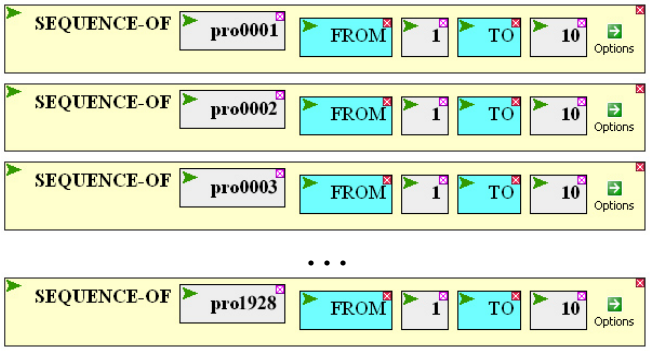
BioBIKE Language Syntax

Working with large numbers of items: Mapping and Loops

Bioinformatics, by definition (to the extent it has a definition), means considering large data sets: large sequences, large numbers of genes, large sets of measurements. You already confronted the problem raised by large data sets in the first guided tour, *What is a gene?* in which you wanted to find out what sequence begins genes. It wouldn't have done any good, of course, to examine just one gene. You could have examined many, one at a time…



… but that wouldn't have been practical at all! How can you get the sequences from all 1900+ genes without killing yourself?

BioBIKE language (BBL) allows you to choose between two approaches to handle repetitive operations: mapping and loops. Mapping is generally simpler, so simple that you may use it without realizing it, because it happens implicitly, behind the scenes. It is also possible to invoke mapping explicitly. Loops are more general, helpful when direct mapping of a process is not available.

I. Mapping

You're starting off in a new job working in a library. Your new supervisor tells you the first thing to do is to put magnetic strips on all the new books that have come in. Here's how… and she shows you how to put a magnetic strip in a book. Now do that for this stack of books.

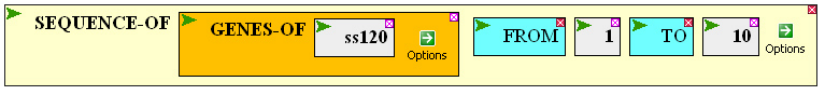
If you've done something like this, then you know how to ***map*** a function, that is, to apply the function defined for one object to a list of objects. You also know how to map a function in BioBIKE, at least by implicit mapping.

I.A. Implicit mapping

You've already had experience with mapping within BioBIKE. Instead of typing out 1900 lines of code to extract sequences from 1900 genes, you applied the `SEQUENCE-OF` function to all genes at once:

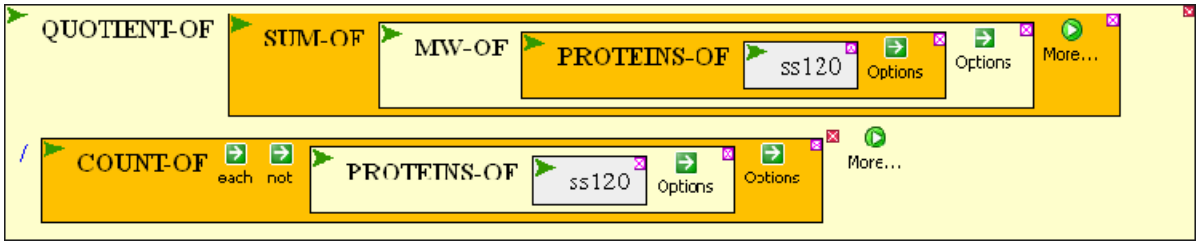
(the function applied to one object)

(the function applied to a list of objects)



What you did was to ***map*** the function `SEQUENCE-OF` over the set of ss120 genes. Mapping always gives a list of results – one item for every item in the list over which the function is mapped. Because BioBIKE was designed with bioinformatics in mind, it is generally possible to map BioBIKE functions over sets in this way, whenever mapping makes sense.

Here's another example. Suppose you want to find out what's the average size of a protein in *Prochlorococcus marinus* ss120. This requires that you calculate the molecular weight of each protein, sum the molecular weights, and divide the sum by the total number of proteins. With mapping, the strategy can be translated into BBL code directly:



Note that MW-OF is mapped over the set of proteins of ss120. It produces a list of results: one molecular weight for every protein in ss120. Note that SUM-OF is ***not*** mapped over the set of molecular weights, and COUNT-OF is ***not*** mapped over the set of proteins. Both produce single numbers, not a list. The first function produces a sum of a list of numbers and the second produces a count of the number of elements of a list. In neither case is the function acting on ***each*** element of the list. In these cases, there was no ambiguity: (1) Individual proteins have molecular weights, so you can map MW-OF over a list of proteins; (2) Individual proteins don't have counts, so you can't map COUNT-OF over a list of proteins.

Some cases might strike you as ambiguous (Fig. 1A, below). Perhaps `list-of-things` consists of `{"GGG" "ATG" "AGA" TGA"...}` and I want to count how many `"ATG"` elements are in the list. In this case, COUNT-OF should not map itself over the list. It should just count, producing one number representing the number of "ATG"s. Alternatively, `list-of-things` might consist of a list of genes, in which case COUNT-OF ***should*** map itself over each item in the list, producing a list of counts, one for each gene. Since machine intelligence is at present unable to read your mind, how can you tell the computer whether or not to map?

BBL provides a few ways to disambiguate a command. Compare Fig1A to Fig 1C. In the first case, the fact that COUNT-OF is singular, indicates that a single result is desired, while in the second, the plural COUNTS-OF indicates that a list of results is desired. So COUNTS-OF but not COUNT-OF … IN … calls for mapping over `list-of-things`. Another way to specify mapping in potentially ambiguous situations is to use IN-EACH (Fig. 1B), which specifies that COUNT-OF is to be applied to each element of the list. Similarly, EACH preceding the list of nucleotides in Fig. 1D indicates that you wish a count of each individual nucleotide, not the count of the list {"A" "C" "G" "T"} in `list-of-things`.

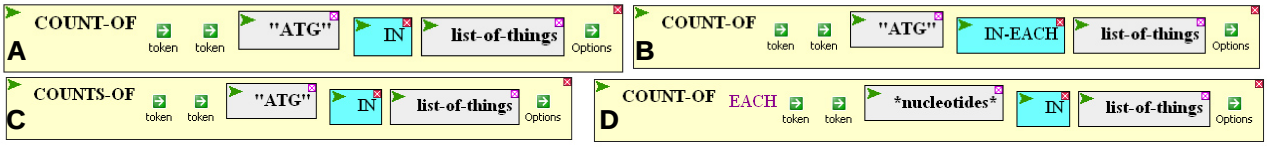


Fig. 1: Comparision of similar instances of COUNT-OF and COUNTS-OF

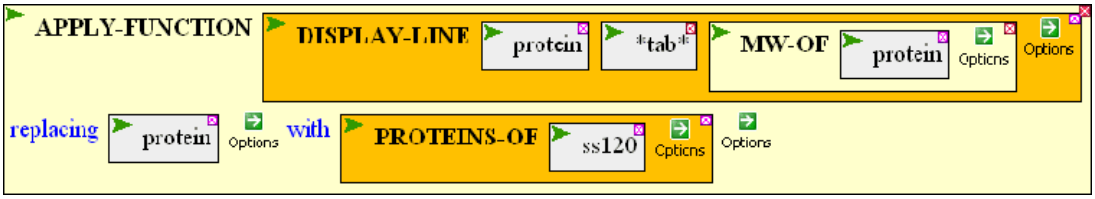
SQ1. Find three different functions able to implicitly map. Find three that cannot implicitly map.

SQ2. Find the lengths of every gene in ss120, using both LENGTH-OF and LENGTHS-OF.

I.B. Explicit mapping

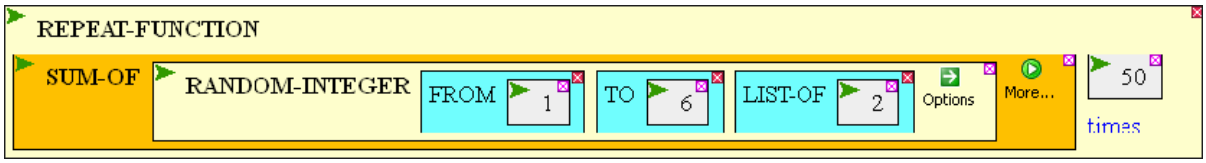
Inevitably the time comes when you want to map a function that, unfortunately, doesn't exist. What then? For example, suppose you want to list all proteins in a plasmid along with their molecular weights. You will not find LIST-PROTEIN-WITH-MOLECULAR-WEIGHT in any menu. However, so long as you can teach BioBIKE how to deal with a single item, you can map those instructions over a list. Here's the single case:

So long as you define `protein` to contain the name of a protein, this command works fine… for one protein. Having figured out how to do this, you can apply the instructions -- you can map the command – over a list.



(Mapping and loop functions may be found in the FLOW-LOGIC menu) The first argument tells BBL which is the variable component of the command, i.e. which component is to be replaced by each element of the list. As with implicit mapping, mapping by APPLY-FUNCTION always returns a list of results.

Explicit mapping can also be used to repeat the execution of a function any number of times, using REPEAT-FUNCTION. Suppose you want to generate 50 dice rolls (two dice at a time). You can do so as follows:



Mapping of this sort is very useful in doing simulations, examples of which you will see in a problem set.

SQ3. Use explicit mapping to display a listing of the squares of numbers from 1 to 10 (the FROM function will also be useful).