Fundamentals Problem Set #2

Covid-19 Computational Biology Workshop

April 19, 2020

1 Motif analysis

The binding sites preferred by DNA- and RNA-binding proteins often include degenerate bases, which are represented as `N`, `R`, `Y`, etc. by IUPAC nomenclature standards. For example, the MBNL1 alternative splicing regulator binds RNA at a canonical motif of `YGCY`, where `Y` can be any pyrimidine (`C` or `U`). One key problem in sequence analysis is identifying the locations of these binding site motifs.

Consider a hypothetical intronic sequence: `GUGCUAGCUGCGGUGCAGCAACAAAUAGAGGCUGCCUAUU` . Define a variable `intron` that contains this sequence as a string. Define another variable called `motif` that contains the the canonical MBNL1 binding site `YGCY` as a string.

1. Generate a list called `all_motifs` that contains all possible RNA sequences that match `YGCY` (eg. `UGCU`, `CGCU`, ...). (With a motif this short, you can easily do all of this by hand. However, in other cases, you may have a degenerate sequence long enough that you would need to write a function to generate this list. Try that out if you're bored!)
2. Find the positions of all occurrences of `YGCY` by iterating through the sequence `intron` and testing each 4-mer subsequence to check if it is present in `all_motifs`. Organize the positions of all found `YGCY` motifs into a list. Print the list and its length.

2 In-place methods

Sorting a list alphabetically can be accomplished in multiple ways. Create a list called `sequences` containing the following nucleotide motif sequences as strings:

- `TGCC`
- `AATG`
- `AGCT`
- `CGTA`

1. Sort this list in-place, using the `.sort()` method. How does the order of `sequences` change?

2. What happens if you try to set a new list `new_list` equal to `sequences.sort()`? If you print `new_list`, is it what you expect? Why is `new_list` not a sorted version of `sequences`?

3. Define `sequences` again with the above order. Set `new_list` equal to `sorted(sequences)`. What happens to the order of `new_list` and `sequences`? Why is this different?

3 Iterators

In Python, `for <item> in <list>` is a great way to iterate over all the items in a list in sequence. However, sometimes you may want to iterate over multiple lists at the same time. For example, consider a paired-end sequencing experiment, where paired reads are stored in order in two separate FASTQ files (eg. R1 and R2). When writing a paired-end sequencing pipeline, it's often helpful to use an iterator that allows you to get both reads together from each pair.

Create the following two lists:

- `reads_R1`: `["CAGAGT", "CTAACA", "GTTTAT", "GAGCTG", "GGGGCC"]`
- `reads_R2`: `["ATCACC", "CTTAAC", "AGTCAC", "TCGGCT", "TTGTGG"]`

1. Write a function called `paired_GC` that takes in two sequences as arguments and returns the total GC content of both reads together. For example, if we call this function using the first pair from the above lists:

   ```
   gc_content = paired_GC(reads_R1[0], reads_R2[0]) # this should be 0.5
   ```

2. `range()` is a useful iterator for *indexing lists* during a loop. Using a `for` loop with the `range()` iterator, calculate the GC content of each pair of reads in the above lists using your `paired_GC` function, and print the result.

3. `enumerate()` is an iterator that gives you both the index of an item in a list *and* the item itself for every iteration. Perform the same task in part 2 using a `for` loop with the `enumerate()` iterator. (Hint: try `for i, read1 in enumerate(reads_R1)` and print `i` and `read1` to see how this works.)

4. `zip()` is an iterator that takes in multiple lists and gives you the *items of both lists together* for every iteration. Perform the same task in part 2 using a `for` loop with the `zip()` iterator. (Hint: try `for read1, read2 in zip(reads_R1, reads_R2)` and print `read1` and `read2` to see how this works.)

(In reality, sequencing runs often contain millions of reads, and loading them into lists in their entirety may not be possible due to memory constraints. Later on, we'll go over ways to parse large sequencing datasets in Python without loading them into memory all at once.)