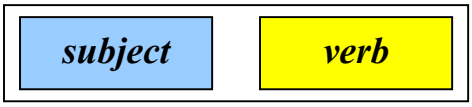
Introduction to BioLingua
BioLingua Syntax

A. General consideration of syntax

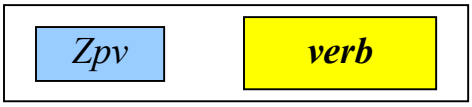
You might think that people who seem to know a computer language posses either some special knowledge or endowed with some magical ability to sense what's right. No… suppose you didn't know English. Then the first sentence ("You might think…") might look to you like this:

> *Zpv njhiu uijol uibu qfpqmf xip tffn up lopx b dpnqvufs mbohvbhf qpttft fjuifs tpnf tqfdjbm lopxmfehf ps foepxfe xjui tpnf nbhjdbm bcjmjuz up tfotf xibu't sjhiu*

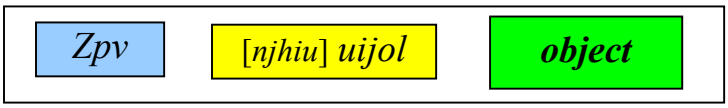
What would you do then? To have any hope of understanding this sentence, you'd have to know some meta-syntax, by which I mean the overall structure of sentences. You might know that many English sentences take the form SUBJECT – VERB.



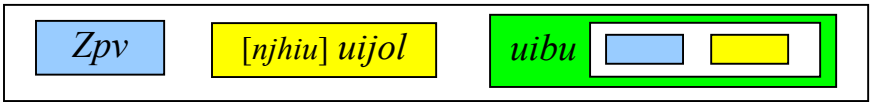
So you look up the first word in your dictionary. Can *Zpv* be a noun? Yes! So you can plug that in:



… and begin looking for the verb. After some scrounging around, you find in your dictionary that *uijol* is indeed a verb and that *njhiu* is a verb modifier, so you have:



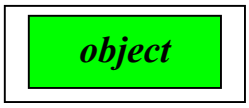
…but that verb has generated a new box, because *uijol* (says your dictionary) is a type of verb that takes an object. Can you really fit the rest of the sentence into the ***object*** box? You examine the next word: *uibu*. The dictionary says that it can introduce a sentence, and the whole thing can function as an object. So now you have:



And you go through another round to figure out the inside of the sentence.

That's just ***understanding*** English. To ***create*** new sentences from scratch, armed with only a dictionary, would be nearly impossible. English syntax is just too complex.

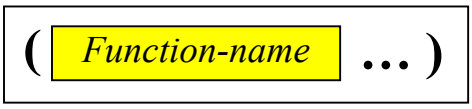
Fortunately, BioLingua syntax is not at all complex. There are only two basic meta-syntax forms:



Form 1 (atom)

Examples:
```
47          ==>  47
gene        ==>  value of gene
"hello"     ==>  "hello"
```

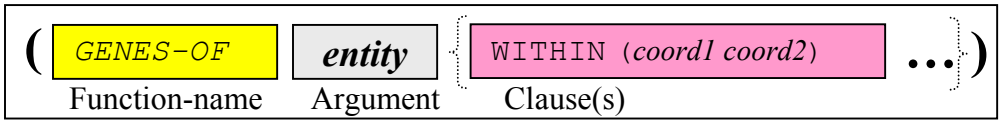
and:



Form 2 (function)

Examples:
```
(BIOLITE-VERSION)
(GENES-OF A7120)
(DEFINE x AS 47)
```

Just as some English verbs can generate new boxes, so can some BioLingua functions. You can learn the syntactical requirements of a BioLingua function by typing `(HELP` *function-name*`)`. For example, enter `(HELP GENES-OF)`… You learn the `GENES-OF` is synonymous with `GENE-OF`.[1] OK, enter `(HELP GENE-OF)`… From the documentation that `HELP` provides, you may discern the following syntax of `GENES-OF`:



The function `GENES-OF` generates a box that must be filled with an entity and optional clauses (only the `WITHIN` clause is shown here). The three types of boxes are:

Function-name, which must be filled with a single word that is the name of a legal function

Argument, which must be filled with a legal atom or function

Clause, which must be filled with a legal keyword (legality determined by the specific function) and the object of the keyword, which must be a legal atom or function

The function may impose further restrictions. `GENES-OF` demands that its argument be (if an atom) or produce (if a function) the name of a gene, protein, contig, replicon, or organism or a list of such names.

The following fit into this syntactical scheme:

```
(GENES-OF A7120)
(GENES-OF pNpA WITHIN (10000 20000))
(GENES-OF (ORGANISM-OF all4312))
```

The following do not fit into the scheme:

`GENES-OF A7120`

Functions are bounded by `( )`. `GENES-OF` *is interpreted as an atom.*

`(GENES OF A7120)`

Function-names are single words. `GENES` *is interpreted as the function name.*

[1] I hope that in the not ***too*** distant future, HELP will work directly on all functions without requiring a second trip to a synonym.

`(GENES-OF organism A7120)`

3rd symbol must be a keyword and must be followed by the object of the keyword.

`(GENES-OF A7120 SMALLAER-THAN 100)`

`SMALLER-THAN` *is not a keyword recognized by* `GENES-OF`

SQ1. For each statement below, find the syntax pattern of the function, identify each part of the statement as the ***function name*** or required ***argument*** or ***clause***. If the statement doesn't fit the syntax pattern, modify it so that it does.

1a. LEFT (SEQUENCE-OF all4312) 10
1b. (DEFINE "mole" (AS 6.02 * 10^23))
1c. (SEQUENCE-OF all4312 (FROM 1) (TO 20))
1d. (COUNT OF GENES OF A7120)

Finding the syntactical pattern of a function is like finding a very complete entry of a word in a dictionary. But how do you find the word in the dictionary if you're not sure what the word is? Not easy with English, but not so bad with BioLingua. You have the following strategies:

1. **`(HELP` *word*`)`**
   This gives you documentation of the function named *word*. If there is no such function, then you get a list of all the functions BioLingua knows about that relates to *word*.

2. **`(HELP "`*word*`")`**
   This gives you a list of all functions BioLingua knows about with the *word* in the name or documentation.

3. **BioLingua Help / Description of Functions (see *Resources & Links*)**
   Functions are organized by subject. You can scan their brief descriptions to find one you like. Clicking on the function brings you (sometimes) to documentation and examples. If such is lacking, at least you know what function name to use with `HELP`.

4. **Find a model**
   Think back on programs you've seen in the notes or elsewhere. One of them might do something like what you want. Get in the habit of figuring out other people's programs. Then steal shamelessly.

5. **Ask someone who knows**
   - Hit the panic button (*Who We Are* on course web page), reaching Jen and me
   - Ask anyone who happens to be on BioLingua:

   `(MESSAGE-TO ALL "`*message*`")`
   `(MESSAGE-TO` *user-name* `"`*message*`")`
   `(MESSAGE-TO (`*user-name user-name ...*`) "`*message*`")`

B. Symbols and symbol boundaries

You make sense out of English sentences only because you can tell where each word begins and ends. This may seem like a minor trick, but you're more complicated than a mere space-recognizer. Apart from spaces, you also recognize some punctuation (like parentheses, commas, and periods) but not others (like hyphens and apostrophes) as boundaries of words, and sometimes judgment and experience is needed.

BioLingua can't use judgment and experience and so must rely on well-defined rules instead. And here they are:

Spaces separate symbols. The number of spaces between symbols is not important (except within a quoted string) so long as the number is at least one. Thus the following are equivalent:

```
(GENES-OF A7120)
(GENES-OF               A7120)
(GENES-OF
          A7120)
```

Parentheses separate symbols. However, they also carry special meaning (delimiting functions or lists). They cannot be optionally thrown in for readability as they can in mathematics. So the following are equivalent:

```
(GENES-OF (ORGANISM-OF all4312))
(GENES-OF(ORGANISM-OF all4312))
```

but not:

```
(GENES-OF (ORGANISM-OF (all4312))
```

Double-quotes delimit a string that is used literally, not as a symbol. It can contain any symbol (even internal double quotes, through a trick). So the following are different:

`A7120`	A symbol containing the name of the organism *Anabaena* PCC 7120
`"A7120"`	The letter "A" followed by the digits "7", "1", "2", and "0"

Some other characters (`:;\#) also delimit symbols, but they have special meanings and so you shouldn't use them as delimiters unless you know what you're doing.

Comma (,) is a *very* special character, so much so that you will never see it in BioLingua statements unless you happen to wander into macros. Therefore, elements of a list are separated by spaces, ***not*** commas!

All other characters may be used within symbols and those symbols may be defined however you like. The following are all legal symbols:

`This-is-a-symbol?`	*Yes it is, including the question mark*
1+1	*Just a symbol. It's not necessarily equal to 2*
`@$%&!`	*Perfectly OK, if not pronouncible in polite company*

SQ2. For each statement below, predict the outcome, then try it out in BioLingua. Fix statements that need fixing. Figure out why things happen as they do.

1a. `(DEFINE 1+1 AS 3)`
1b. `(DEFINE "dozen" AS 12)`
1c. `(3+ 1)`
1d. `(+ 3 +1)`
1e. `(1+ 3)`
1f. `(DEFINE 5-(+ 5 -3))`
1g. `(* 3 "2")`

BioLingua makes no distinction between upper case and lower case, unless the characters lie between double quotes. The following are therefore equivalent:

```
(DEFINE my-proteins AS (PROTEINS-SIMILAR-TO p-all4312 IN S6803))
(define MY-PROTEINS as (proteins-similar-to P-aLL4312 in s6803))
```

I have chosen to render function and keyword names in capital letters and variables in lower case, to improve readability, but that' just my choice.

Even though BioLingua doesn't care about case, the system in which it's running, Linux, ***does*** care, and it is the system that worries about files and file names. Therefore, when you're saving or loading a file, you need to pay attention to upper/lower case. Since only strings (e.g., characters between double quotes) retain case distinction in BioLingua, you must refer to filenames as strings. For example:

```
(LOAD-SHARED-FILE "My-Favorite-Proteins")
```

will work ***only*** if the file has this exact name, capitals and all.

C. Form boundaries

The BioLingua Web Listener executes one form at a time. That form can be a single symbol or five pages of code. Either way, you get one and only one form executed. That's why entering the following code does not give an error, even though you might expect it to:

```
1 + 1 = 2
```

The Listener encounters the first form, the atom 1, returns its value, and ignores the rest.

It's obvious what is the extent of a form that happens to be an atom – it's just one symbol long. The case is often not so clear if the form is instead a function. A function extends from its opening parenthesis to the ***matching*** closing parenthesis. In the case of complex functions like loops, that closing parenthesis may be quite far away and difficult to recognize. Fortunately, there are tools to aid your eye. If you put code in the large program window and place the cursor after a closing parenthesis, the web listener will tell you in the information box below where is the opening parenthesis. There are also more sophisticated programming aids (see ***Links & Resources***).

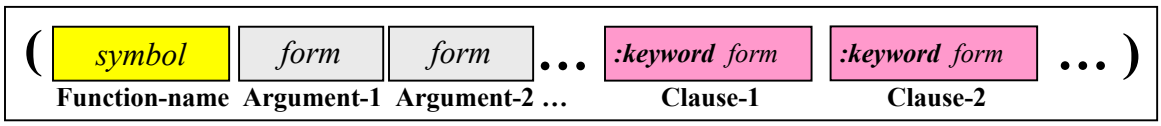
SQ3. What result does the code below return when executed? Why?

```
(FOR-EACH number FROM 1 TO 100)
       DO (DISPLAY number " ")
      SUM number)
```

D. The levels of BioLingua: Lisp, BioLingua, BioLingua-Lite

Now the secret comes out: the language you have been using is a dialect of an extension of the general purpose programming language called Lisp. Lisp offers unusually powerful tools that enable users to build extensions of the language, which is why we're using it. Lisp and BioLingua-proper use a very strict and very simple (some would say very beautiful) syntax. BioLingua-proper adds to Lisp a large number of functions useful to biologists. BioLingua-Lite syntax is fuzzier at the boundaries but makes fewer demands on humans. The language you use is an amalgam of functions from all three sources. It may therefore be helpful to understand the syntactical requirements of each.

Syntax of Lisp and BioLingua-proper functions



Function-name

- Must be first symbol, the name of a Lisp or BioLingua-proper function
- One function has one name

Arguments

- The function may require any number of arguments and allow any number of optional arguments
- Each argument consists of a form (atom or function) that evaluates to a defined value. This means that gene names won't work. The Lisp function below fails unless you have previously defined the gene names to contain a value.

  `(EXTRACT-SEQUENCE all4312)` → Error! (undefined variable)

  but `(EXTRACT-SEQUENCE #$A7120.all4312)` → `(ATG...)`
- An argument usually must be of a certain type: it ***must*** be a list or it ***cannot*** be a list; it ***must*** be a string or it ***cannot*** be a string; ***must*** be a gene or ***cannot*** be a gene, etc.

  `(MOLECULAR-WEIGHT-OF "MARGGRC...")` → molecular weight of sequence

  but `(MOLECULAR-WEIGHT-OF #$A7120.p-all4312)` → Error! (wrong type)

Clauses

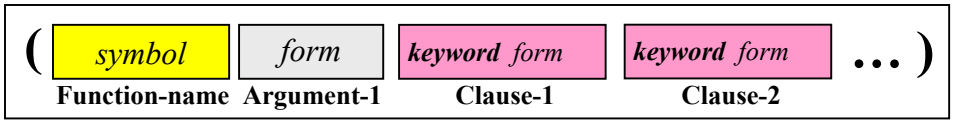
- The function may not require clauses, but may allow any number of optional clauses.
- A clause begins with a keyword defined by the function, preceded by a colon. The keyword is followed by a form (atom or function) that evaluates to a defined value (not a gene name).

Explicit lists

- A list consists of items enclosed in parentheses. To distinguish it from a function, the opening parenthesis is preceded by a single quote.

  `(SETF x '(1 2 3 4))` → Assigns the list of four numbers to the variable `x`
- The items must all evaluate to a defined value (not a gene name).

Syntax of BioLingua-Lite functions



Function-name

- Must be first symbol, the name of a BioLingua-Lite function
- Many functions have multiple synonymous names (e.g. `GENES-OF` and `GENE-OF`)

Arguments

- The function requires either zero or one argument and allow no optional arguments
- The argument (if it exists) consists of a form (atom or function) that may or may not evaluate to a defined value. Undefined symbols (like gene names) are interpreted as you probably want them to be. For example:
  `(SEQUENCE-OF all4312)` → `"ATG..."`
- When possible, the function converts an argument to the type needed by the function.
  `(MW-OF "MARGGRC...")` → molecular weight of sequence
  and `(MW-OF p-all4312)` → molecular weight of named protein

Clauses

- The function may or may not require clauses and may allow any number of optional clauses.
- A clause begins with a keyword defined by the function (colon is optional). The keyword is followed by a form (atom or function) that may or may not evaluate to a defined value.
- A clause may accept any of a number of possible types and direct the appropriate action.
  `(COUNT-OF "M" IN p-All4312)` → How many M's in protein sequence
  and `(COUNT-OF "ATG" IN "ATGACAGGGA…")` → How many ATG's in sequence
  and `(COUNT-OF "ATG" IN (SEQUENCE-OF (GENES-OF ss120) FROM 1 TO 3))`
  → How many ATG's in list of start codons

Explicit lists

- A list consists of items enclosed in parentheses. A BioLingua-Lite function distinguishes a list from a function by whether the first item is a function-name.
- The items in a list need not evaluate to a defined value (gene names are OK).

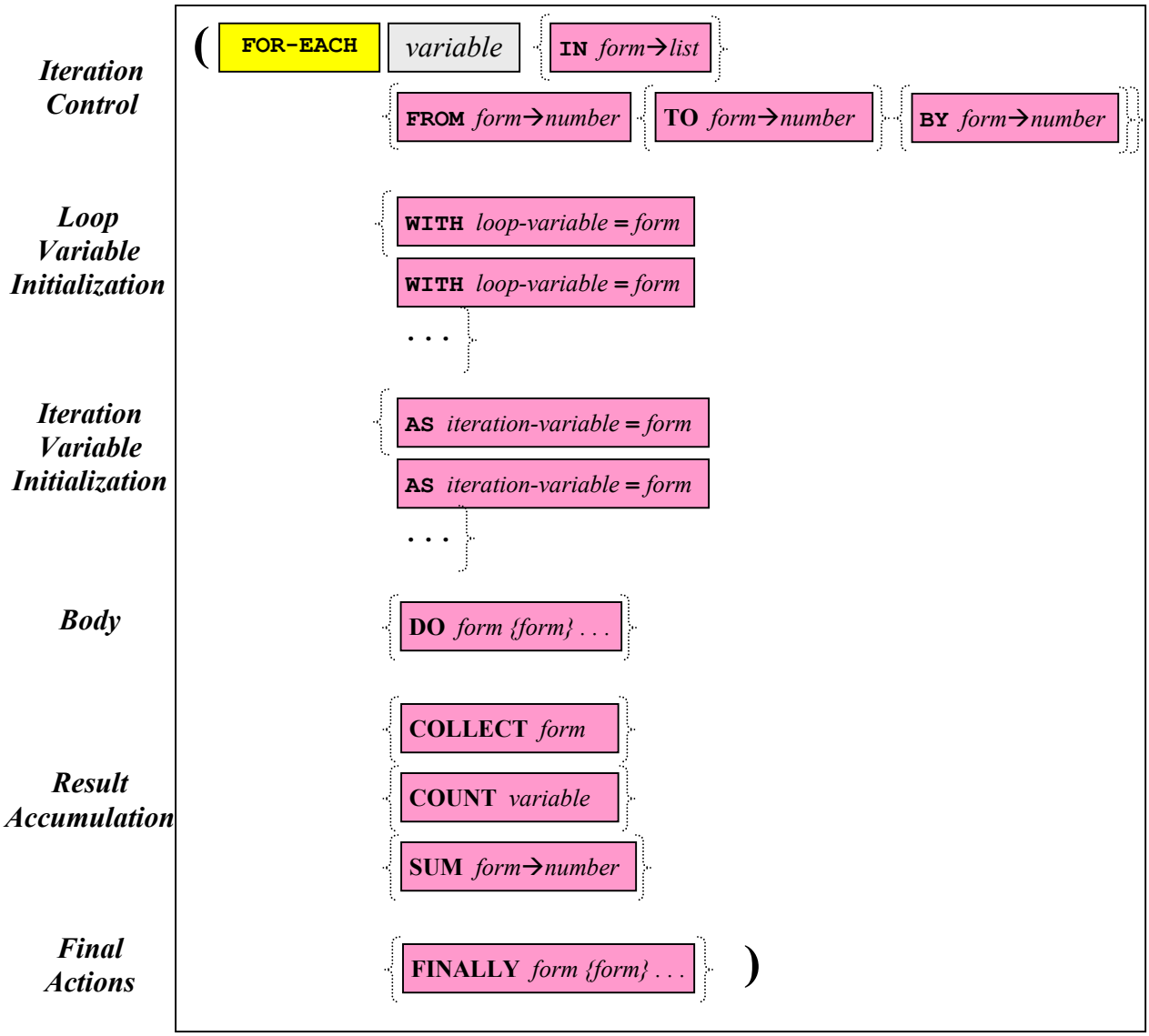
It would be nice if a programming language were coherent, that is the entire language behaved in the same way. If it's coherence you want, then you must stick (at our present stage of evolution) to Lisp or BioLingua-proper. If you want to avail yourself of the conveniences offered by BioLingua-Lite, then you must accept that it is an evolving and as yet incomplete language.

Still, it might seem pretty important to be able to distinguish between, say, a Lisp function and a BioLite function. What clues are there? Actually, you can go pretty far assuming that all the functions you use adhere to BioLite conventions. Most of the time you'll be right, and error messages will quickly amend your thinking if you are wrong. If you'd like to reduce the frequency of errors, the documentation of any function will tell you the function's requirements, and you'll probably visit the documentation before using the function for the first time anyway. Furthermore, the names of BioLite functions are often phrases ending in prepositions (like `SEQUENCE-OF`) instead of BioLingua-proper commands (like `EXTRACT-SEQUENCE`). The coexistence of three languages is not as big a deal as you might think.

SQ4. Look back on a half-dozen functions you've used and identify which language they come from.

E. Loops

The functions `LOOP` and `FOR-EACH` have by far the most complex syntax of any function you'll encounter in BioLingua. Here's a ***partial*** description of `FOR-EACH`:



Note that only the first section, ***Iteration Control***, is required. It is perfectly legal to write a loop such as…

```
(FOR-EACH n FROM 1 TO 1000000)
```

…that does absolutely nothing except waste time. Note that the loop functions use Lisp language conventions, despite the nonstandard clause structure (Lisp purists look upon them with suspicion). Therefore explicit lists must be quoted:

```
(FOR-EACH word IN '("cat" "dog" "horse") ...)
```

SQ5. Compose several loops making use of some of the clause options shown.