**Client:**

1. Class file:

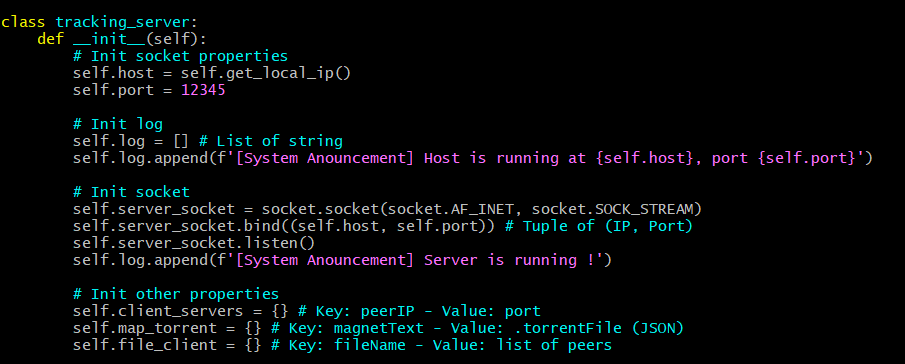
* Include the path to the file. **(variable)**
* Include the file size. **(variable)**
* Include total chunk count. **(variable)**
* Include a list of dictionaries about chunk information. For example, [{1:01001}, {2:10100}, {3,11011}]. This example shows that the overall file will have 3 chunks to combine. Each chunk has a length of 5 bytes. To make the file, just concatenate the chunk bytes => 010011010011011 will form the file we need. **(variable)**
* Have a function to determine the file size -> assign the size to the file size variable. **(function)**
* Have a function to divide the file into chunks -> assign the number of chunks to the chunk count variable. Then add those chunks’ information to the list of dictionaries above. **(function)**
* Have a function to merge all the chunks if the total count is met => only activate when the file has enough chunks to combine. **(function)**
* Have a function to detect if the file is successfully finished/complete. (**function: bool**)

1. Client (Peer) side:

* Include a list of files that the client has. **(variable)**
* Include client host (IP) and client port. **(variable)**
* Include server host (IP) and server port. **(variable)**
* Include a message for command. **(variable)**
* Include an upload path. **(variable)**
* Include a download path. **(variable)**
* Include a list to log the information - logging. **(variable)**
* set\_server\_host. **(function)**
* set\_client\_upload\_path. **(function)**
* set\_client\_upload\_path. **(function)**
* set\_client\_upload\_path. **(function)**
* get\_server\_host. **(function)**
* get\_server\_host. **(function)**
* get\_download\_dir. **(function)**
* get\_client\_host. **(function)**
* get\_message. **(function)**
* send\_chunk\_to\_client. **(function)**
* receive\_chunk\_from\_client. **(function)**
* Have a function to handle server messages and also use this function to send commands to the server. **(function)**

**Server:**

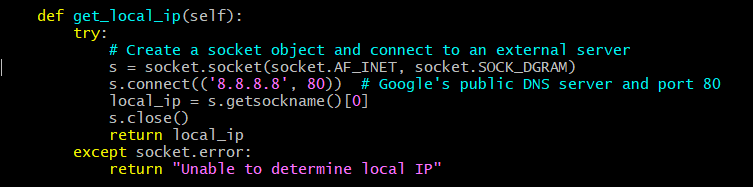
1. Server properties:



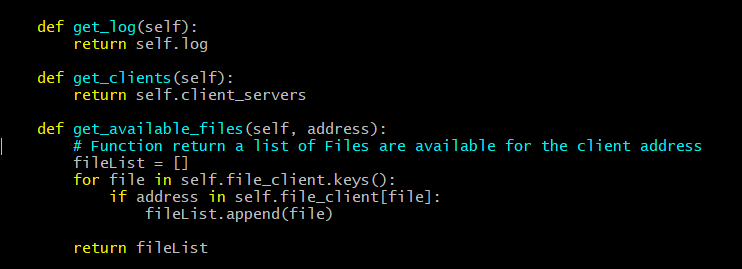
* Including **Host** and **Port** number
* **Log**: a list (vector) of string used for logging.
* **client\_servers:** demonstrates clientIP with clientPort that is active.
* **map\_torrent:** represents the magnet\_text maps with torrent\_file information.
* **file\_client:** demonstrates magnet\_text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

1. Class functions:

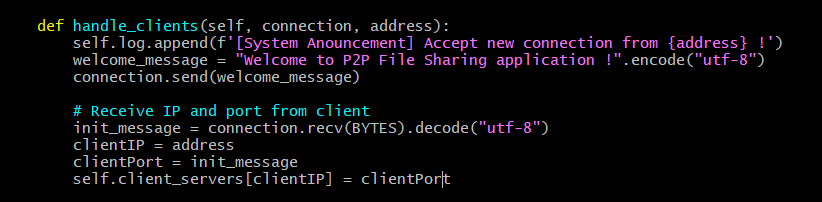
* **get\_local\_ip():** The function should return the local IP address of the server.



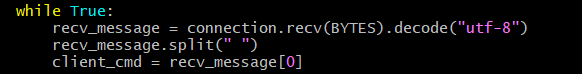
* **“GET” functions**



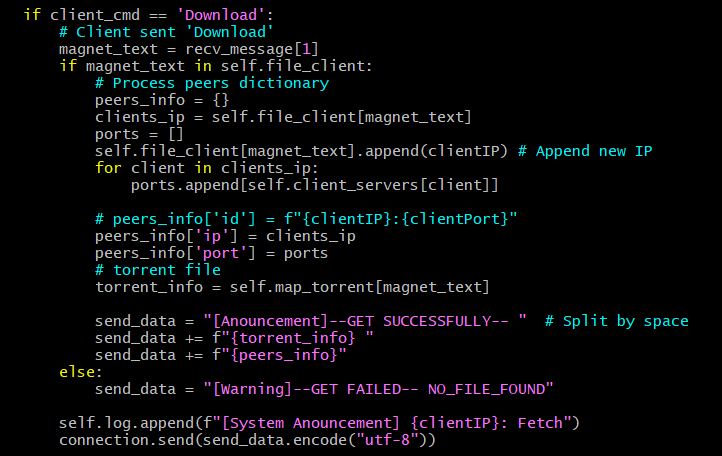
* **handle\_clients():** Function is passed into 2 parameters which are **connection** and **address**. At the first time the client (peer) connects to the server, it is required that the client should send its Port number in the format as “*12345”* (example) in string type.



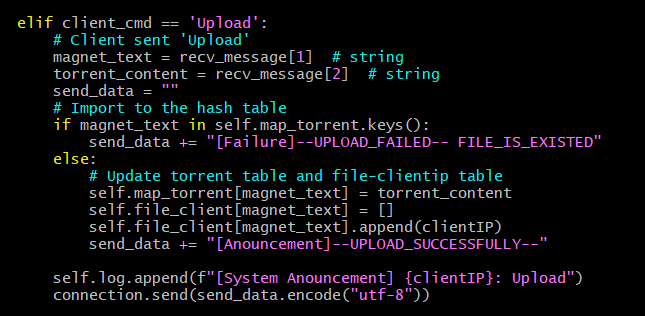
* In the next step, the server will handle the **client commands**. There are 4 main types of commands including: *“Download”, “Upload”, “Disconnect”, “Waiting”*. First step is to separate to get the command.



* **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.

