☆ DNA Analyser

Introduction
Write Python code to generate a report on the statistical occurrences of various amino acids in a given string.

This consists of 3 parts:

Part 1
It should read the DNA strands from a given list of strings in which every element represents a single strand.
The method should refine and update the list to contain valid strands *only*. A strand is valid when:
1. its length is greater than 10,
2. its length is less than 100,
3. it contains only `A`, `T`,`G` and `C` characters,
4. it's not an empty string,
5. all characters are in uppercase.

When the number of clean strands is less than 3, the method should stop its execution and return an empty string.

Part 2
Using the refined list, now you should find overlapping elements and build one long strand which should be returned as a string.
Two strands overlap when last 3 characters of the first one match exactly 3 first characters of the second strand.
The following rules should be used for building the resulting string.
1. All input strands must overlap.
2. When two strands overlap they should be merged, e.g. `AAACCCAATTT` and `TTTACACAGCT` should be merged to `AAACCCAATTTACACAGCT` - the overlapping part should not be repeated.
3. One strand must overlap with another one only by the end of it - this is the starting strand.
4. One strand must overlap with another one only by the beginning of it - this is the ending strand.
5. All other strands must overlap on both ends.

Please note, the starting and ending strands can be anywhere in the received list.

Example
For the following input list
```
[AGTGGGGGGGGG, AAACCCAATTT, TTTACACAGCT, GCTGGGCCCAGT]
```
the expected outcome is

Example
For the following input list
```
[AGTGGGGGGGGG, AAACCCAATTT, TTTACACAGCT, GCTGGGCCCAGT]
```
the expected outcome is
```
AAACCCAATTTACACAGCTGGGCCCAGTGGGGGGGGG
```
where the strands would be matched and merged as shown below.
```
AAACCCAA[TTT] -> [TTT]ACACA[GCT] -> [GCT]GGGCCC[AGT] -> [AGT]GGGGGGGGG
```

Part 3
The last part of this task is to actually generate the report.
Now the constructed string should be split into 3-characters long substrings. Each substring is a codon specifying an amino acid. Based on the populated dictionary - "codon_mapping" (second argument to the function), count amino acids occurrences in the given sequence.
The method should finally return a string consisting of alphabetically sorted amino acids and their count (colon-separated, one per line as depicted below).

Example
For sequence `AAATTTGGGAAA` and `codon_mapping` dictionary with following values:
```
AAA    Lysine
GGG    Glycine
TTT    Phenylalanine
```
the expected outcome is
```
Glycine: 1
Lysine: 2
Phenylalanine: 1
```

Also handle negative scenarios wherever necessary/applicable and return an empty string.