log.py
A custom class Bin for handling binary files with pickle. This inits a step counter, log string, and file name. There are functions to write to the log with string concatenation, to dump the log with pickle, to load the log with pickle. There are magic methods for dunder call and str which will allow you to dump the log to the custom file and then load the log with pickle to print it to the terminal.
network.py
A custom class Network for supporting network attributes shared by client and ap. The class initializes shared attributes and provides getter and setter functions for them.
client.py
A subclass of Network called Client which provides attributes necessary for the client. The class provides getters and setters for client attributes. The client stores information about the access point it is connected to. Bin is imported from log.py to provide a local log.
ap.py
A subclass of Network called AccessPoint which provides attributes necessary for the access point. The class provides getters and setters for access point attributes. There are functions to determine limit, distance and rssi. Bin is imported from log.py to provide a local log.
ac.py
A custom class AccessController which provides functions to allocate channels to access points. Magic method dunder str is used to access the dunder str of the Bin object to dump and load the log content to print to the terminal.
main.py
The simulation occurs in main and takes file name input. It was hard coded to a local file during testing. We use a context manager to handle file reading. Each line goes through the parse_line function to create Class objects for AP or CLIENT or a tuple for a MOVE line. There are assert statements to ensure that lines follow the correct formatting and are valid. Several lists are made for the access_points, clients, moves, and simulation which contains the Client and Moves in order so that we can run the simulation. The function run_simulation takes the simulation list and list of access points to start the simulation. First, a Bin is initialized for all Clients and Access points using create_bin to store the log objects in the operations global dictionary with Client/AP: log. Then the simulation Access Controller is created as control. We call sort_access_points() from ac.py which allocates channels for each access point and logs (into control's log) the channel or channel change if the channel was occupied by overlap. Next we loop through the simulation list which has the order of clients and moves. We call parse_access_points which checks which frequencies are supported and creates a dict with tuple (ap, ap_frequency) where the ap frequency has been converted from 2.4 for example to 2400. Iterate_frequencies gives us only ap/frequency combinations that meet the rssi requirements. Then we call best_point which takes a list of ap/frequency and starts to filter the roaming process order. It will keep moving to next step from "When to Roam" until there is only one compatible ap/frequency. If it completes all checks and there are several ap/frequency to connect to, then it will use final_connect to connect to the available ap/frequency with the lowest number of connections. After finding the best ap to connect to, parse_access_points stores information about the match in the client object and we return to the run_simulation function. We log connections, moves, disconnects, in the client object, ap object and ac object. If the type of line in simulation is tuple then it is a move and we apply the move and check if the rssi dropped below requirement. We track all relevant information to the log. We can print the access control log by printing the object. We can access other logs from the dict but they can be printed by printing the object.log which will access dunder str from Bin.

Assume image simulation is ok and assert statements are fine. Assume functions tested iteratively.