Summary

The first step was to understand the problem statement itself. Handling a FASTA file with DNA sequences wasn't something new, but ChIP-Seq technology was unknown to me. What I learned:

1. Transcription Factors bind to sites in DNA, these sites are hence called Transcription Factor Binding Sites (TFBS).
2. We are interested in spotting these sites because this is where gene expression is regulated, and it can hence be a potential drug target.
3. TFBS form motif and they are in a certain pattern, also known as consensus.
4. The nucleotides are represented as per the IUPAC notation.

Part 1:

The goal is to write a python script that can extract user specified consensus from DNA sequences in a FASTA file (forward and reverse complement).

The first decision to be made was to choose the number of functions required to make the script user friendly and error resistant. Referring to the solution provided for the ORF finding problem by the Professor, I decided to define 2 functions.

1. find_tfbs: This function finds all the Transcription Factor Binding Sites(tfbs) matching the user entered consensus in each DNA sequence as per IUPAC notations.
2. script_op: This function reads a FASTA file line by line to retrieve sequences, calls find_tfbs function and carries out tfbs search both in forward and reverse complementary DNA sequences and returns a .txt file with tab separated input sequence, sequence ID, Direction and consensus conforming tfbs.

In the process of developing the logic, initially I considered writing a simple logic to find a short sub-string in given long stretch of string where I manually entered short nucleotide sequence and direct consensus (only ATGC). After successfully achieving this, I included other notations.

The major difficulty faced was to understand how to compare the consensus when it has multiple possibilities, I tried getting each character from the consensus and matching it with the list of values defined in the notation dictionary with for loop. The code would run but I'd end up getting empty list. This happened because I compared the consensus directly with substring in the loop (string == string) but not their characters, which would always return TRUE.

To get this right I tried accessing string index in a 'for' loop, however the problem persisted. I had to google if there were any built-in function in python that would help me get the logic right. Simple option was to use TRUE – FALSE logic along with the 'for' loop. Initialising True conditions coupled with looping to match characters helped me build the correct logic (Few Rosalind exercises were helpful in getting the idea).

The second part of the code was easy to build because of the usage of *Biopython* features. Parse(), .id, .seq and reverse_complement() were handy. The only challenge was to structure the output neatly so that it could be easily read to address the part 2 questions.

Making it Error free:

- The code is made to raise a ValueError if invalid characters or symbols are entered as consensus.
- The code raises IndexError if any of the arguments(python file, fasta file or the consensus) is missing while execution.

Making it user-friendly:

- Comments on top of the code describes briefly the function of the code and how to run in the terminal with example along with other comments for the major steps.
- User can easily enter the consensus in upper or lower case.
- Additionally, documentation is provided for the functions.
- Help on the functions can be accessed in the terminal as: pydoc part1.find_tfbs and pydoc part1.script_op

Test: Codes are tested for correct operation by verifying the manually obtained result (refer illustration below in the box) with the result obtained from the code. test_find_tfbs function checks if expected result matches the obtained output. test_tsv checks if the part 1 output file has tab separated values.

notation = {'A': ['A'], 'C': ['C'], 'G': ['G'], 'T': ['T'], 'U': ['U'],

'R': ['A','G'], 'Y': ['C','T'], 'S': ['G','C'], 'W': ['A','T'], 'K': ['G','T'], 'M': ['A','C'],

'B': ['C', 'G', 'T'], 'D' : ['A', 'G', 'T'], 'H' : ['A', 'C', 'T'], 'V' : ['A', 'C', 'G'],

'N': ['A', 'C', 'G', 'T'], '.': ['gap'], '-': ['gap']}

sequence = 'GGGAGTTTCCGGGAATTCCC'

consensus = 'GGGRNWYY'

As per the notation,

GGGRNWYY

'GGGAGTTTCCGGGAATTCCC' where R=A, N=G, W=T & Y=T

and

GGGRNWYY

' GGGAGTTTCCGGGAATTCCC' where R=A, N=A, W=T, Y=T & Y=C

expect = ['GGGAGTTT', 'GGGAATTC']

test_script_op is given with a test_file(.txt) with known output which is compared with output file(.txt) generated from script_op. These 2 tests prove that find_tfbs can perform correct consensus search and script_op can successfully generate a .txt file with output.

User can further assess the functionality of the code by trying inputs of various lengths and notations including '.' and '-'.

Part 2:

A) Finding top 10 most common consensus-conforming sequences found among all input sequences was not complicated because the output file was well structured and had all the sequence listed. The tricky part was to read only the sequence from the file - since I structured the output file to have exact 4 repetitive lines with seq id, direction, sequence and a blank line, I was aware that every 3rd line needs to be accessed and added to the list.
Since python has a *Counter* object from *collection* module, I could use .most_common(top_n) method of the Counter to get top 10 occurrences.
Result is represented as a dictionary with sequence as the key and it's count as the value.

User can further assess the functionality of the code by trying inputs of various lengths (example: top_n = 5).

B) Writing a script to find the number of sequences that did not contain any sequences conforming with the user-specified consensus was easy because I had included sequence ID in the output file.
The logic was to retrieve all the sequence IDs from the input file into a list and retrieve all the sequence IDs from the output file and add to another list. The next step was to compare what is present in input file that was absent in the output file. The output that I obtained was bizarre.

To cross-check, I created a 2 test files, test_part2_input.fasta which had 3 unique input sequence IDs and test_part2_output.txt file which had 3 consensus (1 with unique sequence ID and 2 with same sequence ID) . And the output was 3.
The error was that I had not thought of duplicates. I had to google the functions that could be useful in removing duplicates from the list. The most useful function here was the set(). This helped me remove duplicates and create unique list. Following this I could easily take the difference out of the number of sequence IDs in input file from that of the output file after removal of duplicate IDs to get number of sequences that did not have any sequences conforming with the user-specified consensus.
However, since the input file has numerous sequences, there could be chances of repetition in the DNA sequences, hence set() is used to remove duplicates from the input file as well (if any).

Making it user-friendly:

- Comments on top of the code describes briefly the function of the code and how to run in the terminal with example along with other comments for the major steps.
- Additionally, documentation is provided for the functions.
- Help on functions can be accessed in the notebook as: help(common_seq) and help(no_seq)
- Some plots are given for easy visualization of the results.

Test: Codes are tested for correct operation by verifying the manually obtained result with the result obtained from the code. test_common_seq and test_no_seq function checks if expected result matches the obtained output.

As mentioned above both the tests are provided with 2 files and expected output successfully matched with that of actual output.