Roarke Myers, Dylan Schneider, Josias Polonia

December 8, 2019

Network Design

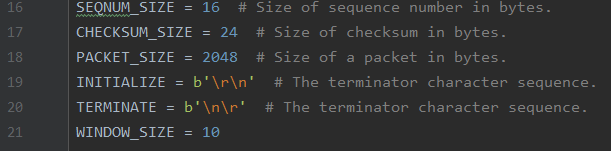
Vokkarane

Design Document: Phase 4

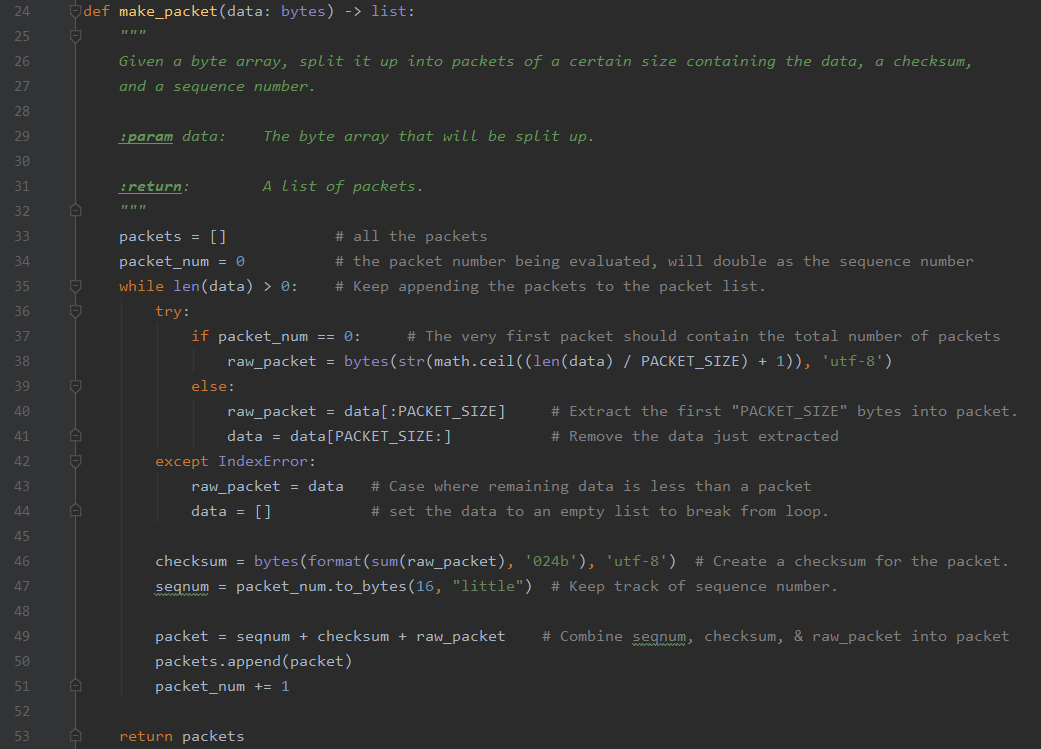
**Purpose of the Phase:**

**Code Explanation:**

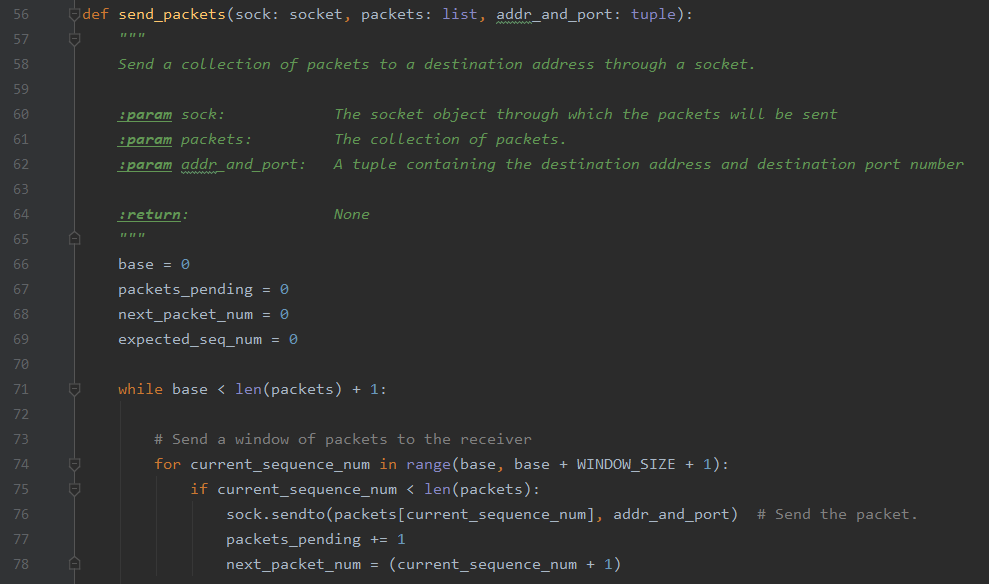
Packet\_functions.py



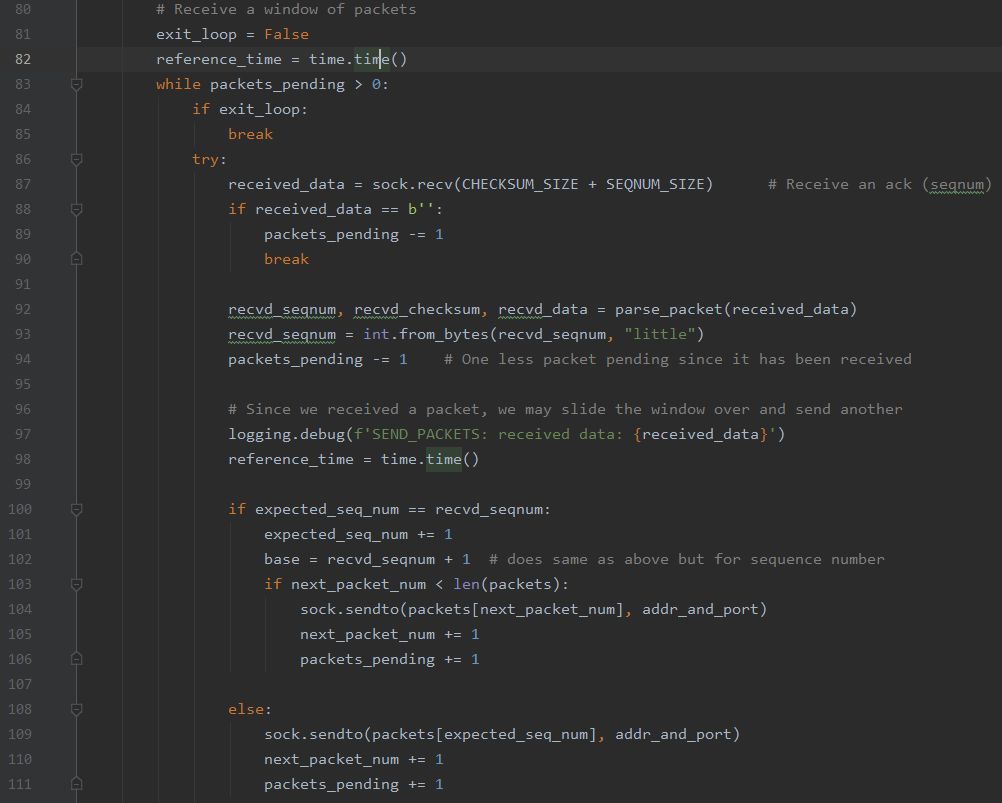
Lines 16-20 deal with initialization of sizes and character sequences. Lines 21 can be altered depending on the desired window size.



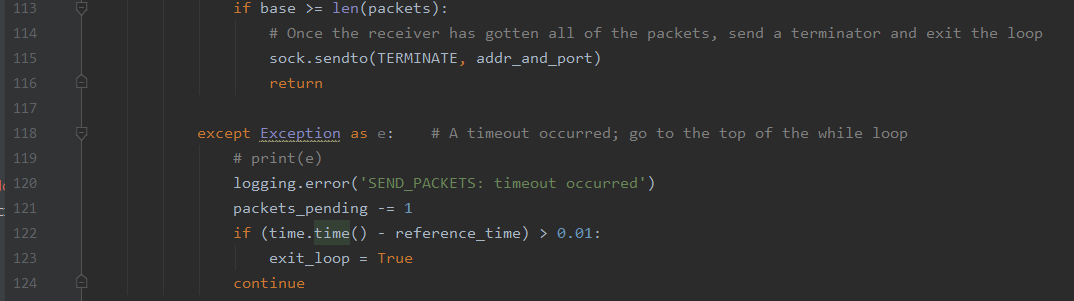
The make\_packet function takes in a byte array and breaks it up into packets of a certain size with the sequence number, checksum and data. After some initializations, and while data is still being read in, the packets are added to the total packet list. For the first packet, the total number of packets is just equal to the first packet. After that, the bytes are extracted into a packet. If there is an Index Error due to the remaining data being less than a packet, the loop is stopped by setting the data to an empty list. A checksum is created for the packet; and sequence number is made to track it. For a packet, it is equal to the sum of the raw packet, checksum and sequence number. Packet is then added to the entire packets. Finally, the total number of packets is increased for every iteration and that value is returned.



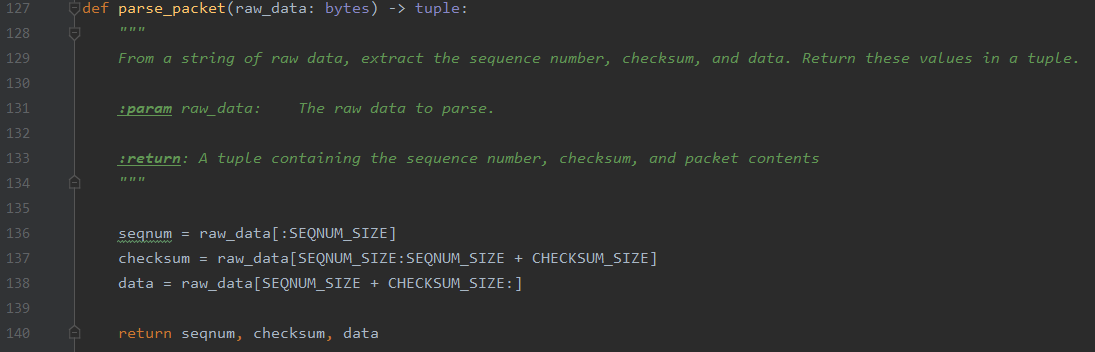
The send\_packets function which is used in Sender, will send a group of packets to the destination through a socket; parameters are the socket, packets and the address/port number of the destination. While there are packets to still be sent, window of packets are sent over to the receiver. If current sequence number is less than the number of packets. Line 76 sends the oacket to the receiver by identifying its address and port.



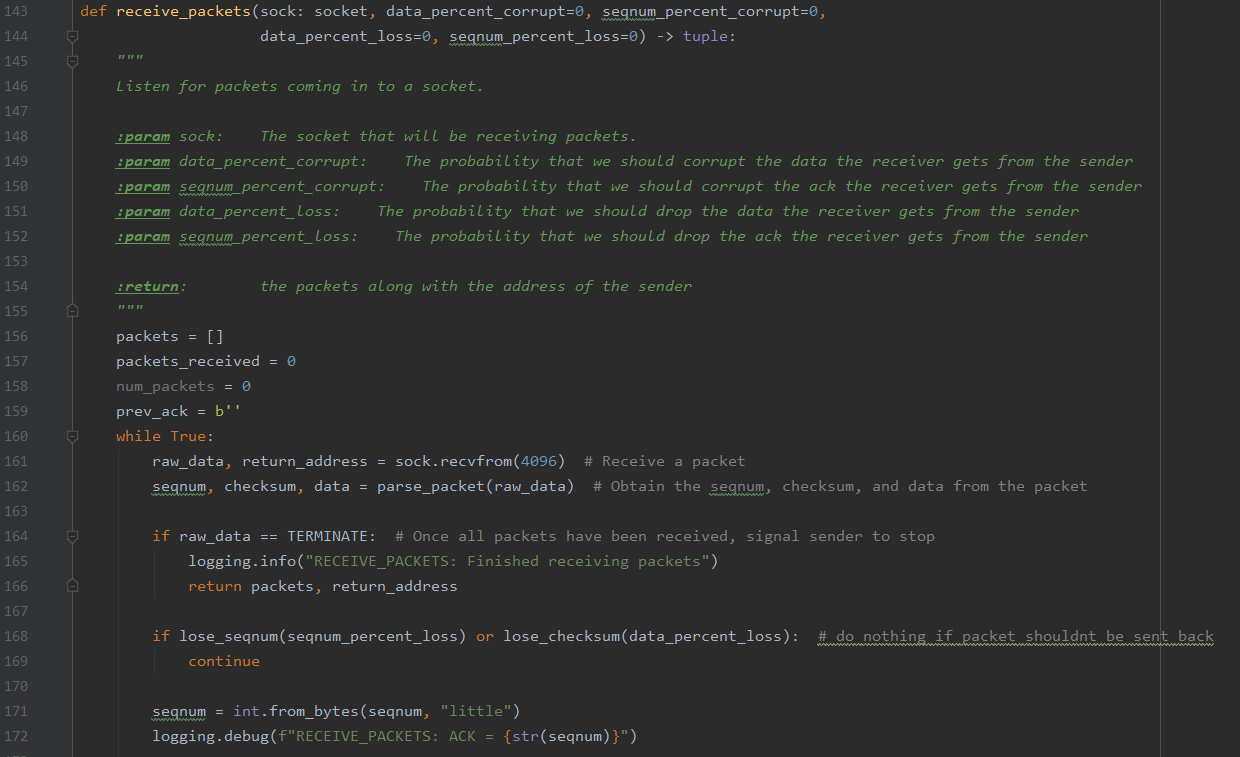
While having not exited, and packets pending to be send, an acknowledgment is received for the data. With a packet received, the window can be moved over and another window of packets can be prepared for sending beginning in line 97. If expected sequence number equals recovered sequence number, we proceed to following step. Line 103 says that if the packet number being worked on is less than the total number of packets then continue to send the window of packets over.



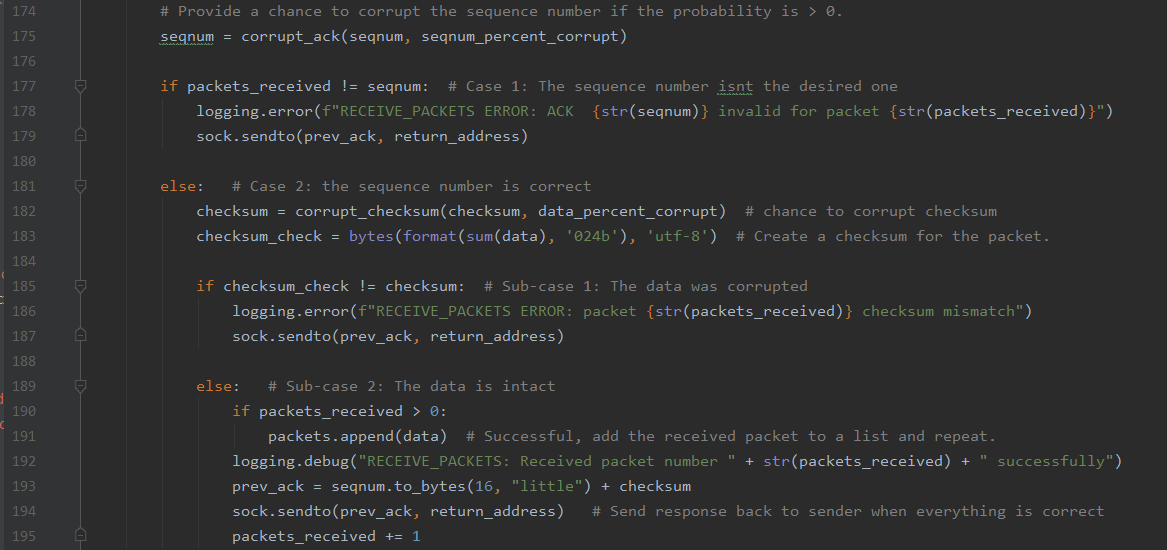
Once all the packets have been sent to the receiver, a terminator is sent and the loop is ended. In the event of a timeout in line 118, an error message is printed and pending packets total are subtracted by 1.



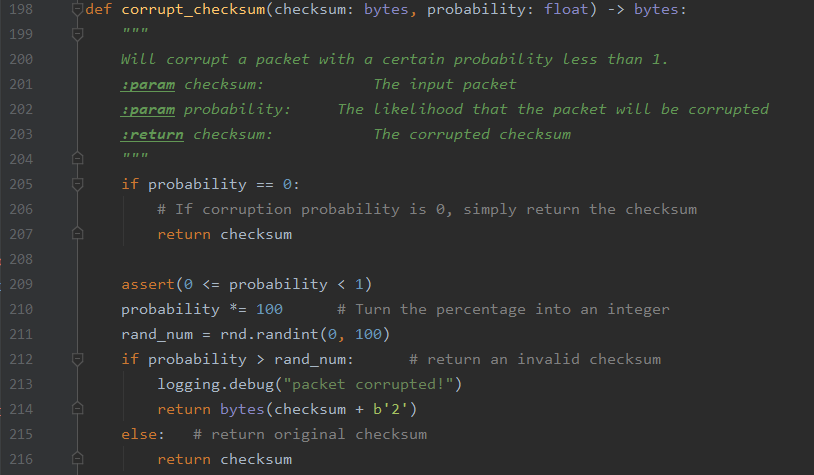
Next, the parse\_packet function extracts the data, checksum and sequence number from the data. Lines 136-138 do the extracting and return a “tuple” holding all three values.



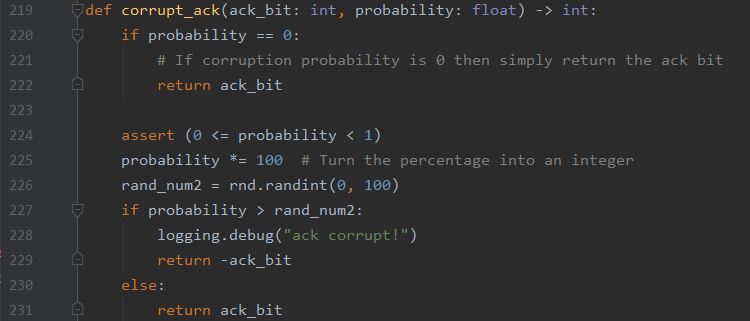
The receive\_packet function deals with the listening process for packets coming into the socket. Parameters are the socket, and the probabilities for certain corruptions or losses. While packets are received, the data, sequence number and checksum are taken from the packet. If all the packets are received, the sender is notified to cease sending packets.



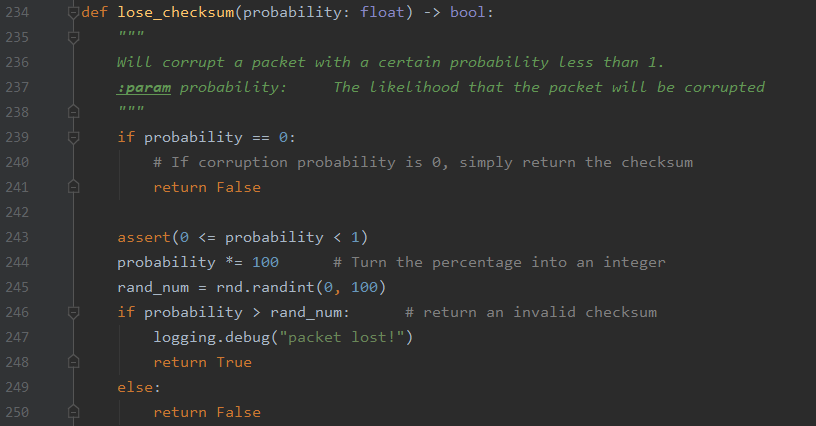
A chance to corrupt the sequence number is given if the probability is greater than zero. If the packets received are not equal to sequence number, error that a certain sequence number is invalid for a certain packet. Ack is sent through socket back to sender. If sequence number is good then corruption chance is provided; checksum created. In the event that the checksum isn’t qual to the checksum check then the data is corrupt and an ack is sent. If data is good and received packets are more than 1 then packet can be added to list and it may keep being done while there are still packets to be received. Response is sent to sender to notify of correct sending.



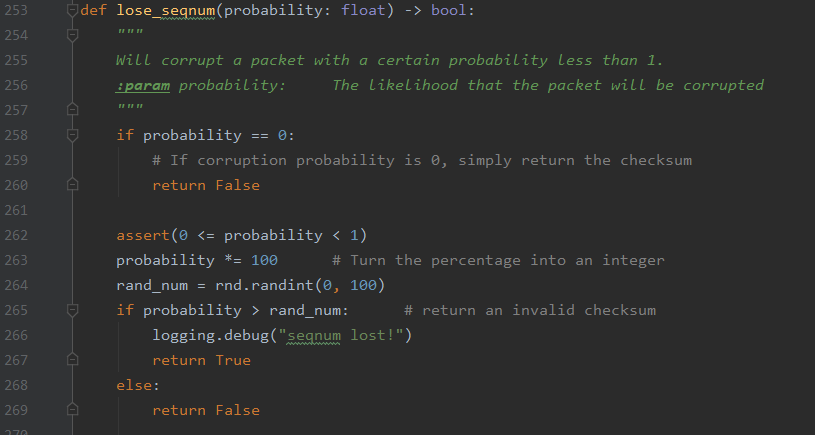
Function corrupt\_checksum will intentionally corrupt packet with a certain probability and reads in checksum, and probability of corruption. If probability is 0 then checksum is returned. Probability is generated between 0 and 1 and multiplied to become an integer. Line 211 generates a random number and if probability is greater than it, then return an invalid checksum indicating that the packet is corrupt.



For corrupt\_ack function, if the probability is 0 then ack is returned. When probability is greater than 0 and less than 1, the chosen probability is multiplied by 100 to become an integer. Line 226 generates a random number between 0 and 100 once more; if less than the probability than corrupt message sent.

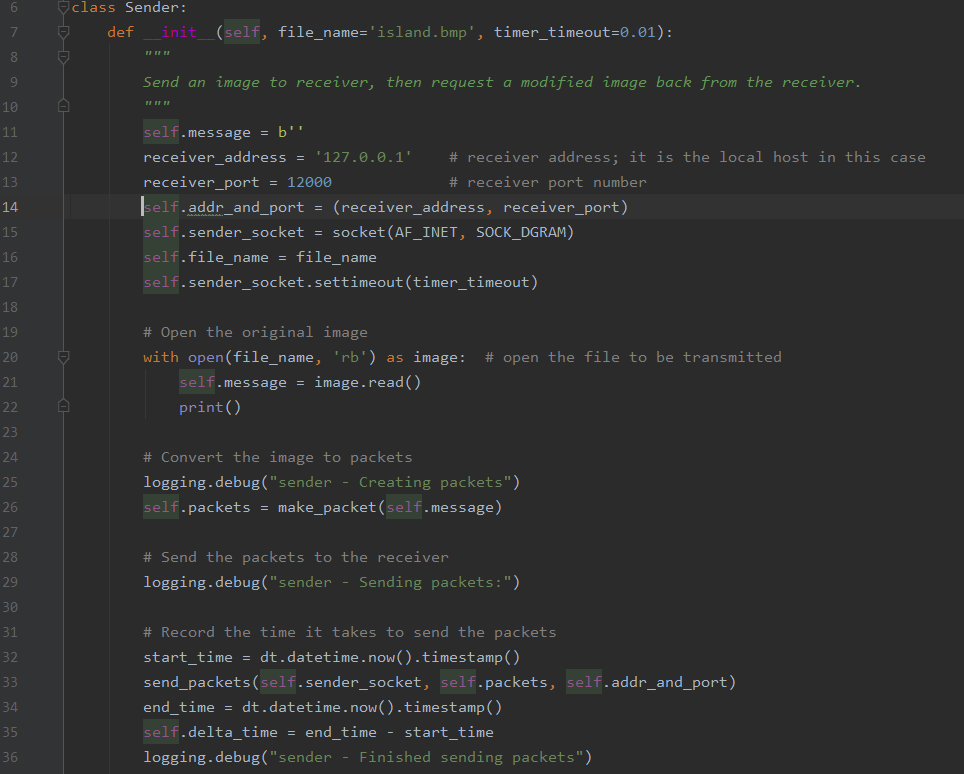


Lose\_checksum function corrupts packet with certain probability. If 0 then checksum is returned. Lines 243-247 perfrom same ideas as previous two function except will print packet lost error.

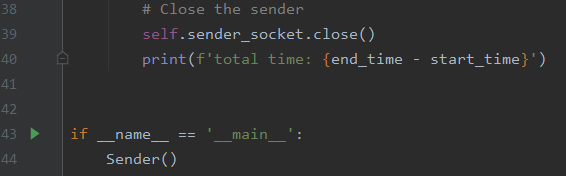


Lose\_seqnum performs similar idea as last function except will corrupt packet when sequence number lost.

**Sender.py**

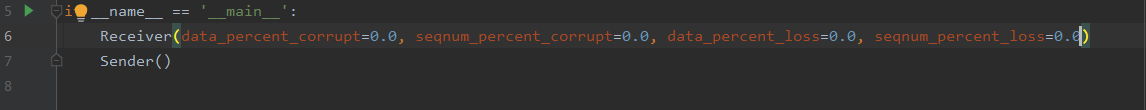


In the sender class, it begins with a definition that sends an image to the receiver then takes back in the image sent as the receiver sends it back. Island.bmp is the image name. Receiver address and port are set and a socket is aligned to prepare the sending. Origninal image is opened in line 20 and converted to packets in line 26. The time to send all the packets is recorded starting with start\_time. Once all packets are sent end\_time is set to the end time – start time which is 0.



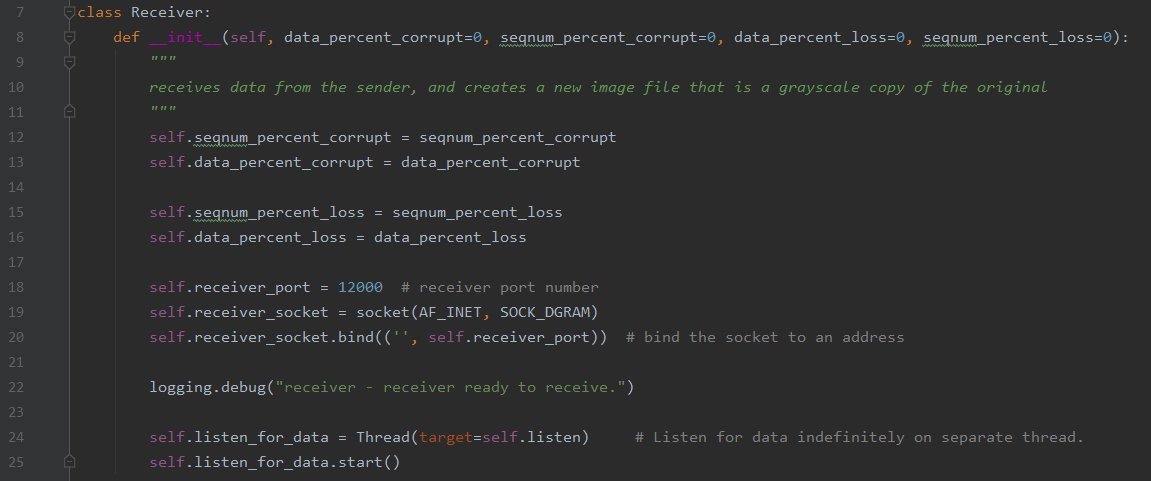
Sender is closed and time is printed out.

**Test.py**

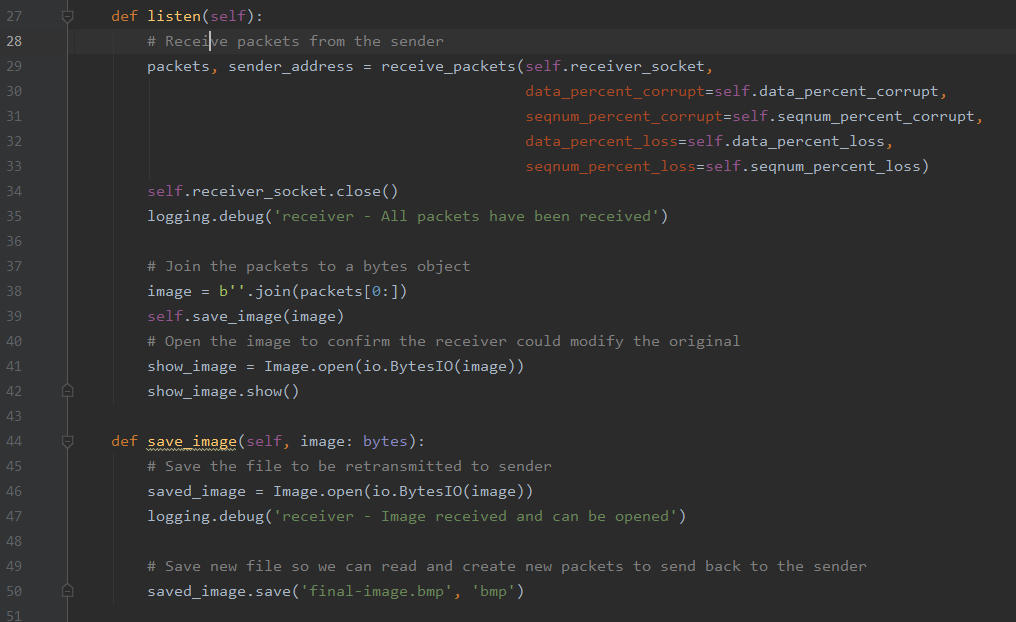


Used to test overall program. Various corrupts or losses can be tested at desired percentages.

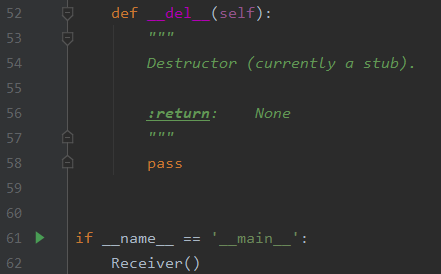
**Reciever.py**



Receiver class begins with def\_init\_() that receives in the data from the sender and makes a new image file. Its parameters are the percentage of corrupt data, sequence number percentage, and data and sequence number loss. All of them are initialized to zero. The port number of the receiver and sockets are set up prior to the binding of the socket to an address. The receiver continues to listen for incoming data separately and stores it in Thread().

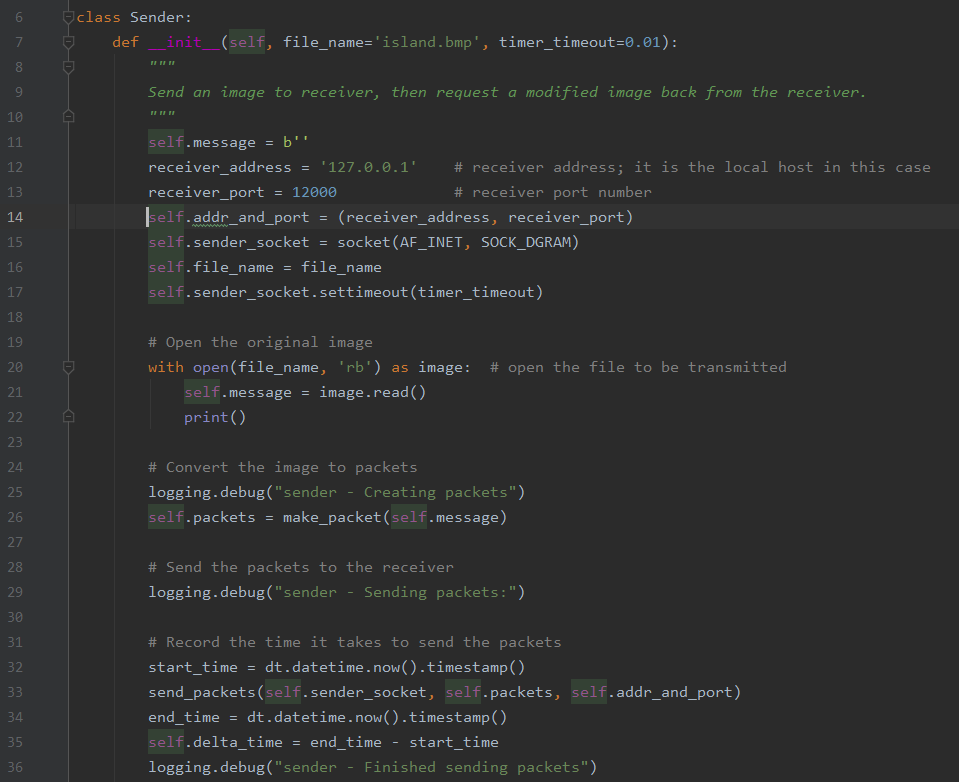


Listen function is defined to take in packets from the sender; these packets depend on the various corruptions and losses. Packets are combined to form bytes until end of array. Image is opened for verification. Line 44 defines a save\_image function to store the combined bytes (image).

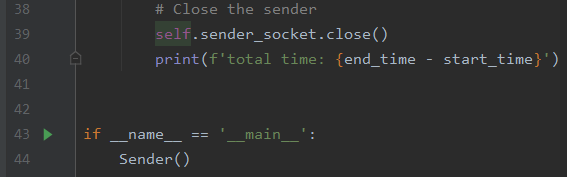


Destructor class and test.

**Sender.py**



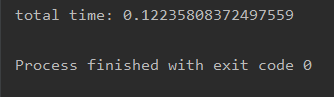
In the sender class, it begins with a definition that sends an image to the receiver then takes back in the image sent as the receiver sends it back. Island.bmp is the image name. Receiver address and port are set and a socket is aligned to prepare the sending. Original image is opened and then converted to packets. Line 28 begins sending the packets to the receiver. Timer begins in line 32 for the times to send all the packets. Once all are sent, the sender is closed and the time spent sending the packets is stored.



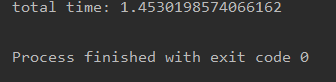
The total time is stated in line 40.

**Code Results:**

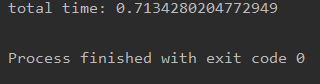
*No Corruptions/Losses*



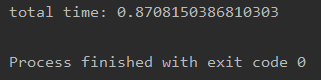
*Data Percent Corruption 0.1*



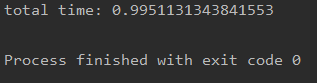
*Sequence Percent Corruption 0.1*



*Data Percent Loss 0.1*



*Sequence Percent Loss 0.1*



*Data Percent Corruption 0.1/Sequence Percent Corruption 0.1/Data Percent Loss 0.1/Sequence Percent Loss 0.1*

