1. Interface

There are 4 threads that run concurrently once a peer is initialized via `init` or `join` , namely the

1. `TCPServer` : starts an always on (while the peer is alive) TCP server and sits on the port identified by `12000+peerID` and listens to TCP connections from other peers.
2. `UDPServer` : starts a UDP server that welcomes UDP connections on the port `12000+peerID` . The UDP server has a timeout limit which is set to 2 seconds.
3. `Ping Successors` : a peer pings it's successors upon joining the network and then periodically pings them via UDP to check their existance
4. `Input Listener` : a peer always moniters screen inputs by user and hanles the queries accordingly.

2. Task Implementation

2.1 Initialization

To start the peers, execute the shell script `init_python3.sh` .

2.2 Ping Successors

In this implementation, a peer pings it's successor and responds to pings from predecessors under 3 circumstances:

2.2.1 On Initialization

A peer should ping it's successors upon initialization, that is, when the python script is called in an Xterm with 'init'. Suppose peer2 is initialized (along with other peers to form a valid chord p2p network) via the input `init 2 4 6 10` The following message will be displayed on the peer's Xterm screen:

```
> Ping requests sent to Peers 4 and 6
```

Peer2 and peer4's screens will dispaly the following message if they received the message intack, and respond to advise peer2 about it's existance:

```
> Ping request message received from Peer 2
```

If the response is safely received by peer2, it will know a certain successor is still alive and it will go and display the following message:

```
> Ping response received from Peer 4
> Ping response received from Peer 6
```

2.2.2 After Joining in

A peer pings it's successors after it has joined in the network, that is, it has find out the proper 'position' in the network. At this poin it knows who it's successors are (through the task "join" whoes details will be discussed later on) and they will perform the above mentioned operations.

2.2.3 During the Existance

While a peer is still alive, it pings it's successor periodically to check their existance. I set the ping interval to 10 seconds and this parameter is passed in as the last command line argument. As mentioned in the first section, the UDP server has a timeout set to 2 seconds, if a peer misses three consequtive ping response from a successor, it will consider this successor as absent and announce the absence by displaying the following message, the details of which will be discussed in "2.5 peer departure (abruptly)".

2.3 Peer Joining

A peer joins the network by excuting the programme file with command line arguments `join <peer> <known_peer> <ping_interval>` , assume, for example, that peer 25 wants to join via peer 4, and ping interval is 10 seconds. This can be realised by executing the following command in a new Xterm window:

```
> python3 Dht.py join 25 4 10
```

Then the program will call a function `join()` to find the right position for peer25 in the network through a sequence of TCP connections, each peer on the road will forward the joining request if it decides peer25 should not be it's successor. A message will be displayed on their screen to suggest the forwarding, for example:

```
> Peer 25 Join request forwarded to my successor
```

here we'll assume peer25's current position in the network is right after peer20, peer20 will then perform 3 operations after it received the joining request from it's predecessor:

1. informing peer 25 of the information it needs, that is, peer20's successors
2. informing it's first predecessor to update it's second successor to peer25
3. updating it's second successor to it's first successor and it's first successor to peer25

Upon the receiving of information from peer20, peer25 knows where it's position is and who it's successors are, and it can go ahead and join the network and functions normally as other peers. A message suggesting the successful joining will be dispalyed on peer25's screen:

```
> Join request has been accepted
> My first successor is Peer ...(first successor received from peer20)
> My second successor is Peer ...(second successor received from peer20)
```

2.4 Peer Departure (Graceful)

A peer gracefully leaves the network upon reading in the screen input command "Quit". Assume that peer 25 wants to quit, once it decides to leave the network, it's state turns into `False` and will inform it's absence to it's two predecessors via TCP connection. The predecessors will update their successors and annouce the update on their screens after the arrival of the information.

```
> Peer 25 will depart from the network
> My new first successor is Peer ...
> My new second successor is Peer ...
```

2.5 Peer Departure (Abrupt)

Peers may abnormally leave the network, for instance, by invoking keyboardinterruption or closing the Xterm window. As explained previously, a peer periodically pings it's successors via UDP, if one successor is absent, the ping request will not be responded and therefore the peer knows which successor might have left. Since UDP connection is not reliable by nature, in this implementaion, a peer can detect such departure of it's successor(s) by evaluating a cumulative sum of lost ping responses.

If a peer misses 3 consecutive ping responses from another peer, it considers that peer to be not alive and annouces it's absence. Then the peer will update it's successors. The `UDP_receiver()` function is implemented in such a manner that it is able to detect which (the first or the second) successor is not alive.

2.6 Data Insertion

A peer performs file insertion upon reading in the screen input command

```
> Store <file_id>
```

A peer decides to store or forward the storing request of a file by checking the hash value of the file id. The distribution of files among peers adopts the rules of chord protocol. The storing request will be forwarded along the network as long as the peer receiving the request decides that the file should not be stored here, a message like below will be displayed on the peer's screen if this is the case:

```
> Store <file_id> request forwarded to my successor
```

Once the request is received by a peer that accepts this file, it will annouce the insertion and displays a message like below:

```
> Store <file_id> request accepted
```

2.7 Data Retrieval

A peer performs data retrival upon reading in the screen input command

```
> Request <file_id>
```

As described in section 2.6, files are distibuted among peers in the network, the rule for insertion and retrieval is the same, the request will be passed along the network untill a peer claims the storage of the queried file. A message will be displayed on the screen of each peer during the forwarding:

```
> Request for File <file_id> has been received, but the file is not stored h
ere
```

Once the request is passed to the peer where the file is stored, it will send the file to the peer that initiated the retrieval request via TCP, messages suggesting such actions will be displayed on its screen:

```
> File <file_id> is stored here
> Sending file <file_id> to Peer <peer who initiated the retrieval>
> The file has been sent
```

TCP will send the file data to the peer querying the file and the peer will write the data of the whole file to a new copy with a prefix "received". The following messages will be dispalyed on its screen:

```
> Peer <the peer who has the file> had File <file_id>
> Receiving File <file_id> from Peer <the peer who has the file>
> File <file_id> received
```

After successful retrieval, there is a copy in the same folder as the original file. could make to improve the perfomance and scalability of the program.