

lotto_client.py 구현

lotto_client.py 인자 값 default 지정

```
1  if __name__ == '__main__':
2      import argparse
3
4      parser = argparse.ArgumentParser()
5
6      parser.add_argument('--address', type=str,
7                          default='127.0.0.1', help='Server address')
8
9      parser.add_argument('--port', type=int, default=3034,
10                          help='Server port')
11
12  FLAGS, _ = parser.parse_known_args()
13
14  main()
```

루프백 ip주소 127.0.0.1 와 3040 포트번호를 기본 인자값으로 줍니다.

lotto_client main() 함수 및 핵심 로직

다음은 client의 main() 함수의 구현입니다.

```

1  def main():
2      sock = socket.socket(socket.AF_INET,
        socket.SOCK_DGRAM)
3      print(f'UDP Lotto Client Ready ({FLAGS.address}:
        {FLAGS.port})')

```

`socket.AF_INET` 로 IPv4 주소 체계를 지정하고, `socket.SOCK_DGRAM` 로 UDP 소켓을 생성합니다.

다음으로 `lotto_client` 의 핵심 로직입니다.

기능은 다음과 같습니다.

- `exit`을 입력 받으면 현재 client 프로세스를 종료합니다.
- `quit`을 입력 받으면 server에 `quit` 메시지를 보냅니다. 이를 수신한 server 프로세스는 종료되도록 로직이 구성되어있습니다.
- 입력 받은 숫자는 최대 6개로 제한합니다.
- 입력 받은 숫자가 1~45 사이의 숫자가 아니면 다시 입력을 받습니다.
- 입력 받은 값이 숫자가 아니면 예외처리를 합니다.
- 위 과정을 무사히 통과하면 `sock.sendto` 로 입력 받은 숫자를 `server` 로 전송합니다.
- 이후, `server` 로부터 응답을 받는 과정에서 문제가있다면 예외를 발생시킵니다.

```

1  while True:
2      data = input("공백을 기준으로 최대 6개까지의 로또 번호
        를 입력(1~45) \n exit : client exit | quit : server exit
        \n 입력:").strip()
3
4      if data.lower() == 'exit':
5          print("Exiting client.")
6          break

```

```

7
8     try:
9         if data.lower() == 'quit':
10             sock.sendto(b'quit', (FLAGS.address,
11                 FLAGS.port))
12                 continue
13
14         nums = [int(n) for n in data.split()]
15
16         if len(nums) > 6:
17             print("로또 번호는 최대 6개까지 입력 가능합니
18 다.\n")
19             continue
20
21         if not all(1 <= n <= 45 for n in nums):
22             print("입력된 번호는 1부터 45 사이여야 합니
23 다.\n")
24             continue
25
26     except ValueError:
27         print("숫자만 입력해야 합니다.\n")
28         continue
29
30     sock.sendto(data.encode('utf-8'), (FLAGS.address,
31         FLAGS.port))
32     print(f"Sent: {data}")
33
34     try:
35         response, server = sock.recvfrom(2**16)
36         print(f"Received: {response.decode('utf-
37 8')}}\n")
38
39     except :
40         print("서버의 응답을 받는 과정에서 문제 발생\n")
41         break

```

lotto_server.py 구현

lotto_server 기본 인자 값 지정

```

1  if __name__ == '__main__':
2      import argparse
3
4      parser = argparse.ArgumentParser()
5
6      parser.add_argument('--address', type=str,
7                          default='127.0.0.1')
8      parser.add_argument('--port', type=int, default=3034)
9
10     FLAGS, _ = parser.parse_known_args()
11     main()

```

마찬가지로 `address = 127.0.0.1`, `port = 3034` 를 기본 값으로 지정해줍니다.

다음은 `lotto_server` 의 핵심 로직 구현입니다.

lotto_server main() 함수

```

1  def main():
2      sock = socket.socket(socket.AF_INET,
3                          socket.SOCK_DGRAM)
4      sock.bind((FLAGS.address, FLAGS.port))
5      print(f'Listening on {sock}')

```

동일하게 IPv4 주소체계로 UDP 소켓을 생성 후, 인자로 전달 받은 ip주소와 port번호를 기반으로 bind 합니다.

lotto_server 의 핵심 로직은 다음과 같습니다.

while True 반복문 동안, sock.recvfrom(2**16) 으로, 클라이언트로부터 받은 값은 data 변수에 저장하고, 이를 utf-8 방식으로 decode 해줍니다.

```
1  while True:
2      data, client = sock.recvfrom(2**16)
3      data = data.decode('utf-8').strip()
4      print(f'Received {data} from {client}')
5
6      if(data == 'quit') :
7          sock.sendto(b'quit this server', client)
8          break
```

이후, data 값이 quit 이면 현재 반복문을 종료하여 server프로세스를 종료하게 합니다.

quit 이외의 data 값은 항상 숫자임을 client 쪽에서 검증하여 전송하기에, 이후 로직에서는 항상 data 값이 숫자 문자열임을 전제로 동작합니다.

client 로부터 전달 받은 번호를 기반으로 로또 번호를 생성하는 로직은 다음을 수행합니다.

- 전달 받은 번호 중 중복이 있다면, 이를 제거하고 생성
- 전달 받을 수 있는 번호의 갯수는 중복을 제거하고 0~6 개
- 0~6 개의 전달 받은 로또 번호와 6-(전달 받은 로또 갯수) 개의 랜덤한 번호를 조합하여 이를 client에게 전송

먼저 입력 받은 data = 수열을 set 으로 중복을 제거한 형태로 user_numbers 에 저장합니다.

로또 번호는 중복된 번호의 조합을 허용하지 않으니, `lotto_pool` 은 아래와 같이 `1~45 set` 으로부터 `user_numbers` 를 빼주어 만들어줍니다. 이 `lotto_pool` 로부터 `needed = 6 - len(user_numbers)` 만큼 랜덤하게 번호를 뽑아 `list` 로 `additional` 에 저장합니다.

```
1 user_numbers = set(int(n) for n in data.split())
2
3 lotto_pool = list(set(range(1, 46)) - set(user_numbers))
4 needed = 6 - len(user_numbers)
5 additional = random.sample(lotto_pool, needed) if needed >
  0 else []
```

이후, `user_numbers` 와 `lotto_pool` 로부터 랜덤하게 뽑은 수열을 합한 `final_numbers` 를 만들고 이를 다시 `random.shuffle` 로 번호의 순서를 섞어줍니다.

```
1
2     final_numbers = list(user_numbers) + additional
3     random.shuffle(final_numbers)
4     print(f'Selected Lotto numbers: {final_numbers}')
```

이후, 위와 같이 생성된 `final_numbers` 를 `response` 로 `client` 에 전송합니다.

```
1 response = 'Lotto: ' + ' '.join(map(str, final_numbers))
2     print(f'sending client... {response}')
3
4     sock.sendto(response.encode('utf-8'), client)
```

실제 동작 예시

lotto_server.py

```
이문영@DESKTOP-DVQT6QL MINGW64 /c/Users/이문영/Desktop/3-1/01. 데이터 통신/데이터 통신 실습/week11
$ python lotto_server.py
Listening on <socket.socket fd=308, family=2, type=2, proto=0, laddr=('127.0.0.1', 3034)>
Received 12 34 from ('127.0.0.1', 64665)
Selected Lotto numbers: [12, 43, 26, 34, 42, 15]
sending client... Lotto: 12 43 26 34 42 15
□
```

lotto_client.py 전송 번호 : 12 , 34

```
이문영@DESKTOP-DVQT6QL MINGW64 /c/Users/이문영/Desktop/3-1/01. 데이터 통신/데이터 통신 실습/week11
$ python lotto_client.py
UDP Lotto Client Ready (127.0.0.1:3034)
공백을 기준으로 최대 6개까지의 로또 번호를 입력(1~45)
exit : client exit | quit : server exit
입력:12 34
Sent: 12 34
Received: Lotto: 12 43 26 34 42 15
```

수신 받은 번호 12 43 26 34 42 15

정상적으로 12 와 34 를 포함한 로또 번호를 server가 생성하여 client가 무사히 수신받고 있습니다.

1 1 1 5 5 와 같이 중복된 숫자를 포함한 로또 번호를 서버측에서 중복을 제거하고 올바르게 1 과 5 를 포함한 랜덤한 로또 번호 6자리를 생성하고 전달 받은 모습입니다.

lotto_client.py

```
공백을 기준으로 최대 6개까지의 로또 번호를 입력(1~45)
exit : client exit | quit : server exit
입력:1 1 1 5 5
Sent: 1 1 1 5 5
Received: Lotto: 24 14 5 1 36 3
```

lotto_server.py

```
Received 1 1 1 5 5 from ('127.0.0.1', 64665)
Selected Lotto numbers: [24, 14, 5, 1, 36, 3]
sending client... Lotto: 24 14 5 1 36 3
□
```

다음은 client에서 5 5 라는 수열을 보냈을 때, wireshark 캡처입니다.

udp.port == 3034					
No.	Time	Source	Destination	Protocol	Length Info
622	18.581094	127.0.0.1	127.0.0.1	UDP	35 64665 → 3034 Len=3
623	18.582669	127.0.0.1	127.0.0.1	UDP	55 3034 → 64665 Len=23

▶ Frame 622: 35 bytes on wire (280 bits), 35 bytes captured (280 bits) on interface \Device\NPF_{Loopback},
▶ Null/Loopback
▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
▶ User Datagram Protocol, Src Port: 64665, Dst Port: 3034
▶ Data (3 bytes)

0000 02 00 00 00 45 00 00 1f 0d 5c 00 00 80 11 00 00E.....
0010 7f 00 00 01 7f 00 00 01 fc 99 0b da 00 0b 8f 41A
0020 35 20 35 5 5

udp.port == 3034					
No.	Time	Source	Destination	Protocol	Length Info
622	18.581094	127.0.0.1	127.0.0.1	UDP	35 64665 → 3034 Len=3
623	18.582669	127.0.0.1	127.0.0.1	UDP	55 3034 → 64665 Len=23

▶ Frame 623: 55 bytes on wire (440 bits), 55 bytes captured (440 bits) on interface \Device\NPF_{Loopback},
▶ Null/Loopback
▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
▶ User Datagram Protocol, Src Port: 3034, Dst Port: 64665
▶ Data (23 bytes)

0000 02 00 00 00 45 00 00 33 0d 5d 00 00 80 11 00 003.....
0010 7f 00 00 01 7f 00 00 01 0b da fc 99 00 1f 47 aaG
0020 4c 6f 74 74 6f 3a 20 32 34 20 31 31 20 35 20 33 Lotto: 2 4 11 5 3
0030 31 20 31 32 20 33 39 1 12 39