My Anti-Chess code was somewhat simple, I avoided using complex tricks, and hacks, and didn't implement any strategy, but it was complex in terms of what the code is doing and as such I broke it down into several easy to understand modules. When the code is launched it determines the current player and launches directly into the Chess_Player class. .

The chess player class contains all of the functions that relate to the game and to the player itself; in it initialization it determines which player it is based off the passed in variable and loads the board, as well it also set's up all the other shared variables that are used in the other functions. The first thing to be run after initialization is the Play_game module; this module determines whether the current board is an end search board (either win/loss or depth reached) and if so return it's score; if not we get into the meat of the playing algorithm: we start looping through every one of our pieces and calling a module that calculates every possible move that the piece can do, these moves are broken into two lists: must_Takes and other_Moves. If there are any moves that take an opponent piece we loop through each of those moves and try to determine the best of those moves and if there are no must_Takes we determine which of the moves is the best of other_Moves. We loop through recursively using a minimax algorithm with alpha-beta pruning doing the same process until we reach either the winning/losing board or the goal depth.

The code is optimized in several ways to speed up processing (since it is a timed project) but the main two optimizations are using dictionaries and avoiding copying. Dictionaries in python are nice as they allow O(1) lookup and associating an index with a result; this plays into this assignment nicely as the O(1) lookup times cut down on processing time, and the indexing allows us to easily associate a piece with a location. The other big optimization is avoiding copying; when looping through and checking each possible move rather than making a copy of the opponent and player dictionaries we simply modify the one change and after we get the return change it back to what it was before; this allows us to avoid constantly reading and writing large chunks to memory.