After reading through the assignment one thing came to light, the patterns must take up the whole entire list, there cannot be strays and there cannot be half of the pattern remaining at the end. This gave me the mindset that these patterns are just a part of a whole and they cannot be more than half the length of the list because there will be fragments remaining. My approach checks for all possible patterns, starting at the longest pattern possible (half the list length) and down to 2. After checking the bases cases for whether the list is too short(a list cannot have a pattern if it is not at least length 4) or if the list is empty, My algorithm checks to see if the pattern length can fit inside of the list by using a modulus command. If the pattern length does not fit in the list then the pattern length is skipped over without checking. Once a pattern length does fit, the pattern is taken from the beginning of the list. For example with the list
[a, b, c, a, b, c, a, b, c] the pattern size 3 fits into the list so the pattern [a, b, c] will be taken.
In another loop this pattern is then compared with characters that appear every pattern length apart. So in our example the three characters including and following index 3 will be compared to the pattern then again with the characters including and following index 6. In my code this is done all in one line with a for loop inside of the all function. The loop produces indexes that are a pattern length apart, the value of this loop is used to create the pattern checks. If all of the patterns to be checked are equal to the pattern (True) then the all function will result in true giving a is_valid_pat variable the value true. If is_valid_pat is true then it passes an if gate allowing the pattern to be returned. If this fails, all pattern sizes will be tried. If all of the pattern sizes fail, then None is returned.

After understanding the task at hand I only had a few more intellectual challenges. The first came with one of my implementations that had my outside loop starting from 2, then going to my max pattern size, this resulted in failing tests of longer patterns. If the input was
[a, b, a, b, a, b, a, b] the algorithm would check for a pattern of length 2 first and it would pass having [a, b] repeated 4 times when to be correct it should be [a, b, a, b] repeated twice. I changed this by starting with the longest pattern possible and then decrementing down. This allows for the largest pattern to be checked first so if it passes it will be the returned pattern. When I did this, I accidentally made the loop not be able to reach 1 so I had some issues because of that. The next small issue I ran into came with an odd lengthed list, I initially found my largest pattern length by dividing the length by 2. I changed this by using integer division "\\" that divides and rounds down just like int typed division in another language. This allowed my function to no longer error out when checking odd lengthed lists.

My algorithm is of worst case big-O complexity n^2. This comes from the algorithm having two loops, an outside loop that provides the possible length of the pattern from biggest to smallest. And an inside loop that checks an identified pattern against other sublists in the list. Having a quadratic big-O feels like my code is slow, but every solution I think of requires going through the list twice which would require a nested loop. I am sure there is a solution out there that is much faster, but $O(n^2)$ is all I could come up with.