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…

```
(SEQUENCE-OF pmm0001 FROM 1 TO 10)
(SEQUENCE-OF pmm0002 FROM 1 TO 10)
(SEQUENCE-OF pmm0003 FROM 1 TO 10)
...
(SEQUENCE-OF pmm1655 FROM 1 TO 10)
```

… 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. 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'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:

```
(SEQUENCE-OF pro0029 FROM 1 TO 10)
```

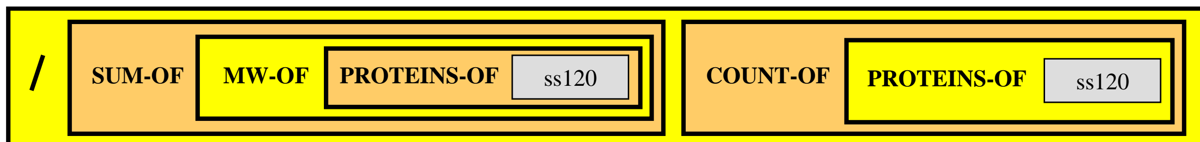
(the function applied to one object)

```
(SEQUENCE-OF (GENES-OF ss120) FROM 1 TO 10)
```

(the function applied to a list of objects). What you did was to ***map*** the function `SEQUENCE-OF` over the set of ss120 genes.

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* 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:



Or in text form:

```
(/ (SUM-OF (MW-OF (PROTEINS-OF ss120)))
   (COUNT-OF (PROTEINS-OF ss120)))
```

Recall that division, like all BioBIKE functions, begins with the name of the function, followed by the items the function acts on. Of course, you could do this in multiple steps, perhaps more intelligibly, executing each line one at a time:

```
(DEFINE protein-MWs AS (MW-OF (PROTEINS-OF ss120)))
(DEFINE total-MW AS (SUM-OF protein-MWs))
(DEFINE protein-count AS (COUNT-OF (PROTEINS-OF ss120))
(/ total-MW protein-count)
```

When mapping works, it often makes a process simple to write and simple to understand. Some processes, however, cannot be mapped. Then loops can save the day.

II. Loops

II.A. Overview of loops by example

Mapping is simple: just replace a single item with a set of items. In contrast, looping uses what seems like a separate language. A ***loop*** executes one set of instructions repeatedly. Each time through the instructions is called an ***iteration***. Here's the previous example rendered as a loop:

```
(FOR-EACH protein IN (PROTEINS-OF ss120)
     SUM (MW-OF protein))
```

Translation:

a. *Consider each protein in the set of all proteins of ss120, one at a time.*
b. *Accumulate the molecular weights of each protein.*
c. *When the last protein has been considered, return the sum*

That gets you the total molecular weight. Or this code gets you the entire answer by means of a more complicated loop:

```
(FOR-EACH protein IN (PROTEINS-OF ss120)
     INITIALIZE total-MW = 0
     INITIALIZE protein-count = (COUNT-OF (PROTEINS-OF ss120))
     AS mw = (MW-OF protein)
     (INCREMENT total-MW BY mw)
     FINALLY (RETURN (/ total-MW protein-count)))
```

Translation:

d. *Consider each protein in the set of all proteins of ss120, one at a time.*
e. *Before the loop begins, set the sum of molecular weights to zero. The initialization occurs only once.*
f. *Before the loop begins, set the number of proteins. This will be a constant.*
g. *Find the molecular weight of the one protein you're considering at the moment. This assignment is repeated each time through the loop.*
h. *Add that molecular weight to the growing total.*
i. *(Loop) Repeat steps d and e until you've considered each protein in the set.*
j. *When you've finished considering each protein, calculate the average molecular weight and use that as the value returned by the `FOR-EACH` function.*

Those of you who are familiar with loops from other computer languages may have been expecting something more along the lines of:

```
(FOR-EACH n FROM 1 TO 10
     (DISPLAY-LINE n *tab* (* n n)))
```

Translation:

a. Consider each number n from 1 to 10, one at a time.

b. Display the number you're considering at the moment as well as its square.

c. (Loop) Repeat step b with a new n each time until n reaches 10.

BBL can do this kind of loop, but they're not common in bioinformatics applications. It's more common to go through lists of things, like genes or organisms, as in the first example.

II.B. Anatomy of the loop

Loops can be divided into the following (mostly optional) parts:

Iteration control: Determines how the loop begins and ends and sets up the iteration variable

Loop-specific initialization: Set variables before the loop begins, to be available throughout the lifetime of the loop

Iteration-specific assignment: Set temporary variables used within one iteration

Body: Instructions to be executed each iteration

Return value: Set or accumulate values to be returned when the loop is finished

Final actions: One last hurrah before the loop function is completed

There are two loop functions supported by BBL: `LOOP` and `FOR-EACH` (a special case of `LOOP`). The latter is diagrammed on the next page. Don't be put off by the complex possibilities. Let this page serve as a reference. For now, it might be prudent for you to gain familiarity with a small fraction of it rather than try to comprehend the whole at once.

II.C. Iteration control

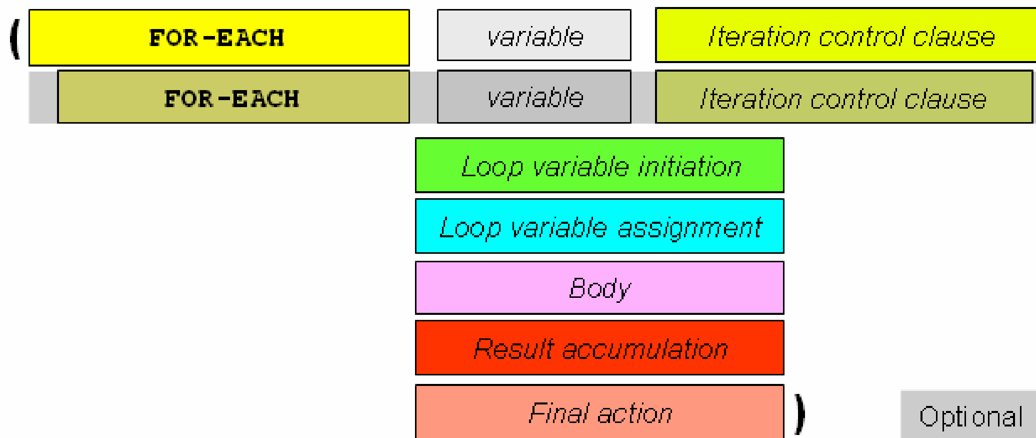
Loops repeat instructions over and over. How many times? This may be determined by the length of a list:

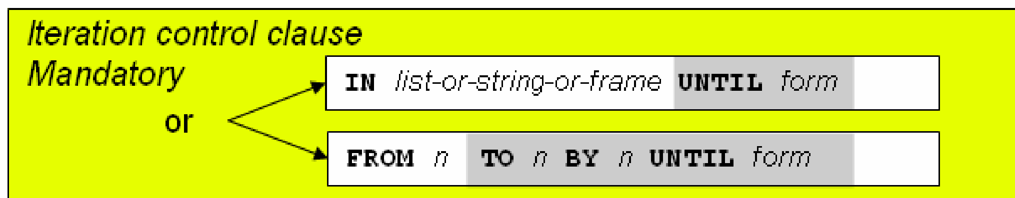
```
(FOR-EACH organism IN *all-organisms*
     (DISPLAY-LINE organism *tab* (LENGTH-OF organism)))
```

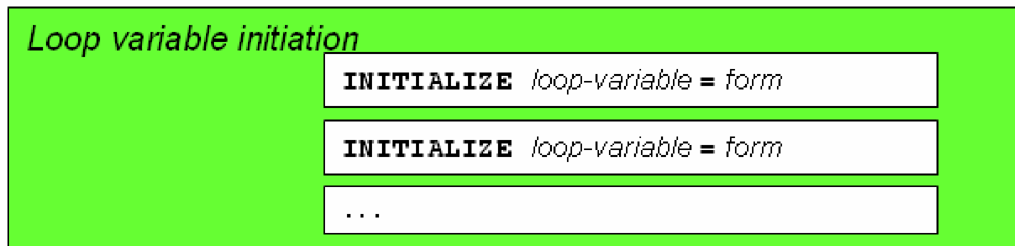
If BioBIKE knows of 13 organisms, then the body of the loop will execute 13 times, and the loop variable organism will assume the identity of each of the organisms, one at a time, for each time through the loop. Copy and paste the loop into BioBIKE and see what it does.

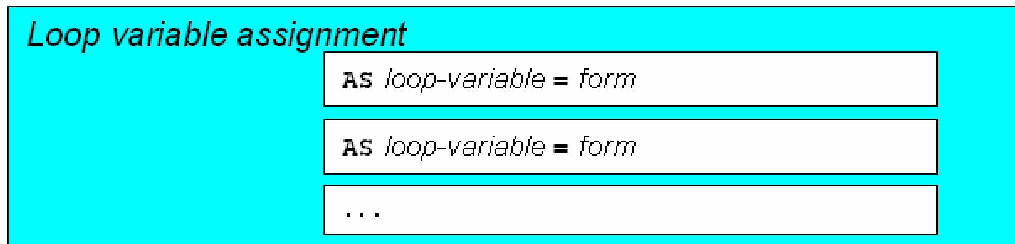
The iterations of the loop may be controlled by numbers as well:

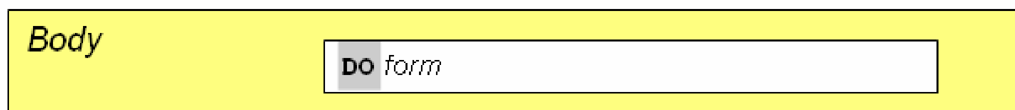
```
(FOR-EACH coordinate FROM 1 TO (LENGTH-OF all4312) BY 3
    AS end-of-codon = (+ coordinate 2)
    AS codon = (SEQUENCE-OF all4312 FROM coordinate TO end-of-codon)
    AS aa = (TRANSLATION-OF codon)
    COLLECT {codon aa} )
```

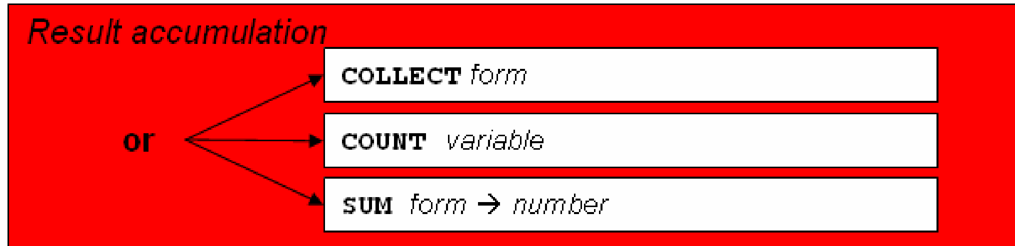

(
FOR-EACH
variable
Iteration control clause
FOR-EACH
variable
Iteration control clause
Loop variable initiation
Loop variable assignment
Body
Result accumulation
Final action
)
Optional


Iteration control clause
Mandatory
or
IN list-or-string-or-frame UNTIL form
FROM n TO n BY n UNTIL form


Loop variable initiation
INITIALIZE loop-variable = form
INITIALIZE loop-variable = form
. . .


Loop variable assignment
AS loop-variable = form
AS loop-variable = form
. . .


Body
DO form


Result accumulation
or
COLLECT form
COUNT variable
SUM form → number

Final action
FINALLY form

Translation:

a. *Consider each coordinate starting with 1, then 4, then 7, … until you've exceeded the length of the gene all4312*
b. *Each time through the loop, determine the end of the codon, by adding 2 to the current coordinate*
c. *Each time through the loop, determine the codon by extracting the sequence from the gene all4312, from the coordinate to the end of the codon*
d. *Translate the codon sequence to an amino acid*
e. *Add to a growing list of codons and corresponding amino acids*
f. *(Loop) Repeat steps b through e for each coordinate*

If you are not clear about how this loop works, then ask the program to tell you its inner thoughts as it goes. This is often a good strategy to see how code works:

```
(FOR-EACH coordinate FROM 1 TO (LENGTH-OF all4312) BY 3
    AS end-of-codon = (+ coordinate 2)
    AS codon = (SEQUENCE-OF all4312 FROM coordinate TO end-of-codon)
    AS aa = (TRANSLATION-OF codon)
    (DISPLAY-DATA coordinate end-of-codon codon aa)
    COLLECT {codon aa} )
```

This loop now displays all of the iteration-variables as it goes along, so you can see the loop in action.

Or the loop may be open ended, continuing until a condition is met:

```
(FOR-EACH coordinate FROM 1 BY 3
    UNTIL (> coordinate (LENGTH-OF all4312))
    AS end-of-codon = (+ coordinate 2)
    AS codon = (SEQUENCE-OF all4312 FROM coordinate TO end-of-codon)
    AS aa = (TRANSLATION-OF codon)
    COLLECT {codon aa} )
```

Translation:

a. *Consider each coordinate starting with 1, then 4, then 7, …open ended for the moment*
b. *Continue through the loop only so long as the coordinate currently under consideration is less than the length of the gene*
c. *Extract from the gene the triplet codon at the position of the coordinate*
d. *Add to a growing list of codons and corresponding amino acids*
e. *(Loop) Repeat steps c and d for each coordinate*

But be warned! Open ended loops are open to abuse:

```
(LOOP INITIALIZE hell-freezes-over = false
      UNTIL hell-freezes-over
      (DISPLAY "It's hot down here! "))
```

Don't do it!!! This loop will go on forever… until BioBIKE reaches the conclusion that you're yanking its chain and stops the proceedings.

Note that LOOP differs from FOR-EACH in that it allows you to construct loops that don't use iteration variables. Another difference illustrated in this example is that LOOP, unlike FOR-EACH, insists that all initialization clauses precede all iteration clauses.

II.D. Loop-variable initialization

Variables may be initialized before the loop begins using the INITIALIZE *variable* = *value* clause (INIT is a legal nickname). Variables thus initialized are created at the beginning of the loop and destroyed when the loop is finished. This is valuable when you want to initialize a variable that will be incremented by each iteration. For example:

```
(FOR-EACH gene IN (GENES-OF ss120)
     INITIALIZE ATG-count = 0
     INITIALIZE non-ATG-count = 0
     AS start-codon = (SEQUENCE-OF gene FROM 1 TO 3)
     (IF-TRUE (SAME start-codon "ATG")
         THEN (INCREMENT ATG-count)
         ELSE (INCREMENT non-ATG-count))
     FINALLY (RETURN (LIST ATG-count non-ATG-count)))
```

Translation:

a. *Consider each gene within the set of genes of Prochlorococcus marinus SS120*
b. *Initialize to zero two variables, one used to count ATG start codons and the other to count all other start codons*
c. *Extract the start codon from the gene under consideration*
d. *If the start codon is "ATG", then add 1 to the number of ATG start codons*
e. *Otherwise add 1 to the number of non-ATG start codons*
f. *(Loop) Repeat steps c through e until the set of genes has been exhausted.* ***Note that the initializations (step b) are NOT repeated.***

If you mistakenly tried to initialize the two variables with an AS clause (see next section), the results will not be as you might hope (try it!).

II.E. Loop-variable assignment

The values of variables may be revised each iteration using the AS *variable* = *value* clause. Variables thus assigned persist only for one iteration and then are destroyed. You therefore want to use this clause to initialize variables whose values differ from one iteration to the next.

```
(FOR-EACH gene IN (ORTHOLOGS-OF all4312)
     AS organism = (ORGANISM-OF gene)
     AS length = (LENGTH-OF gene)
     AS short-descr = (FIRST 20 IN (DESCRIPTION-OF gene))
     COLLECT {organism gene length short-descr})
```

Translation:

a. *Consider each gene amongst those with shared evolutionary antecedents as all4312*
b. *Identify the organism of the gene*
c. *Identify the length of the gene*

d. *Identify the first 20 characters of the description of the gene*

f. *Add to a growing list of identifiers the information collected for the gene under consideration*

e. *(Loop) Repeat steps b through f until the set of genes has been exhausted.* ***Note that the assignments (steps b through d) are redone each iteration.***

II.F. Body

The body of the loop is defined as the collection of statements not governed by a keyword. This collection must be in one piece, must come after all iteration, initialization, and assignment clauses, and must precede the accumulation clause and final action (if they exist). The body may be preceded by the optional keyword **DO**, if you think that aids readability. You can put any number of statements you like in the body, and each is executed during each iteration. Here's an example, illustrating the difference between initializing a loop-variable and assigning a value to a loop-variable within an iteration.

```
(LOOP INITIALIZE position = 1
      INITIALIZE letters = "ABCDE"
      INITIALIZE initialized-variable = letters[position]
      UNTIL (> position 5)
      AS assigned-variable = letters[position]
      DO (INCREMENT position)
         (DISPLAY-LINE "Here I am in the loop with the "
               "initialized variable = " initialized-variable
               " and the changing variable = "
               assigned-variable))
```

II.G. Result accumulation

Like all BioBIKE functions, loops return a value. If you pay no attention to what it returns -- for example if you're concerned only what the loop does in its body (as in the last example) -- then NIL will be returned. More often, however, you'll want the loop to return a value or a list of values. You've already seen several examples where a list of values was made and ultimately returned using the COLLECT clause. Here are three other ways of returning values:

- RETURN, which immediately exits the loop and returns a given value
- COUNT, which returns the number of times the clause is invoked
- SUM, which returns the sum of a number of items.

Note in the following examples how each of these methods (and COLLECT) work with WHEN, which governs whether or not the following clause is executed.

```
; Find very large genes
(FOR-EACH gene IN (GENES-OF A7120)
     AS length = (LENGTH-OF gene)
     WHEN (> length 2000)
       COLLECT {gene length} )
```

```
; Count very small proteins
(FOR-EACH protein IN (PROTEINS-OF ss120)
     AS MW = (MW-OF protein)
     WHEN (<= MW 10000)
       COUNT protein)

; How many nucleotides are in small genes?
(FOR-EACH gene IN (GENES-OF ss120)
     AS length = (LENGTH-OF gene)
     WHEN (< length 2000)
       SUM length)

; What's the first small gene?
(FOR-EACH gene IN (GENES-OF ss120)
     AS length = (LENGTH-OF gene)
     WHEN (< length 2000)
       (RETURN gene))
```