Introduction to Bioinformatics

Working with large numbers of items in BioLingua: 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?

BioLingua 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've already had experience with mapping. 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 (GENES-OF ss120) FROM 1 TO 10)
```

What you did was to map the function `SEQUENCE-OF` over the set of ss120 genes. Because BioLingua was designed with bioinformatics in mind, it is generally possible to map BioLingua 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 BioLingua code directly:

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

Recall that division, like all BioLingua 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, are too complicated to fit in one line. Then loops can save the day.

II. Loops

Mapping is simple: just replace a single item with a set of items. In contrast, looping uses what seems like a separate language. Here's the previous example rendered as a loop:

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

Translation:

a. *Consider each protein in the set of all proteins of ss120, one at a time.*
b. *Before the loop begins, set the sum of molecular weights to zero. This sum will change as the loop progresses.*
c. *Before the loop begins, set the number of proteins. This will be a constant.*
d. *Find the molecular weight of the one protein you're considering at the moment.*
e. *Add that molecular weight to the growing total.*
f. *(Loop) Repeat steps d and e until you've considered each protein in the set.*
g. *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
       DO (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.*

BioLingua 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.

Loops can be divided into the following optional parts:

Iteration control: Determine how the loop begins and ends

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

Iteration-specific initialization: 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

I'll consider each part and the most common variations you'll encounter. It must be said, however, that loop syntax is very involved – a language in itself – and 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.A. 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*
       DO (DISPLAY-LINE organism *tab* (LENGTH-OF organism)))
```

If BioLingua knows about 13 organisms, the body of the loop will execute 13 times.

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

```
(FOR-EACH coordinate FROM 1 TO (LENGTH-OF all4312) BY 3
     WITH sequence = (SEQUENCE-OF all4312)
       AS triplet = (GET-ELEMENT coordinate TO (+ coordinate 2)
                             FROM sequence)
       DO [something interesting with the triplet codon] )
```

← *marks the end of the loop*

Translation:

a. *Consider each coordinate starting with 1, then 4, then 7, … until you've exceeded the length of the gene all4312*
b. *Initialize the variable* `sequence` *to be the sequence of the gene all4312. This is done only once.*
c. *For each coordinate, extract the triplet codon at that position*
d. *Do something interesting with that codon*
e. *(Loop) Repeat steps c and d for each coordinate*

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

```
(ASSIGN gene = (GENE-NAMED all0409))
(ASSIGN intergenic-interval = 0)
(LOOP WHILE (< intergenic-interval 100)
    COLLECT gene
         DO (ASSIGN intergenic-interval
                = (LENGTH-OF (SEQUENCE-RIGHT-OF gene)))
            (ASSIGN gene = (GENE-RIGHT-OF gene)))
```

Translation (this one's pretty complicated):

a. *Set the variable* `gene` *to be the gene all0409*
b. *Set the variable* `intergenic-interval` *to be zero (just to get inside the loop)*
c. *Put the gene under consideration in a growing set of genes*
d. *Calculate the distance between the gene considered and the gene to its right*
e. *Make the gene to the right the gene under consideration*
f. *(Loop) Repeat steps c through e, so long as the distance between the two genes is small (< 100)*

This loop gives a set of genes that are close by each other.

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

```
(ASSIGN hell-freezes-over = false)
(LOOP UNTIL hell-freezes-over
         DO (DISPLAY "It's hot! "))
```

This loop will go on forever… until BioLingua reaches the conclusion that you're yanking its chain and stops the proceedings. This will occur after 40 seconds of execution. If you have a legitimate calculation that requires more than 40 seconds of computer time, you can up the time limit, but using gobs of computer time will not endear you to your colleagues who are using the same computer. So leave massive calculations to the wee hours.

II.B. Loop-specific initialization

Variables may be initialized before the loop begins using the `WITH` *variable* = *value* clause. Variables thus initialized are created at the beginning of the loop and destroyed when the loop is finished. There are two reasons why you might want to initialize this. First, you might want to initialize a variable that will be added to by each iteration. For example:

```
(FOR-EACH gene IN (GENES-OF ss120)
     WITH ATG-count = 0
     WITH non-ATG-count = 0
       AS start-codon = (SEQUENCE-OF gene FROM 1 TO 3)
       DO (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 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*

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!).

A second reason to use a `WITH` clause is to set a constant that may be used by all iterations of the loop. If you do this with an AS clause, the results will not change, but the loop may take (much!) more time to execute. For example:

```
(FOR-EACH coordinate from 1 TO (LENGTH-OF ss120) BY 1000
       AS upper-coordinate = (MIN (+ coordinate 999) (LENGTH-OF ss120))
       AS GC-fraction
         = (GC-FRACTION-OF
             (SEQUENCE-OF ss120
                   FROM coordinate TO upper-coordinate))
  COLLECT (LIST coordinate GC-fraction))
```

Translation:

a. *Consider each coordinate 1, 1001, 2001,… until the genome size of ss120 is exceeded*
b. *Calculate the fraction of nucleotides that are G or C in the 1000-nt interval of the chromosome starting from the coordinate under consideration*
c. *Save the coordinate and GC-fraction in a growing list of results*
d. *(Loop) Repeat steps b and c until the coordinate exceeds the genome size*

This code will work but will take nearly forever to run, because it takes a significant time for BioLingua to load into memory the entire genome sequence of ss120, which you're asking it to do over a thousand times. Execution time is dramatically decreased by:

```
(FOR-EACH coordinate from 1 TO (LENGTH-OF ss120) BY 1000
     WITH chromosome = (SEQUENCE-OF ss120)
     WITH ss120-length = (LENGTH-OF chromosome)
       AS upper-coordinate = (MIN (+ coordinate 999) ss120-length)
       AS GC-fraction
         = (GC-FRACTION-OF
             (SEQUENCE-OF chromosome
                   FROM coordinate TO upper-coordinate))
  COLLECT (LIST coordinate GC-fraction))
```

II.C. Iteration-specific initialization

Variables may be initialized during an iteration using the `AS` *variable* = *value* clause. Variables thus initialized 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 = (LEFT (DESCRIPTION-OF gene) 20)
  COLLECT (LIST organism gene length short-descr))
```

II.D. Body

The body of the loop is defined as those statements that follow `DO` and precede another loop key word or the end of the loop. 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 loop-specific variables and interation-specific variables:

```
(FOR-EACH n FROM 1 TO 5
     WITH loop-specific = n
       AS iteration-specific = n
       DO (DISPLAY-LINE "Here I am in the loop with the "
                "loop-specific variable = " loop-specific
                " and the iteration-specific variable = "
                iteration-specific)
           (ASSIGN iteration-specific
                 = (+ loop-specific iteration-specific))
  COLLECT iteration-specific)
```

II.D. Return value

Like all BioLingua 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 -- 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, using `RETURN`, which exits the loop and returns a given value, `COUNT`, which returns the number of times the clause is invoked, and `SUM`, which returns the sum of a number of items.

```
; Find the full organism name for a nickname
(FOR-EACH organism IN *all-organisms*
       AS nicknames = (GET-ELEMENT nicknames FROM organism)
     WHEN (POSITION-OF "Pmed4" IN-LIST nicknames)
         RETURN organism)

; Count the number of small proteins (< 10000 g/mole)
(FOR-EACH protein IN (PROTEINS-OF ss120)
       AS MW = (MW-OF protein)
     WHEN (<= MW 10000)
         COUNT protein)

; How many nucleotides in genome are devoted to genes?
(FOR-EACH gene IN (GENES-OF ss120)
       AS length = (LENGTH-OF gene)
      SUM length)
```

Note the use of WHEN, which governs whether or not the following clause is executed.