Client:

1. Upload data:

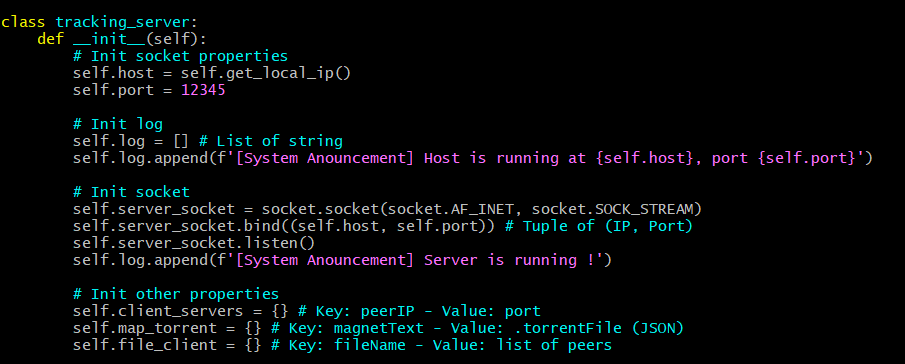
* Break data into pieces (each piece has at most 512kB).
* Create a order.JSON file that saves these information: name of data, names of files, types of datas, some tables that saves the order of pieces (each table save for one file in case there are many files in data).
* Upload the .JSON file to the server.
* Receive the IP address and port from server to transit these data file for peers in networks. Choosing to upload to the peer that have the fastest speed.
* Encode the files (optional).
* Update for the server before changing to another peers or disconect.

1. Download data:

* Upload the file.JSON to the server, then get a table about positions of each file in networks.
* For each pieces, choosing to connect to the peer that has the fastest speed if there are 2 or more peers that have the same speed, choosing the one that has most pieces.
* Dowload the order.JSON from server.
* Merge these pieces and name the complete files.
* Delete order.JSON.
* After finishing downloading the necessary files. If the clients is still online, then continue downloading some pieces from other data pieces and updating for the server.

**Server:**

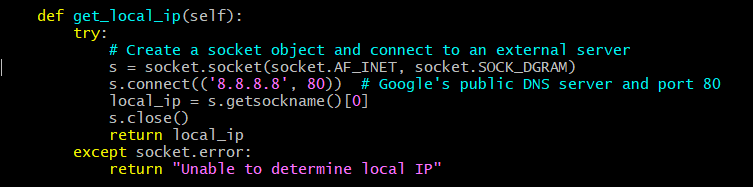
1. Server properties:



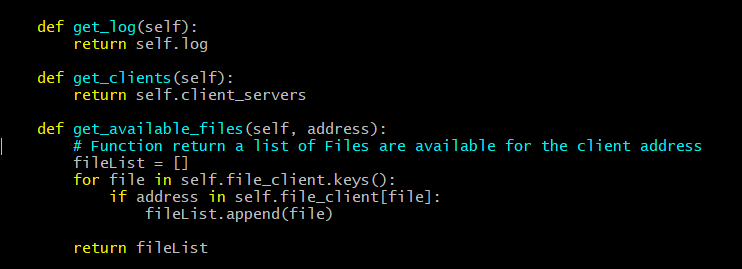
* Including **Host** and **Port** number
* **Log**: a list (vector) of string used for logging.
* **client\_servers:** demonstrates clientIP with clientPort that is active.
* **map\_torrent:** represents the magnet\_text maps with torrent\_file information.
* **file\_client:** demonstrates magnet\_text (file name) maps with client IPs.

1. Class functions:

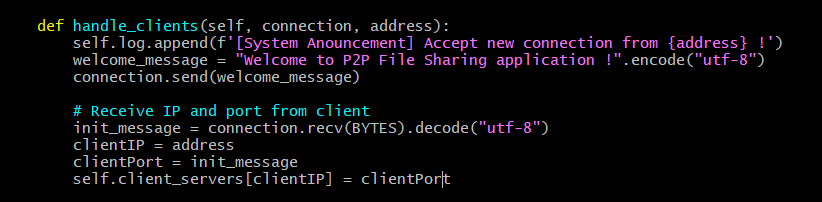
* **get\_local\_ip():** The function should return the local IP address of the server.



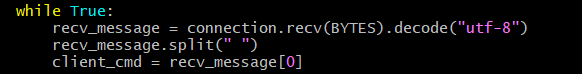
* **“GET” functions**



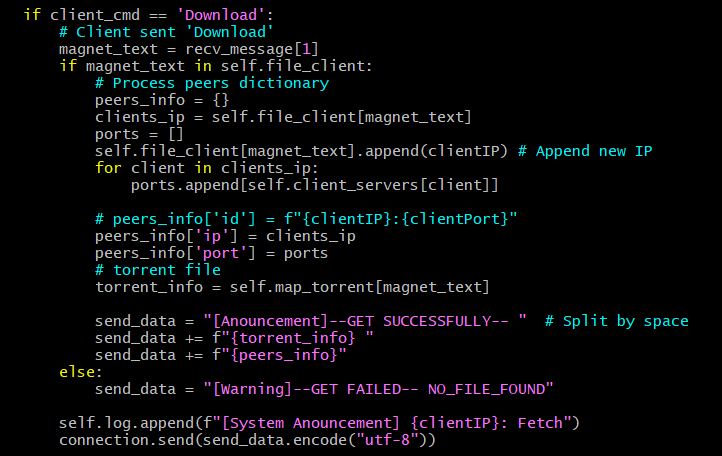
* **handle\_clients():** Function is passed into 2 parameters which are **connection** and **address**. At the first time the client (peer) connects to the server, it is required that the client should send its Port number in the format as “*12345”* (example) in string type.



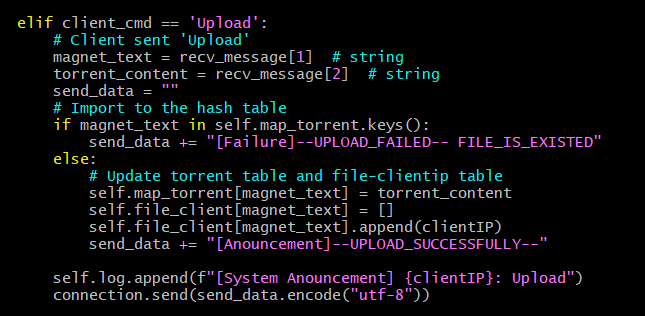
* In the next step, the server will handle the **client commands**. There are 4 main types of commands including: *“Download”, “Upload”, “Disconnect”, “Waiting”*. First step is to separate to get the command.



* **Download command:** The client requires to get the information about the file that it wants to download. **The format of the download command from client side** looks like this: “[Download] [magnet\_text]”. Example: “Download alice”means getting information about the file with the *magnet\_text* is “alice”. After applying some algorithms, the server will send the message back to the client. At the client side, it is required for the client to process this message.



* **Upload command:** The client requires to upload the information about the file so that other peers can know about the file and download it. **The format of the upload command from the client side** looks like this: “[Upload] [magnet\_text] [content of .torrent file]”. Example: “Upload alice {piece counts: 12, piece length: 512, bla bla bla}”. After applying some algorithms, the server will send the message back to the client. At the client side, it is required for the client to process this message.



* **Other commands:** Format of Disconnect command and the Waiting command from the client side looks like this: **“Disconnect”** and **“Waiting”**. Otherwise, it will count as invalid command and send an error message.

