytes 객체)
- 수신한 데이터를 문자열로 처리하기 위해서는 `decode()` 메소드 사용
- `print(msg.decode())`

# TCP 클라이언트

## ■ 테스트하기

- 슬라이드 9의 서버 프로그램 실행: `python first_server.py`  
✓ 항상 서버부터 실행해야 함
- 클라이언트 프로그램 실행: `python first_client.py`
- 결과 확인

first\_server.py U X

first\_server.py > ...

```
1 import socket
2
3 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
4 s.bind(('', 9000))
5 s.listen(2)
6
7 while True:
8     client, addr = s.accept()
9     print('Connection from ', addr)
10    client.send(b'Hello ' + addr[0].encode())
11    client.close()
12
```

first\_client.py U X

first\_client.py > ...

```
1 import socket
2
3 sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
4 addr = ('localhost', 9000)
5 sock.connect(addr)
6 msg = sock.recv(1024)
7 print(msg.decode())
8 sock.close()
9
```

문제 출력 디버그 콘솔 터미널

```
(venv) PS C:\Users\dhkim\net_program> python first_server.py
Connection from ('127.0.0.1', 60509)
Connection from ('127.0.0.1', 60510)
Connection from ('127.0.0.1', 60511)
█
```

```
(venv) PS C:\Users\dhkim\net_program> python first_client.py
Hello 127.0.0.1
(venv) PS C:\Users\dhkim\net_program> python first_client.py
Hello 127.0.0.1
(venv) PS C:\Users\dhkim\net_program> python first_client.py
Hello 127.0.0.1
(venv) PS C:\Users\dhkim\net_program> █
```

# send(), recv() 함수

## ■ 소켓.send(bytes)

- 원격 소켓에 연결되어 있는 소켓을 통해 **bytes 데이터**를 전송
  - ✓ 데이터를 송신하기 전에 bytes 형으로 변환 필요
- 전송된 **바이트 수**를 반환
  - ✓ (대부분의 경우 잘 전송되지만) 항상 모든 데이터가 정상적으로 전송되지는 않음
  - ✓ 응용 프로그램은 모든 데이터가 전송되었는지 확인해야 함
  - ✓ 일부 데이터만 전송되었으면, 응용 프로그램은 나머지 데이터의 전달을 시도해야 함

## ■ 소켓.recv(bufsize)

- 최대 **bufsize** 만큼의 데이터를 수신
  - ✓ (대부분의 경우 잘 수신되지만) 항상 모든 데이터가 정상적으로 수신되지는 않음
  - ✓ 원하는 만큼의 데이터를 수신했는지 확인해야 함
- 수신된 데이터를 나타내는 **bytes 데이터**를 반환
  - ✓ bytes 데이터를 수신한 후 원래 데이터 형태로 변환 필요
  - ✓ 0바이트를 반환하면, 다른 쪽이 연결을 닫았거나 닫고 있다는 의미

# send(), recv() 함수

## ■ 문자열 송수신

- 문자열 전송 시, bytes 형으로 변환(`encode()`) 후 `send()` 실행
- 문자열 수신 시, `recv()`를 통해 수신한 데이터(bytes)를 문자열로 변환(`decode()`)

## ■ 정수 송수신 (1<sup>st</sup> Option)

- 문자열로 변경한 후, 송신. 수신한 후 다시 정수로 변경
  - ✓ 장점: 문자열은 바이트 단위로 전송되므로 "엔디언"을 고려할 필요 없음
- 정수를 bytes 객체로 변경 후 `send()` 수행
  - ✓ 정수를 문자열로 변경: `str()`
  - ✓ 문자열을 bytes 객체로 변경: `문자열.encode()`
- `recv()` 후, 수신한 객체를 정수형으로 변경
  - ✓ bytes 객체를 문자열로 변경: `문자열.decode()`
  - ✓ 문자열을 정수로 변경: `int()`

# send(), recv() 함수

## ■ 정수 변환 예제

```
>>> import socket
>>> a = 12345678
>>> b = str(a)
>>> b
'12345678'
>>> c = b.encode()
>>> c
b'12345678'

>>> d = c.decode()
>>> d
'12345678'
>>> e = int(d)
>>> e
12345678
```

# send(), recv() 함수

## ■ 정수 송수신 (2<sup>nd</sup> Option)

- $\text{int} \longleftrightarrow \text{bytes}$  직접 변환
- `int.to_bytes(length, byteorder, signed=False)`
  - ✓ 원하는 길이의 정수를 나타내는 bytes 객체를 돌려줌
  - ✓ length: 정수 길이
  - ✓ byteorder: 'big' or 'little'
  - ✓ signed: 부호 여부
  - ✓ 예
    - `(1).to_bytes(4, 'big')` → `b'\x00\x00\x00\x01'`
    - `(1).to_bytes(2, 'little')` → `b'\x01\x00'`
- `int.from_bytes(bytes, byteorder, signed=False)`
  - ✓ 주어진 bytes 객체로 표현되는 정수를 돌려줌
  - ✓ bytes: 정수를 나타내는 bytes 객체
  - ✓ byteorder: 'big' or 'little'
  - ✓ signed: 부호 여부
  - ✓ 예
    - `int.from_bytes(b'\x00\x00\x00\x01', 'big')` → 1
    - `int.from_bytes(b'\x01\x00', 'little')` → 1

# send(), recv() 함수

## ■ 정수 송수신 (2<sup>nd</sup> Option)

- 정수를 네트워크 바이트 순서의 bytes 객체로 변경 후 전송  
✓ `(정수).to_bytes(4, 'big')`
- `recv()` 후, 수신한 bytes 객체(네트워크 바이트 순서)를 호스트 바이트 순서 정수로 변경  
✓ `int.from_bytes(bytes, 'big')`

```
>>> a = 1
>>> b = a.to_bytes(4, 'big')
>>> b
b'\x00\x00\x00\x01'

>>> c = int.from_bytes(b, 'big')
>>> c
1
```

## 과제 2: 문자열(이름), 정수(학번) 전송 추가하기

### ■ 클라이언트(좌측 코드)

- 본인의 이름(예:'Daehee Kim')을 문자열로 전송하기
- 본인의 학번을 수신한 후 **엔디언 변환**하여 출력

### ■ 서버(우측 코드)

- 학생의 이름을 수신한 후 출력
- 학생의 학번(정수형 변수, **엔디언 변환 필수**, **문자열 변환하지 말 것!**)을 전송하기

```
import socket
```

```
sock = socket.socket(socket.AF_INET,  
                      socket.SOCK_STREAM)
```

```
addr = ('localhost', 9000)
```

```
sock.connect(addr)
```

```
msg = sock.recv(1024)
```

```
print(msg.decode())
```

```
# 본인의 이름을 문자열로 전송
```

```
# 본인의 학번을 수신 후 출력
```

```
sock.close()
```

```
import socket
```

```
s = socket.socket(socket.AF_INET,  
                  socket.SOCK_STREAM)
```

```
s.bind(('', 9000))
```

```
s.listen(2)
```

```
while True:
```

```
    client, addr = s.accept()
```

```
    print('Connection from ', addr)
```

```
    client.send(b'Hello ' + addr[0].encode())
```

```
    # 학생의 이름을 수신한 후 출력
```

```
    # 학생의 학번을 전송
```

```
    client.close()
```

```
Hello 127.0.0.1  
20170031
```

```
Connection from ('127.0.0.1', 57127)  
Daehee Kim
```



## 과제 2: 문자열(이름), 정수(학번) 전송 추가하기

### ■ 과제2

- 서버와 클라이언트를 각각 1개의 소스 코드(.py)로 저장 (2개)
- 서버와 클라이언트의 실행 결과 캡처 (1개 또는 2개)
  - ✓ 서버와 클라이언트 실행 결과를 각각 캡처해도 되고, 서버와 클라이언트 실행 결과를 1개의 파일로 캡처해도 됨

### ■ 소스 코드

- GitHub에 hw2 폴더 생성 후, 소스 코드 업로드
- GitHub 화면 캡처
  - ✓ 폴더 이름과 파일 이름이 보이도록 캡처할 것

### ■ 제출

- 과제 공지 후 1주일, LMS 제출
- 제출물
  - ✓ 실행 결과 캡처 파일 (1개 또는 2개)
  - ✓ GitHub 화면 캡처 파일 (1개)

# TCP 클라이언트: 웹 브라우저

## ■ 간단한 웹 브라우저

```
from socket import *
```

simple\_web\_browser.py

```
sock = socket(AF_INET, SOCK_STREAM)
sock.connect(('google.com', 80))
sock.send(b'GET / HTTP/1.1\r\n\r\n')
data = sock.recv(10000)
print(data.decode())
sock.close()
```

```
HTTP/1.1 200 OK
Date: Sat, 04 Mar 2023 05:32:37 GMT
Expires: -1
Cache-Control: private, max-age=0
Content-Type: text/html; charset=ISO-8859-1
P3P: CP="This is not a P3P policy! See g.co/p3phelp for more info."
Server: gws
X-XSS-Protection: 0
X-Frame-Options: SAMEORIGIN
Set-Cookie: 1P_JAR=2023-03-04-05; expires=Mon, 03-Apr-2023 05:32:37 GMT; path=/; domain=.google.com; Secure
Set-Cookie: AEC=ARSKqsIpogkYt0y2X_Q0wmSoKc9IudsmLGpwev3iyJzul0a-xlAYjKh-zUQ; expires=Thu, 31-Aug-2023 05:32:37
le.com; Secure; HttpOnly; SameSite=lax
Set-Cookie: NID=511=rGPp6ILMyrNLYPoIvJ0z9U77-pb9MfHfdG1BUwixEU0SKzbXJSVgkVIZf1UW32prN1YzQxgXAja1QLVRGGgZDaU1Q93
V4xogmHKRoIBAXt5S4o7IkR1vC982yG5zNXG46ML_Wfb7WCPnNqu5xQ; expires=Sun, 03-Sep-2023 05:32:37 GMT; path=/; domain=
Accept-Ranges: none
Vary: Accept-Encoding
Transfer-Encoding: chunked

5e8e
<!doctype html><html itemscope="" itemtype="http://schema.org/WebPage" lang="ko"><head><meta content="text/html; charset=ISO-8859-1"
iv="Content-Type"><meta content="/images/branding/googleg/1x/googleg_standard_color_128dp.png" itemprop="image"
ript nonce="3u6QDi6NOMKCuj9ZVzaURw">(function(){window.google={kEI:'9dcCZJ6VOPG80PEP-ciPwA4',kEXPI:'0,1359409,0
```

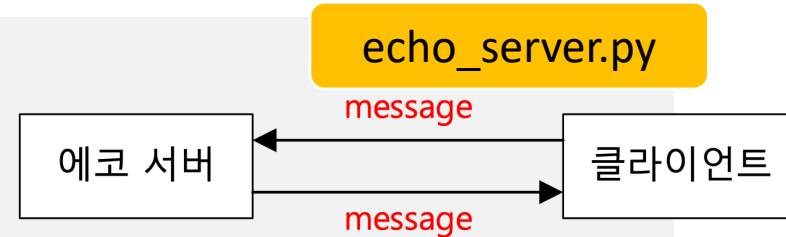
# TCP 에코 서버/클라이언트 프로그램

## ■ TCP 에코 서버

- 클라이언트로부터 수신한 데이터를 출력하고, 상대방에게 다시 전송

```
from socket import *
port = 2500
BUFSIZE = 1024
sock = socket(AF_INET, SOCK_STREAM)
sock.bind(('', port))
sock.listen(1)
conn, (remotehost, remoteport) = sock.accept()
print('connected by', remotehost, remoteport)
while True:
    data = conn.recv(BUFSIZE)
    print("Received message: ", data.decode())
    conn.send(data)

conn.close()
sock.close()
```



# TCP 에코 서버/클라이언트 프로그램

## ■ TCP 에코 클라이언트

- 서버에 연결하여 메시지를 전송하고, 수신 메시지를 출력

```
import socket
```

```
port = int(input("Port No: "))  
address = ("localhost", port)  
BUFSIZE = 1024
```

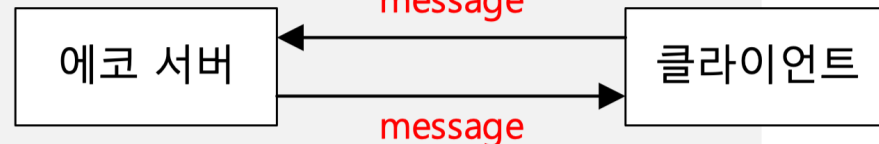
```
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)  
s.connect(address)
```

```
while True:
```

```
    msg = input("Message to send: ")  
    s.send(msg.encode())    #send a message to server  
    data = s.recv(BUFSIZE)  #receive message from server  
    print("Received message: %s" % data.decode())
```

```
s.close()
```

echo\_client.py



```
(venv) PS C:\Users\dhkim\net_program> python echo_server.py  
connected by 127.0.0.1 62719  
Received message: Hello, Server  
Received message: Nice to meet you!  
█
```

```
(venv) PS C:\Users\dhkim\net_program> python echo_client.py  
Port No: 2500  
Message to send: Hello, Server  
Received message: Hello, Server  
Message to send: Nice to meet you!  
Received message: Nice to meet you!  
Message to send: █
```

# 타임 서버/클라이언트 프로그램

## ■ 타임 서버

- 클라이언트가 접속하면 현재 시간을 전송하는 서버 프로그램

```
import socket
import time
```

time\_server.py

```
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind(('', 9999))
s.listen(5)
```

```
while True:
    client, addr = s.accept()
    print('connection from ', addr)
    client.send(time.ctime(time.time()).encode())
    client.close()
```

- `time.time()`
  - ✓ UTC 기준 1970년 1월 1일 0시 0분 0초부터의 경과시간을 나타냄
  - ✓ 예) 1583368987.3660107
- `time.ctime()`
  - ✓ 초 단위의 시간을 문자열로 변환하는 함수
  - ✓ 예) 'Thu Mar 4 09:43:07 2021'

# 타임 서버/클라이언트 프로그램

## ■ 타임 클라이언트

- 타임 서버에 접속하여 시간을 읽어 오는 클라이언트 프로그램

```
import socket
```

time\_client.py

```
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect(('localhost', 9999))
print("Time: ", sock.recv(1024).decode())
sock.close()
```

```
(venv) PS C:\Users\dhkim\net_program> python time_server.py
connection from ('127.0.0.1', 62762)
connection from ('127.0.0.1', 62763)
connection from ('127.0.0.1', 62764)
connection from ('127.0.0.1', 62765)
```

```
(venv) PS C:\Users\dhkim\net_program> python time_client.py
Time: Fri Mar 11 17:02:13 2022
(venv) PS C:\Users\dhkim\net_program> python time_client.py
Time: Fri Mar 11 17:02:14 2022
(venv) PS C:\Users\dhkim\net_program> python time_client.py
Time: Fri Mar 11 17:02:17 2022
(venv) PS C:\Users\dhkim\net_program> python time_client.py
Time: Fri Mar 11 17:02:19 2022
(venv) PS C:\Users\dhkim\net_program>
```

# 예외 처리

- 통신 도중 연결이 끊어지는 경우?
  - 클라이언트가 비정상적(예: **[Ctrl+C]**)으로 종료되는 경우

<pre>Received message: Received message: Received message: Received message: Received message: Received message: Received message: Received message:</pre>	<pre>(venv) PS C:\Users\dhkim\net_program&gt; python echo_client.py Port No: 2500 Message to send: Hello Received message: Hello Message to send: Traceback (most recent call last):   File "C:\Users\dhkim\net_program\echo_client.py", line 11, in &lt; module&gt;     msg = input("Message to send: ") KeyboardInterrupt</pre>
--	---

TCP 에코 서버

TCP 에코 클라이언트

```
while True:
    data = conn.recv(BUFSIZE)
    if not data:
        break
    print("Received message: ", data.decode())
    conn.send(data)

conn.close()
```

<pre>(venv) PS C:\Users\dhkim\net_program&gt; python echo_server.py connected by 127.0.0.1 59746 Received message: Hello (venv) PS C:\Users\dhkim\net_program&gt; █</pre>	<pre>(venv) PS C:\Users\dhkim\net_program&gt; python echo_client.py Port No: 2500 Message to send: Hello Received message: Hello Message to send: Traceback (most recent call last):   File "C:\Users\dhkim\net_program\echo_client.py", line 11, in &lt; module&gt;     msg = input("Message to send: ") KeyboardInterrupt</pre>
---	---

# 예외 처리

## ■ 통신 도중 연결이 끊어지는 경우?

- 서버가 비정상적([Ctrl+Break] 또는 강제종료)으로 종료되는 경우
  - ✓ 클라이언트의 `send()` 함수에서 `ConnectionResetError` 발생 후, 프로그램 종료

```
c:\Wexample>py echo_server.py
connected by 127.0.0.1 49765
^C
c:\Wexample>
```

서버

```
c:\Wexample>py echo_client.py
Port No: 2500
Message to send: Hello
Traceback (most recent call last):
  File "echo_client.py", line 12, in <module>
    s.send(msg.encode()) #send a message to server
ConnectionResetError: [WinError 10054] 현재 연결은 원격 호스트에 의해 강제로 끊겼습니다
```

클라이언트

- 클라이언트가 비정상적([Ctrl+Break] 또는 강제종료)으로 종료되는 경우
  - ✓ 서버의 `recv()` 함수에서 `ConnectionResetError` 발생 후, 프로그램 종료

```
c:\Wexample>py echo_server.py
connected by 127.0.0.1 50160
Traceback (most recent call last):
  File "echo_server.py", line 13, in <module>
    data = conn.recv(BUFSIZE)
ConnectionResetError: [WinError 10054] 현재 연결은 원격 호스트에 의해 강제로 끊겼습니다
```

서버

```
c:\Wexample>py echo_client.py
Port No: 2500
Message to send: ^C
c:\Wexample>
```

클라이언트



# 예외 처리

- 상대방과의 연결이 끊어졌을 때 `send()/recv()`를 호출하면 예외 발생
- 정상적인 프로그램 실행을 위해 예외 처리 필요

```
while True:
    try:
        data = conn.recv(BUFSIZE)
    except:
        break                    #수신할 때 예외 발생
    else:
        if not data:
            break
        print(data.decode())    #정상 처리

    try:
        conn.send(data)
    except:
        break                    #송신할 때 예외 발생

conn.close()
```

# 새로운 에코 서버 프로그램

```
from socket import *
```

```
port = 2500  
BUFSIZE = 1024
```

```
sock = socket(AF_INET, SOCK_STREAM)  
sock.bind(('', port))  
sock.listen(1)  
conn, (remotehost, remoteport) = sock.accept()  
print('connected by', remotehost, remoteport)
```

```
while True:  
    try:  
        data = conn.recv(BUFSIZE)  
    except:  
        break  
    else:  
        if not data:  
            break  
        print("Received message: ", data.decode())  
  
    try:  
        conn.send(data)  
    except:  
        break
```

```
conn.close()
```

echo\_server\_with\_exception.py

# 새로운 에코 클라이언트 프로그램

```
import socket

port = int(input("Port No: "))
address = ("localhost", port)
BUFSIZE = 1024

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect(address)

while True:
    msg = input("Message to send: ")
    try:
        bytesSent = s.send(msg.encode())
    except:
        print('connection closed')
        break
    else:
        print("{} bytes send".format(bytesSent))

    try:
        data = s.recv(BUFSIZE)
    except:
        print('connection closed')
        break
    else:
        if not data:
            break
        print("Received message: %s" % data.decode())

s.close()
```

echo\_client\_with\_exception.py

# 간단한 요청 처리 서버/클라이언트 프로그램

## ■ 서버 동작

- 클라이언트로부터 1~10까지의 숫자를 받으면 영어로 변환하여 전송

```
from socket import *
```

simple\_number\_server.py

```
table = {'1':'one', '2':'two', '3':'three', '4':'four', '5':'five', \
        '6':'six', '7':'seven', '8':'eight', '9':'nine', '10':'ten'}
```

```
s = socket(AF_INET, SOCK_STREAM)
s.bind(('', 3333))
s.listen(5)
print('waiting...')
```

```
while True:
    client, addr = s.accept()
    print('connection from ', addr)
    while True:
        data = client.recv(1024)
        if not data:
            break

        try:
            rsp = table[data.decode()]
        except:
            client.send(b'Try again')
        else:
            client.send(rsp.encode())

    client.close()
```

# 간단한 요청 처리 서버/클라이언트 프로그램

## ■ 클라이언트 동작

- 1~10까지의 숫자를 서버로 전송하고, 응답을 받으면 출력
- 'q'를 입력하면, 클라이언트 종료

simple\_number\_client.py

```
from socket import *

s = socket(AF_INET, SOCK_STREAM)
s.connect(('localhost', 3333))

while True:
    msg = input('Number to send (1~10):')
    if msg == 'q':
        break

    s.send(msg.encode())

    print('Received message:', s.recv(1024).decode())

s.close()
```

```
Number to send (1~10):1
Received message: one
Number to send (1~10):10
Received message: ten
Number to send (1~10):11
Received message: Try again
Number to send (1~10):q
(venv) PS C:\Users\dhkim\net_program>
```

## 과제 3: TCP 계산기

### ■ TCPcalculator 서버/클라이언트 프로그램 작성하기

#### ● 클라이언트

- ✓ 사용자로부터 “20 + 17” 형태의 계산식을 입력 받음
  - 피연산자(숫자): 정수
  - 지원 연산: 더하기, 빼기, 곱하기, 나누기(소수점 1자리까지 표시)
  - 피연산자와 연산자 사이에 공백이 있어도 되고, 없어도 됨
- ✓ 입력 받은 계산식을 서버로 전송
- ✓ 서버로부터 전송 받은 결과를 화면에 출력
- ✓ 사용자가 “q”를 입력하면 종료

#### ● 서버

- ✓ 클라이언트로부터 계산식을 수신
- ✓ 문자열을 파싱한 후, 계산식을 계산
- ✓ 계산결과를 클라이언트로 전송

## 과제 3: TCP 계산기

### ■ 과제3

- 서버와 클라이언트를 각각 1개의 소스 코드(.py)로 저장 (2개)
- 서버와 클라이언트의 실행 결과 캡처 (1개 또는 2개)
  - ✓ 서버와 클라이언트 실행 결과를 각각 캡처해도 되고, 서버와 클라이언트 실행 결과를 1개의 파일로 캡처해도 됨

### ■ 소스 코드

- GitHub에 hw3 폴더 생성 후, 소스 코드 업로드
- GitHub 화면 캡처
  - ✓ 폴더 이름과 파일 이름이 보이도록 캡처할 것

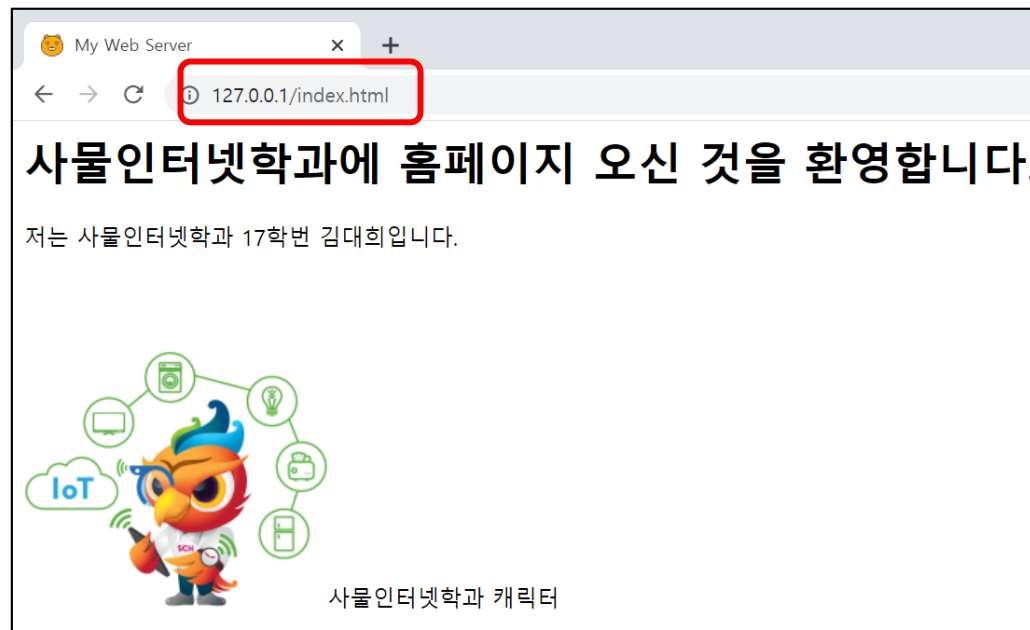
### ■ 제출

- 과제 공지 후 1주일, LMS 제출
- 제출물
  - ✓ 실행 결과 캡처 파일 (1개 또는 2개)
  - ✓ GitHub 화면 캡처 파일 (1개)

## 과제 4: 웹 서버

### ■ 간단한 웹 서버 프로그램 작성

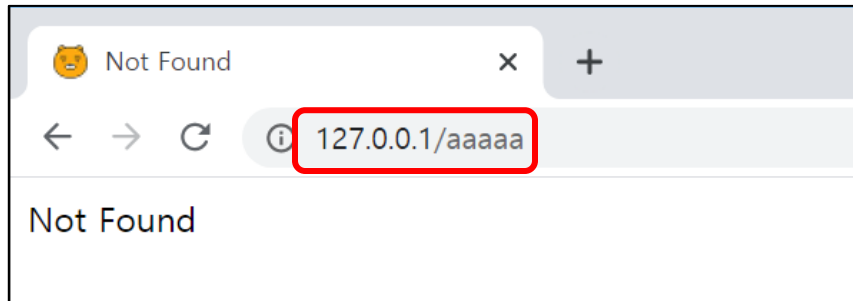
- 클라이언트로부터 HTTP 요청이 들어오면, 해당 요청에 따라 자원을 전송해주는 웹 서버 구현
- 웹 서버에서 처리 가능한 자원은 'index.html'
  - ✓ 웹 브라우저는 'index.html' 파일을 참조하여 'iot.png'를 추가 요청
  - ✓ 웹 브라우저 자체적으로 'favicon.ico' 파일을 요청할 수 있음
- URL '<http://127.0.0.1/index.html>'만 처리 가능하다고 가정





## 과제 4: 웹 서버

- 처리 불가능한 자원 요청 시, “Not Found”를 html로 전송하여 출력되도록 함
- 즉, “<http://127.0.0.1/aaaaa>” 클라이언트 입력 시, 다음과 같이 “Not Found”를 출력하여야 함



- 전송 객체(파일)
  - ✓ [index.html](#): 웹 페이지를 구성하는 html 파일
    - 이름/학번 부분을 “본인 이름”, “본인 학번”으로 변경
  - ✓ [iot.png](#): IoT 캐릭터 그림 파일
  - ✓ [favicon.ico](#): 웹 브라우저의 주소창에 표시되는 대표 아이콘
  - ✓ 과제 게시판에 첨부된 파일을 다운로드 받아서, 웹 서버 프로그램이 있는 폴더로 복사

# 과제 4: 웹 서버

## ■ 동작

- 웹 서버 소켓(포트번호 80)을 열고 웹 클라이언트의 연결을 기다림
- 클라이언트의 연결이 들어올 경우, 클라이언트와 통신할 소켓으로부터 HTTP Request의 첫 번째 라인(요청 라인)을 읽어 들임
  - ✓ 요청 라인은 “GET /index.html HTTP/1.1”의 형식을 가짐
- 요청하는 자원(파일 이름)을 나타내는 “/index.html”을 파싱한 후, “/” 제거하여 파일 이름(filename이라고 가정)을 얻음
- 해당 파일이 존재하는 경우 해당 파일을 열고, 각 파일에 대한 mimeType 설정
  - index.html 파일인 경우
    - f = open(filename, 'r', encoding='utf-8')
    - mimeType = 'text/html'
  - iot.png 파일인 경우
    - f = open(filename, 'rb')
    - mimeType = 'image/png'
  - favicon.ico 파일인 경우
    - f = open(filename, 'rb')
    - mimeType = 'image/x-icon'

mimeType

text/html, image/png  
처럼 전송되는 객체가  
무엇인지를 나타내 줌

## 과제 4: 웹 서버

- 해당 파일이 존재하는 경우, HTTP Response 메시지를 다음과 같이 생성하여 전송
  - ✓ HTTP 헤더 전송
    - 'HTTP/1.1 200 OK\r\n'
    - 'Content-Type: ' + mimeType + '\r\n'
    - '\r\n'
  - ✓ HTTP 바디 전송
    - 파일을 읽어서 전송
    - 파일 읽는 방법: `data = f.read()`
    - 파일 전송 방법
      - index.html 파일(한글 텍스트 파일)의 경우: `c.send(data.encode('euc-kr'))`
      - 그 외 파일(바이너리 파일)의 경우: `c.send(data)`
- 해당 파일이 존재하지 않는 경우, HTTP Response 메시지(헤더+바디)를 다음과 같이 생성하여 전송
  - ✓ 'HTTP/1.1 404 Not Found\r\n'
  - ✓ '\r\n'
  - ✓ '<HTML><HEAD><TITLE>Not Found</TITLE></HEAD>'
  - ✓ '<BODY>Not Found</BODY></HTML>'

## 과제 4: 웹 서버

### ■ Sample 코드

```
from socket import *

s = socket()
s.bind(('', 80))
s.listen(10)

while True:
    c, addr = s.accept()

    data = c.recv(1024)
    msg = data.decode()
    req = msg.split('\r\n')

    # 웹 서버 코드 작성
    # 각 객체(파일 또는 문자열) 전송 후, 소켓 닫기(c.close())
```

## 과제 4: 웹 서버

### ■ 과제4

- 작성한 웹 서버 코드를 1개의 소스 코드(.py)로 저장
- 실행 결과 캡처화면 (정상 화면 1개, Not Found 화면 1개)

### ■ 소스 코드

- GitHub에 hw4 폴더 생성 후, 소스 코드 업로드
- GitHub 화면 캡처
  - ✓ 폴더 이름과 파일 이름이 보이도록 캡처할 것

### ■ 제출

- 과제 공지 후 2주일, LMS 제출
- 제출물
  - ✓ 실행 결과 캡처 파일 (2개)
  - ✓ GitHub 화면 캡처 파일 (1개)

# 사용자 모듈을 이용한 TCP 서버 프로그램

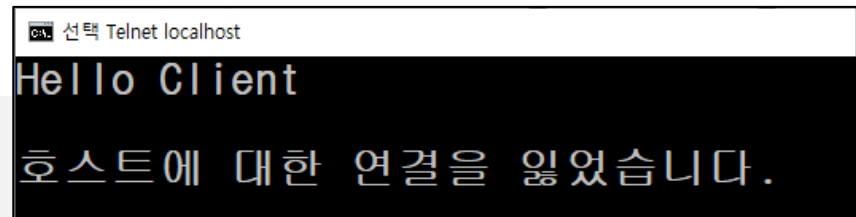
## ■ TCP 서버용 사용자 모듈 작성

- 소켓 생성과 연결을 모듈로 구현
- 이후, TCP 서버 프로그램 구현 시 재사용 가능
- `MyTCPServer.py`

```
class TCPServer:
    def __init__(self, port):
        import socket
        self.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.sock.bind(('', port))
        self.sock.listen(5)

    def Accept(self):
        self.c_sock, self.c_addr = self.sock.accept()
        return self.c_sock, self.c_addr

if __name__ == '__main__':
    sock = TCPServer(8888)
    c, addr = sock.Accept()
    print('connected by ', addr)
    c.send(b'Hello Client')
    c.close()
```



```
선택 Telnet localhost
Hello Client
호스트에 대한 연결을 잃었습니다.
```

# 사용자 모듈을 이용한 에코 서버 프로그램

## ■ 사용 방법

- MyTCPServer를 import
- TCPServer 객체 생성 후 Accept() 메소드 호출

```
import MyTCPServer as my
port = 2500
BUFSIZE = 1024
sock = my.TCPServer(port)
conn, addr = sock.Accept()
print('connected by', addr)
```

```
while True:
    data = conn.recv(BUFSIZE)
    if not data:
        break
    print("Received message: ", data.decode())
    conn.send(data)

conn.close()
```

echo\_server\_using\_module.py

```
(venv) PS C:\Users\dhkim\net_program> python echo_server_using_module.py
connected by ('127.0.0.1', 55718)
Received message: Do you understand me?
Received message: Of course!
█
```

클라이언트는 기존의  
echo\_client.py를 사용하면 됨

# 간단한 서버 접속 함수: `create_connection()`

## ■ `create_connection()`

- 클라이언트에서 TCP 소켓 생성과 연결 요청을 한번에 수행해주는 함수
- `socket()`과 `connect()`가 합쳐진 것

## ■ 에코 클라이언트 프로그램

```
import socket
BUFSIZE = 1024

s = socket.create_connection(('localhost', 2500))

while True:
    msg = input("Message to send: ")
    s.send(msg.encode())
    data = s.recv(BUFSIZE)
    if not data:
        break
    print("Received message: %s" % data.decode())

s.close()
```

`cc_echo_client.py`

```
(venv) PS C:\Users\dhkim\net_program> python cc_echo_client.py
Message to send: Hi!
Received message: Hi!
Message to send: Please finish this class as soon as possible.
Received message: Please finish this class as soon as possible.
Message to send: 
```

서버는 기존의  
`echo_server.py`를 사용하면 됨



# 간단한 서버 생성 함수: `create_server()`

## ■ `create_server()`

- 서버에서 TCP 소켓 생성, `bind`, `listen`을 한번에 수행해주는 함수
- `socket()`, `bind()`, `listen()`이 합쳐진 것

## ■ 에코 서버 프로그램

```
from socket import *
```

```
port = 2500
```

```
BUFSIZE = 1024
```

```
cs_echo_server.py
```

```
sock = create_server('', port, family=AF_INET, backlog=1)
```

```
conn, (remotehost, remoteport) = sock.accept()
```

```
print('connected by', remotehost, remoteport)
```

```
while True:
```

```
    data = conn.recv(BUFSIZE)
```

```
    if not data:
```

```
        break
```

```
    print("Received message: ", data.decode())
```

```
    conn.send(data)
```

```
conn.close()
```

```
(venv) PS C:\Users\dhkim\net_program> python cs_echo_server.py
connected by 127.0.0.1 49902
Received message:  Hi!
Received message:  I am hungry!!!
█
```

클라이언트는 기존의  
`echo_client.py`를 사용하면 됨

# 명령행 인자 에코 서버 프로그램

## ■ 명령행 인자 에코 서버 프로그램

- 서버의 포트 번호를 명령행 인자로 입력받음
- 예) python cmd\_server.py 5555

```
from socket import *  
import sys
```

cmd\_server.py

```
port = 2500  
BUFSIZE = 1024
```

```
if len(sys.argv) > 1:  
    port = int(sys.argv[1])
```

```
sock = socket(AF_INET, SOCK_STREAM)  
sock.bind('', port)  
sock.listen(1)  
conn, addr = sock.accept()  
print('connected by', addr)
```

```
while True:  
    data = conn.recv(BUFSIZE)  
    if not data:  
        break  
    print("Received message: ", data.decode())  
    conn.send(data)
```

```
conn.close()
```

```
(venv) PS C:\Users\dhkim\net_program> python cmd_server.py 5555  
connected by ('127.0.0.1', 49939)  
Received message: Hello  
Received message: IoT!!!
```

클라이언트는 기존의  
echo\_client.py를 사용하면 됨

# 명령행 인자 에코 클라이언트 프로그램

## ■ 명령행 인자 에코 클라이언트 프로그램

- 서버의 IP 주소와 포트 번호를 명령행 인자로 입력받음
  - ✓ 예1) `python cmd_client.py`
  - ✓ 예2) `python cmd_client.py -s localhost`
  - ✓ 예3) `python cmd_client.py -p 8888`
  - ✓ 예4) `python cmd_client.py -s 127.0.0.1 -p 5000`

## ■ argparse 모듈

- 명령행 파싱 모듈
- 사용 방법
  - ✓ 파서 만들기
    - ArgumentParser 객체 생성
      - ArgumentParser 객체는 명령행을 파이썬 데이터형으로 파싱하는데 필요한 모든 정보를 담고 있음
  - ✓ 인자 추가하기
    - add\_argument() 메소드 호출
  - ✓ 인자 파싱하기
    - parse\_args() 메소드를 통해 인자를 파싱

# 명령행 인자 에코 클라이언트 프로그램

```
from socket import *
import argparse

s = socket(AF_INET, SOCK_STREAM)

parser = argparse.ArgumentParser()
parser.add_argument('-s', default='localhost')
parser.add_argument('-p', type=int, default=2500)
args = parser.parse_args()

s.connect((args.s, args.p))
print('connected to ', args.s, args.p)

while True:
    msg = input("Message to send: ")
    if msg == 'q':
        break
    s.send(msg.encode())
    data = s.recv(1024)
    if not data:
        break
    print('Received message: ', data.decode())

s.close()
```

cmd\_client.py

```
(venv) PS C:\Users\dhkim\net_program> python cmd_server.py 9999
connected by ('127.0.0.1', 49952)
Received message: Who are you?
Received message: I am your father!
█
```

```
(venv) PS C:\Users\dhkim\net_program> python cmd_client.py -s 127
.0.0.1 -p 9999
connected to 127.0.0.1 9999
Message to send: Who are you?
Received message: Who are you?
Message to send: I am your father!
Received message: I am your father!
Message to send: █
```

# Partial Writes/Reads

## ■ send()/recv() 함수의 문제점

- OS 버퍼, 네트워크 상태 등에 따라 부분적 쓰기/읽기가 발생할 수 있음
- send()는 실제 전송한 바이트를 반환함
- recv(BUFFER\_SIZE)의 BUFFER\_SIZE는 최대 수신가능한 크기임

```
>>> len(data)          #보내려는 data가 1000000바이트일 때,  
1000000
```

```
>>> s.send(data)        #실제 전송한 바이트는 data보다 적을 수 있음  
37722  ← 실제 전송한 바이트
```

```
>>> data = s.recv(10000) #10000바이트를 수신하려고 할 때,  
>>> len(data)           #실제 수신한 바이트는 더 적을 수 있음  
6420   ← 실제 수신한 바이트
```

## ■ send() 해결책

- 대부분의 경우 send()는 잘 동작함
- 잘 동작하지 않는 경우, sendall() 함수 사용

# Partial Writes/Reads

## ■ recv() 해결책

- 수신할 데이터의 크기(data\_size)를 알고 있다고 가정
- 해당 데이터를 모두 수신할 때까지, 반복적으로 데이터 수신

```
rx_size = 0

while rx_size < data_size:
    data = s.recv(8192)
    if not data:
        break
    rx_size += len(data)
```

## ■ 수신할 데이터의 크기는 어떻게 알 수 있을까?

- 송수신할 데이터의 크기를 미리 약속
- 데이터 송신 전 데이터의 크기를 먼저 전송

# Partial Writes/Reads

## 서버

```
from socket import *
```

```
server = create_server(('', 9999))  
conn, addr = server.accept()
```

```
conn.send(b'This is IoT world!!!')  
conn.close()
```

partial\_read\_server.py

## 클라이언트

```
from socket import *
```

partial\_read\_client.py

```
sock = create_connection(('localhost', 9999))
```

```
data_size = 20      # 수신할 데이터의 크기
```

```
rx_size = 0
```

```
total_data = []
```

```
while rx_size < data_size:
```

```
    #최대 4바이트 수신하도록 설정
```

```
    data = sock.recv(4)
```

```
    if not data:
```

```
        break
```

```
    rx_size += len(data)
```

```
    total_data.append(data.decode())
```

```
    print(total_data)
```

```
message = ''.join(total_data)
```

```
print(message)
```

```
sock.close()
```

```
(venv) PS C:\Users\dhkim\net_program> python partial_read_client.py
```

```
['This']
```

```
['This', ' is ']
```

```
['This', ' is ', 'IoT ']
```

```
['This', ' is ', 'IoT ', 'worl']
```

```
['This', ' is ', 'IoT ', 'worl', 'd!!!']
```

```
This is IoT world!!!
```

# 동영상 스트리밍하기

opencv 설치

```
python -m pip install opencv-python
```

## ■ 동영상 스트리밍 예제

- OpenCV 모듈을 이용하여, 동영상 파일을 이미지로 캡처하여 전송
- 캡처한 **이미지의 크기**를 먼저 전송하고, 그 이후 **실제 이미지**를 전송





# 동영상 스트리밍하기: 서버

streaming\_server.py

```
import socket
import cv2
import numpy as np

BUF_SIZE = 8192
LENGTH = 10
videoFile = 'test.mp4'      #현재 폴더에 있는 비디오 파일. 다른 경로에 있을 경우, 정확한 경로를 입력해야 함
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.bind(('', 5000))
sock.listen(5)

while True:
    csock, addr = sock.accept()
    print('Client is connected')
    cap = cv2.VideoCapture(videoFile)

    while cap.isOpened():
        ret, frame = cap.read()
        if ret:
            temp = csock.recv(BUF_SIZE)                #'start' 수신
            if not temp:
                break

            result, imgEncode = cv2.imencode('.jpg', frame)
            data = np.array(imgEncode)
            byteData = data.tobytes()
            csock.send(str(len(byteData)).zfill(LENGTH).encode())    #10개 문자열로 표현된 길이 전송

            temp = csock.recv(BUF_SIZE)                #'image' 수신
            if not temp:
                break

            csock.send(byteData)                        #이미지 데이터 전송
        else:
            break

    cap.release()
    cv2.destroyAllWindows()
    csock.close()
```

# 동영상 스트리밍하기: 클라이언트

```
import socket
import cv2
import numpy as np

BUF_SIZE = 8192
LENGTH = 10

sock = socket.socket(socket.AF_INET,
                     socket.SOCK_STREAM)
sock.connect(('localhost', 5000))

while True:
    sock.send(b'start')

    rx_size = 0
    data = b''
    while rx_size < LENGTH:      #이미지 크기 수신
        rx_buf = sock.recv(BUF_SIZE)
        if not rx_buf:
            break
        data = data + rx_buf
        rx_size += len(rx_buf)

    if rx_size < LENGTH:
        break

    frame_len = int(data)

    sock.send(b'image')
```

## streaming\_client.py

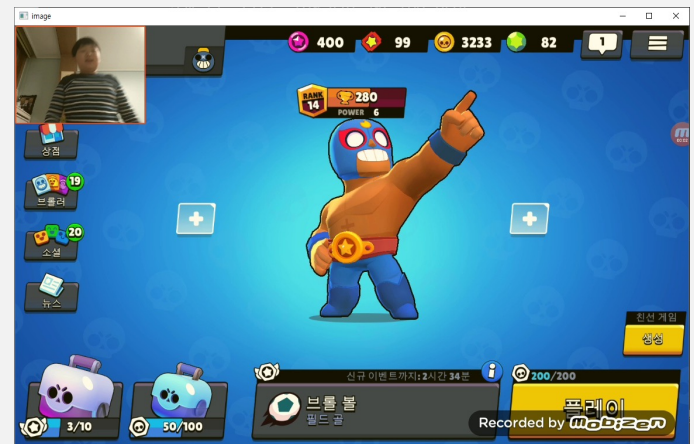
```
rx_size = 0
byteData = b''
while rx_size < frame_len: #실제 이미지 수신
    rx_buf = sock.recv(BUF_SIZE)
    if not rx_buf:
        break
    byteData = byteData + rx_buf
    rx_size += len(rx_buf)

if rx_size < frame_len:
    break

data = np.frombuffer(byteData, dtype='uint8')
imgDecode = cv2.imdecode(data, 1)
cv2.imshow('image', imgDecode)

if cv2.waitKey(1) & 0xFF == ord('q'):
    break

sock.close()
```



# TCP를 이용한 파일 송수신 프로그램

## ■ 파일 송수신 프로그램 설계

- 클라이언트: 서버에게 **파일을 요청**하여, 해당 파일을 수신
- 서버: 클라이언트의 요청에 따라 **요청 파일을 전송**
- 파일 송수신 절차 및 메시지 정의



- 예외사항 처리 (**여기서 모두 고려하지는 않으나, 실제로는 매우 중요한 부분**)
  - ✓ 클라이언트가 접속했으나 'Hello' 메시지가 오지 않는 경우
  - ✓ 클라이언트가 서버에 없는 파일을 요청한 경우
  - ✓ 클라이언트가 파일 수신 후 'Bye' 메시지를 보내지 않는 경우

# TCP를 이용한 파일 송수신 프로그램

## ■ 클라이언트 동작 설계

- 서버 접속 후 'Hello' 메시지 전송
- 서버로부터 'Filename' 메시지 수신
- 메시지 수신 후, 사용자로부터 파일이름 입력 받기
- 입력 받은 파일이름을 서버로 전송
- 서버로부터 수신할 파일의 크기 수신
- 서버로부터 수신한 내용을 저장할 파일 열기
- 서버로부터 수신한 파일 내용을 수신하여 파일에 저장
- 서버로 'Bye' 메시지 전송
- 연결 종료

## ■ 각 단계 별 예외사항 고려 필요

- 메시지가 수신되지 않거나, 다른 메시지가 오는 경우
  - ✓ 예) 서버 접속 후 'Hello' 메시지를 보낸 후, 'Filename' 메시지가 오지 않는 경우  
→ 일정 시간 기다린 후, 연결 종료
  - ✓ 예) 서버 접속 후 'Hello' 메시지를 보낸 후, 'Bye' 메시지가 오는 경우  
→ 연결 종료 (또는, 해당 메시지를 버리고, 'Filename' 메시지를 계속 기다리기)

# TCP를 이용한 파일 송수신 프로그램

## ■ 서버 동작 설계

- 클라이언트 접속 대기
- 클라이언트로부터 'Hello' 메시지 수신
- 클라이언트로 'Filename' 메시지 전송
- 클라이언트로부터 파일이름 수신
- 클라이언트로 해당 파일의 크기를 전송
- 해당 파일이름의 파일 열기
- 클라이언트로 해당 파일 내용을 전송
- 클라이언트로부터 'Bye' 메시지 수신
- 연결 종료

## ■ 각 단계 별 예외사항 고려 필요

- '파일이름'에 해당되는 파일이 존재하지 않는 경우: 연결 종료
- 메시지가 수신되지 않거나, 다른 메시지가 오는 경우
  - ✓ 예) 파일 전송 완료 후 'Bye' 메시지가 오지 않는 경우  
→ 일정 시간 기다린 후, 연결 종료

## 파일 클라이언트

file\_client.py

```

from socket import *
import sys

BUF_SIZE = 1024
LENGTH = 4          # '파일 크기': 4바이트

s = socket(AF_INET, SOCK_STREAM)
s.connect(('localhost', 7777))

s.send(b'Hello')      # 'Hello' 메시지 전송

msg = s.recv(BUF_SIZE) # 'Filename' 메시지 수신
if not msg:
    s.close()
    sys.exit()
elif msg != b'Filename':
    print('server:', msg.decode())
    s.close()
    sys.exit()
else:
    print('server:', msg.decode())

filename = input('Enter a filename: ')
s.send(filename.encode()) # 파일 이름 전송

msg = s.recv(BUF_SIZE) # 파일 크기 수신
if not msg:
    s.close()
    sys.exit()
elif msg == b'Nofile':
    print('server:', msg.decode())
    s.close()
    sys.exit()

```

```

else:
    rx_size = len(msg)
    data = msg
    while rx_size < LENGTH:
        msg = s.recv(BUF_SIZE)
        if not msg:
            s.close()
            sys.exit()
        data = data + msg
        rx_size += len(msg)
    if rx_size < LENGTH:
        s.close()
        sys.exit()
    filesize = int.from_bytes(data, 'big')
    print('server:', filesize) # 4바이트

rx_size = 0
f = open(filename, 'wb') # 파일 열기
while rx_size < filesize: # 실제 파일 수신
    data = s.recv(BUF_SIZE)
    if not data:
        break
    f.write(data)
    rx_size += len(data)

if rx_size < filesize:
    s.close()
    sys.exit()

print('Download complete')
s.send(b'Bye') # 'Bye' 메시지 전송
f.close()
s.close()

```

## 파일 서버

file\_server.py

```

from socket import *
import os

BUF_SIZE = 1024
LENGTH = 4          # '파일 크기': 4바이트

sock = socket(AF_INET, SOCK_STREAM)
sock.bind(('', 7777))
sock.listen(10)
print('File server is running...')

while True:
    conn, addr = sock.accept()

    msg = conn.recv(BUF_SIZE) # 'Hello' 메시지 수신
    if not msg:
        conn.close()
        continue
    elif msg != b'Hello':
        print('client:', addr, msg.decode())
        conn.close()
        continue
    else:
        print('client:', addr, msg.decode())
    # 'Filename' 메시지 전송
    conn.send(b'Filename')
    # 파일 이름 수신
    msg = conn.recv(BUF_SIZE)
    if not msg:
        conn.close()
        continue
    filename = msg.decode()
    print('client:', addr, filename)

```

```

try:
    filesize = os.path.getsize(filename)
except:
    conn.send(b'Nofile')
    conn.close()
    continue
else: # 파일 크기 전송
    fs_binary = filesize.to_bytes(LENGTH, 'big')
    conn.send(fs_binary)

f = open(filename, 'rb') # 파일 열기
data = f.read()          # 파일 읽기
conn.sendall(data)       # 파일 전송

msg = conn.recv(BUF_SIZE) # 'Bye' 메시지 수신
if not msg:
    pass
else:
    print('client:', addr, msg.decode())

f.close()
conn.close()

```

# 실행 결과

```
(venv) PS C:\Users\dhkim\net_program> python file_server.py
File server is running...
cleint: ('127.0.0.1', 61972) Hello
client: ('127.0.0.1', 61972) iot.mp4
cleint: ('127.0.0.1', 61973) Hello
client: ('127.0.0.1', 61973) test.mp4
client: ('127.0.0.1', 61973) Bye
█
```

서버

```
(venv) PS C:\Users\dhkim\net_program> cd ..
(venv) PS C:\Users\dhkim> python net_program\file_client.py
server: Filename
Enter a filename: iot.mp4
server: No file
(venv) PS C:\Users\dhkim> python net_program\file_client.py
server: Filename
Enter a filename: test.mp4
server: 92952225
Download complete
(venv) PS C:\Users\dhkim> ls test.mp4
```

디렉터리 : C:\Users\dhkim

Mode	LastWriteTime	Length	Name
----	-----	-----	----
-a----	2022-03-12 오후 5:11	92952225	test.mp4

클라이언트



## 과제 5: 2개의 IoT 디바이스로부터 데이터 수집하기

### ■ 디바이스 1: 온도, 습도, 조도를 측정 제공

- 사용자로부터 'Request' 메시지를 수신하면 온도, 습도, 조도를 전송
  - ✓ 온도: 0~40 사이의 임의의 정수를 반환
  - ✓ 습도: 0~100 사이의 임의의 정수를 반환
  - ✓ 조도: 70~150 사이의 임의의 정수를 반환
- 온도, 습도, 조도 전송 방법은 자유롭게 선택
  - ✓ 각각의 정보를 따로(3번) 전송해도 되고, 한 번에 3개의 정보를 전송해도 됨
  - ✓ 문자열로 전송하거나 정수로 전송하여도 됨. 단, 정수 전송 시 엔디언 고려
- 사용자로부터 'quit' 메시지를 수신하면 종료

### ■ 디바이스 2: 심박수, 걸음수, 소모칼로리를 측정 제공

- 사용자로부터 'Request' 메시지를 수신하면 심박수, 걸음수, 소모 칼로리를 전송
  - ✓ 심박수: 40~140 사이의 임의의 정수를 반환
  - ✓ 걸음수: 2000~6000 사이의 임의의 정수를 반환
  - ✓ 소모칼로리: 1000~4000 사이의 임의의 정수를 반환
- 심박수, 걸음수, 소모칼로리 전송 방법은 자유롭게 선택
  - ✓ 각각의 정보를 따로(3번) 전송해도 되고, 한 번에 3개의 정보를 전송해도 됨
  - ✓ 문자열로 전송하거나 정수로 전송하여도 됨. 단, 정수 전송 시 엔디언 고려
- 사용자로부터 'quit' 메시지를 수신하면 종료

# 과제 5: 2개의 IoT 디바이스로부터 데이터 수집하기

## ■ 사용자

### ● 2개의 IoT 디바이스와 TCP 연결

- ✓ 사용자가 '1'을 입력하면 '디바이스 1'로 'Request' 메시지를 전송하여 데이터 수집
- ✓ 사용자가 '2'를 입력하면 '디바이스 2'로 'Request' 메시지를 전송하여 데이터 수집
- ✓ 사용자가 'quit'을 입력하면 '디바이스 1', '디바이스 2'로 'quit' 메시지를 전송 후 종료

### ● 수집한 데이터는 시간정보를 추가하여 파일에 저장 (파일이름: data.txt)

- ✓ 저장방법: 1줄에 1개의 Device의 데이터 저장(아래 저장 형식 사용)
  - Fri Mar 11 22:55:13 2022: Device1: Temp=20, Humid=50, lillum=100
  - Fri Mar 11 22:57:40 2022: Device2: Heartbeat=80, Steps=2500, Cal=2100
- ✓ 데이터 수집 파일은 10개(디바이스 당 5개) 이상의 결과를 포함하고 있어야 함

# 과제 5: 2개의 IoT 디바이스로부터 데이터 수집하기

## ■ 과제5

- 디바이스1, 디바이스2, 사용자 각각 1개의 소스 코드(.py)로 저장
- data.txt 파일에 수집한 데이터 저장

## ■ 소스 코드

- GitHub에 hw5 폴더 생성 후, 소스 코드 업로드
- GitHub 화면 캡처
  - ✓ 폴더 이름과 파일 이름이 보이도록 캡처할 것

## ■ 제출

- 과제 공지 후 1주일, LMS 제출
- 제출물
  - ✓ data.txt 파일 (1개)
  - ✓ GitHub 화면 캡처 파일 (1개)

# Thank you

Questions?

Contact: [daeheekim@sch.ac.kr](mailto:daeheekim@sch.ac.kr)