# TCP socket programming(client)

## Introduction

This assignment introduces you to writing simple TCP clients using the socket interface of the operating system. We will be working with the socket interface provided by the Linux OS. Be sure to read the submission guidelines for the assignment and remember the rules applicable to all assignments.

## Question 1: Writing a TCP client

Writing a client is easier than a server to some extent so we'll start with that. In this assignment, you will write a TCP client that will connect to a server, read the state of an Uno game and send the following information to the server:

- Total score of all players at end of the game.

- Winner of each round of the game.

- The overall winner of the game.

## Description of Uno game

Uno is a multi-player card game. In Uno, there are a total of 15 kinds of cards and each card has an associated point value(listed in table 1). The unique symbol used for each card by the server is also mentioned in the table.

| Card type | Point value | Symbol |
|---|---|---|
| Number card(0 - 9) | Same as card type | 0 - 9 |
| Skip card | 20 | S |
| Draw two | 20 | T |
| Reverse | 20 | R |
| Wild card | 50 | W |
| Wild and draw four | 50 | F |

Table 1: Types of cards in an Uno deck and their point values

The mechanics of the gameplay are not important for this assignment. It is sufficient to know that a game ends when a player has exhausted all cards in their hand. The first player to do so is deemed the winner of the round. Several such rounds are played to determine who is the winner of the overall game.

When a round has ended, the point values of all cards of the players are added up and that becomes the score of the winner of the round. For an example computation, see table 2.

In the round described in table 2, player 3 has exhausted all cards and thus is the winner of that round. The points scored by player 3 at end of this round is $87 + 8 = 95$. The points scored thus by winner of each round is accumulated. The player who scored maximum total points is the winner.

| Player | Deck | Deck score |
|:---:|:---:|:---:|
| 1 | 0, 2, 7, 3, 5, T, W | 87 |
| 2 | 1, 3, 4 | 8 |
| 3 | | 0 |

Table 2: Example scoring done at end of an Uno round

## Input format

The game state provided by the server consists of the following information in this exact order:

1. Number of players

2. Number of rounds in the game

3. Round ID, $R$

4. Player ID, $P$, followed by cards with player $P$ at end of round $R$. The individual cards in the deck are separated by a comma(','). The player ID and deck are separated by a colon(':'). This is repeated for all players. Note that the deck will be empty for the winner and thus will be not be included in round state.

5. '0' signifying data for round $R$ has ended and data for next round will begin.

Each of the above values are separated by pipes('||') before sending to client. This entire string is sent to the client terminated by a newline character('\n'). Here is a sample game state sent by server to client:

    3||2||1||1:2,0,S,W||3:0,1,S||0||2||2:F,6||3:4,5,W,3,0||0||

Here is an expanded version of the above input with explanation of each field:

| Field | Description |
|:---:|:---:|
| 3 | Number of players |
| 2 | Number of rounds |
| 1 | Start of state of round 1 of game. |
| 1:2,0,S,W | Cards with player 1(here - 2, 0, skip and wild) at end of current round(1). |
| 3:0,1,S | Cards with player 3 at end of current round(1). |
| 0 | End of state of round 1. |
| 2 | Start of state of round 2 of game. |
| 2:F,6 | Cards with player 2 at end of current round(2). |
| 3:4,5,W,3,0 | Cards with player 3 at end of current round(2). |
| 0 | End of state of round 2. |

Table 3: Explanation of various fields in sample input

## Output format

The server expects a JSON object as response from the client. The JSON object must contain the following information:

- **Round winners**: A dictionary mapping between round IDs and round winning player IDs.

- **Scores**: A dictionary mapping between player IDs and their final scores.

- **Overall winner**: ID of player who has won the game.

For the example game state described in input format section, the correct client response would be the JSON string

```
{"round_winners": {"1": 2, "2": 1}, "overall_winner": 1, "scores": {"1": 118,
    "2": 93, "3": 0}}
```

You can create a JSON solution string by dumping a dictionary, consisting of the necessary values in the required format, using the "json" module in python.

**Important: The server expects the JSON response to a game state on a single line terminated by a newline character('\n'). Ensure that the client you write sends the entire JSON response in a single line terminated by newline. If the JSON response is split over multiple lines, it will not be evaluated correctly.**

## How we will test your program

We will pass the IP address and port number of the server as first and second command line arguments respectively to the client. The client must read the game state from the network connection to the server and respond with the required values over the same network connection. A sample invocation is shown below:

```
$ python2 client.py 192.168.56.1 8080
```

In above invocation, the server program to which client must connect to is running at port 8080 on IP address 192.168.56.1. You can assume that the server will be running before the client is invoked.

The server will send a game state and wait for response from client before sending the next game state. The server will send '0' after receiving a response to the last game state from the client. The server will exit after this and the connection will be terminated. The client must exit cleanly after it receives '0' from server(i.e. without any errors: the exit code must be 0).

## How you can test your program

You can run your own server using netcat by running the following command:

```
$ nc localhost 8080
```

after which, you can invoke your client using the command:

```
$ python2 client.py localhost 8080
```

to connect to the netcat server started. Anything you given as input through standard input(i.e. typing) to netcat will be sent to your client. Similarly, whatever your client sends netcat server will be printed to the standard output by netcat. This is useful for testing your client.