Problem Set 3 – Mapping and Loops

1. **Explore implicit mapping.** Consider the BioBIKE functions below and by actual test, determine whether the function is capable of implicit mapping. The test should be conducted as follows:

    a) Execute the function with a simple value or values, e.g. a single number, string, gene... whatever is appropriate..

    b) Replace the simple value with a list of values of the same sort. For example, replace a number, e.g. 2, with a list of numbers, e.g. (2 4 6).

    c) Before you execute the function with a list of values, predict what you would expect to get as the result if the function is capable of implicit mapping.

    d) If you didn't get the result you expected, consider whether there's a reason why not or whether BioBIKE is behaving unreasonably.

    1a. LOG10
    1b. CODONS-OF
    1c. DISPLAY-LINE
    1d. LENGTH-OF
    1e. SUM-OF

In many cases, the easiest way to write a loop or to map a function is to first teach yourself how to do the desired operation a single time. Once you've figured out a series of commands that does the job, you may then be able to cut and paste the commands into the body of the loop or mapping operation. All that remains is to specify how to iterate the appropriate number of times and (in the case of loops) how to collect the individual answers into a result. We'll call this "looping by example". You can do this with Questions 2 through 4 below.

2. Write a loop that says hello 10 times and get the same result with a mapping operation. *(Write a command that displays "hello", then cut and paste it into the body of FOR-EACH or REPEAT-FUNCTION. Consider using a member of the DISPLAY family of functions).*

3. Display the name of each gene of Anabaena variabilis (nicknamed Avar) followed by the 15 nucleotides preceding the gene ending with the start codon and then the 15 nucleotides following the gene starting with the stop codon. *(Do this with a single gene, say Ava0001, by constructing a list, using the LIST function, consisting of the gene, SEQUENCE-OF to get the first sequence fragment, and SEQUENCE-OF again to get the second sequence fragment. Then drag the function into APPLY-FUNCTION, replacing the specific gene with all the genes of Avar).*

4. Write a loop that displays a table of squares (1^2, 2^2, 3^2, …). *(Write a command that displays a number, then tab, then the numbers' square. Then cut and paste it into the body of FOR-EACH or APPLY-FUNCTION. You'll want to know about DISPLAY-LINE, PRODUCT-OF, and *tab*, which can be obtained from the Data menu. You can also try getting the same thing using APPLY-FUNCTION instead of a loop.)*

Some loops cannot be constructed by example, because the loop does more than just iterate over a single action.

5. Write a loop that sums the even numbers from 2 to 100.

6. Write a loop that finds the size of the largest genome known to CyanoBIKE. Note that by mousing over the DATA button and then the **Organism Subsets** item, you get to **All Cyanobacteria**, which is a list of exactly what it says it is.

7. Devise an algorithm to provide the ***name*** of the cyanobacterium that has the smallest genome known to CyanoBIKE, then write a loop to do this. Provide the algorithm in English pseudocode (one unit operation per phrase), and then provide the actual loop. You can also try to get the same result using the functions SORT and FIRST.

8. Genomes are not random (if they were, we wouldn't be having this conversation). Write a loop that counts the actual occurrences of all 6-nucleotide sequences in the genome of the cyanobacterium *Anabaena* PCC 7120 (A7120) ***composed solely of C and G*** (confined to this subset to make the time of execution short enough to be toleraable). Then sort the list of counts so you can try to make sense out of them. Here's how:

   a. Figure out how to get the COUNT-OF a specific hexanucleotide sequence in A7120 (you've done something like this before in *What is a Gene?*).

   b. Put this function in the body of a loop, and iterate over all possible hexanucleotides consisting solely of C's and G's. The function ALL-STRINGS (STRING-SEQUENCE menu, String-Production submenu) will be helpful. COLLECT a LIST, consisting of the sequence and the counts. Or use APPLY-FUNCTION instead of a loop, if you like.

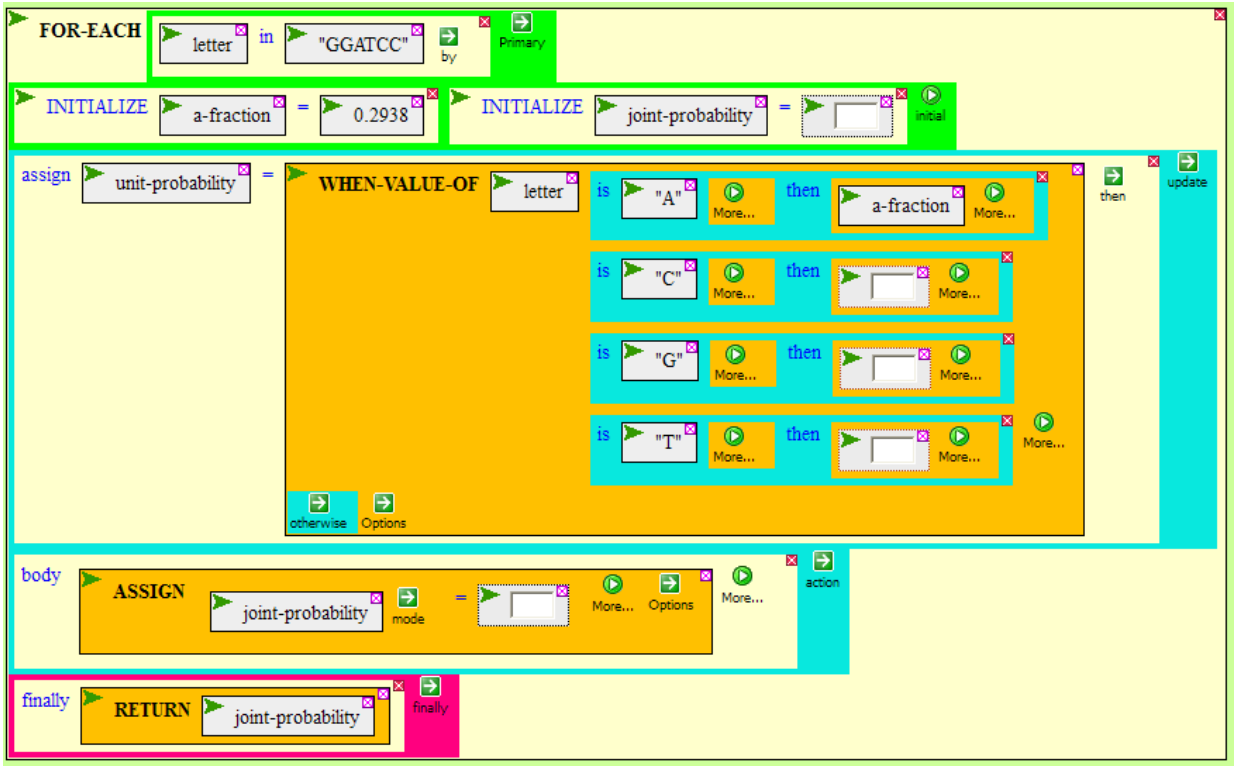
   c. DEFINE the resulting list as a variable named however you wish. The PREVIOUS-RESULT function may be convenient. You can find it on the OTHER-FUNCTIONS menu. Or you can more simply drag the result into the *value* box of DEFINE.

   d. SORT the list, making use of the SORT function found in the LIST-TABLES menu, List-Production submenu. Use the BY-POSITION option to specify that you're sorting by the second element of the list, i.e., the number of counts. SORT it both ASCENDING and DESCENDING, in order to see the sequences that are most common and those that are least common.

   e. What generality can you discern regarding the least common hexanucleotides? Why do you suppose they are least common?

9. Write a loop that calculates the probability of encountering a given nucleotide sequence (e.g. "CGCGCG") in a genome with [A] = 0.2938 (the actual frequency of A in *Anabaena* PCC 7120). The partially completed loop below may be helpful. Use the function to compare the counts you got in problem 3 to what you would predict by chance. See suggested template on the next page.

(*Suggestion for Problem 9*)



10. Devise an algorithm that takes a DNA sequence of arbitrary length (5' to 3'), DEFINEd as `seq`, and produces the complementary sequence (5' to 3'). Yes, I know this is the same question as in Problem Set 1. This time, write a loop that implements it, using (if you like) the template below. You will probably want to play with SUBSTRING-OF before using it. Try using a known string (like "ABCD" and insert different values into the FROM and LENGTH options.