Nesta aplicação cliente-servidor, designados respetivamente por ***cliente_tcp*** e ***server_tcp,*** temos que ter em conta que o servidor ordena grupos de clientes que pretendem ser seriados aleatoriamente para um determinado fim. Por isso cada grupo de clientes é identificado por uma identificação comum (id_processo_ordenação), sendo cada cliente identificado pela sua identificação pessoal (id_pessoal) que é distinta de todos os outros clientes. Para que o servidor comece a aceitar clientes e dê inicio ao processo de ordenação de um dado grupo de clientes existe um cliente inicial cujo id_pessoal é "_".

Vamos considerar que temos um grupo de 10 pilotos participantes numa etapa de rali para serem sorteados e vamos simular a aplicação no *localhost* no porto 12345. Para esse efeito deve executar os seguintes passos numa máquina Linux (ou na máquina virtual):

1. Iniciar um primeiro terminal (vamos designá-lo por terminal 1) e colocar em execução o servidor com a seguinte instrução:

```
$ python3 server_tcp.py 12345
```

Quando o servidor entra em execução nenhuma mensagem é escrita neste terminal.

2. Iniciar um segundo terminal (vamos designá-lo por terminal 2) e colocar em execução o cliente controlador com a seguinte instrução:

```
$ python3 client_tcp.py rali _ 12345
```

Quando este cliente entra em execução no terminal 1 é escrita uma mensagem de novo processo de ordenação:

```
Command: {'op': 'NEW', 'proc': 'rali'}
```

E no terminal 2 são escritas as seguintes mensagens:

```
{'error': ''}
Press return to proceed with ordering on process "rali"
```

Não deve fazer nada neste terminal até lançar os clientes que vão ser ordenados.

3. Iniciar um terceiro terminal (vamos designá-lo por terminal 3) e colocar em execução os 10 clientes. Para facilitar este processo em vez de lançar os clientes um a um pode e deve usar a *shell script* (run_tcp.sh) com a seguinte instrução:

```
$ bash run_tcp.sh rali 12345 10
```

Este comando vai lançar sucessivamente os clientes u10, u9, ..., u1. No terminal 1 é escrita uma mensagem de pedido de cada cliente do tipo:

```
Command: {'op': 'ADD', 'proc': 'rali', 'id': 'u10'}
...
Command: {'op': 'ADD', 'proc': 'rali', 'id': 'u1'}
```

No terminal 3 não é escrita nenhuma mensagem, todas os clientes ficam em *stand-by* à espera de ordens do servidor.

Para obter o resultado da ordenação destes 10 clientes deve selecionar o terminal 2 e fazer ***Return***.

Nesta altura no terminal 1 é escrita a mensagem de início do processo de ordenação:

Command: {'op': 'START', 'proc': 'rali'}

No terminal 2 será apresentado no monitor (ou é escrito no ficheiro de saída) a ordenação dos 10 clientes, qualquer coisa do tipo:

["u5", "u7", "u4", "u9", "u6", "u3", "u10", "u8", "u1", "u2"]

E no terminal 3 cada cliente apresenta no monitor (ou escreve no ficheiro de saída de resultados) o mesmo resultado.