BioBIKE Language Syntax
General Consideration of Syntax

A. English syntax as a model for the syntax of computer languages

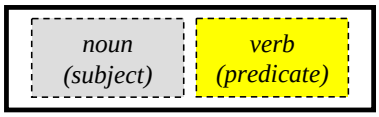
You might think that people who seem to know a computer language either possess some special knowledge or are endowed with some magical ability to sense what's right.

You were probably able to extract some meaning from that complicated sentence, and if so, you have all the basic tools necessary to understand a computer language. You did it, no doubt, without thinking, so let's try it again, this time WITH thinking. So wipe out your knowledge of English.

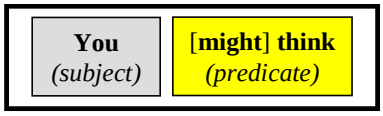
NOW how do you understand the sentence?

Well, of course you'd need an English dictionary. You look up *You* and get the basic meaning and the fact that it is a noun. You could look up the rest the words as well, one by one, but that clearly isn't enough. You also need to know some rules as to how the words work together,... syntax!

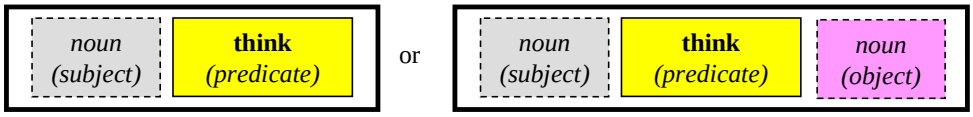
Suppose you learn that English sentences may take a variety of structures, one of the simplest being:



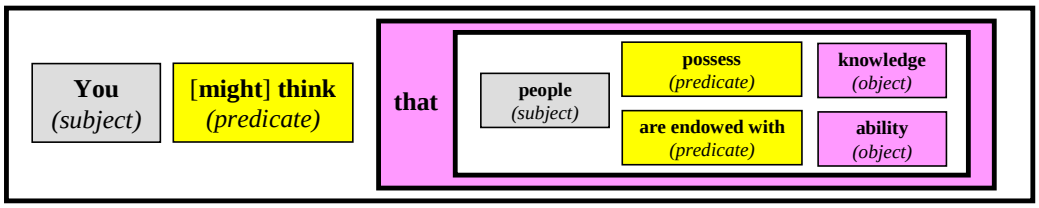
You is a noun, *might* is a verb modifier, and *think* is a verb, says your dictionary. So far:



This, evidently, is a legal utterance, but what about the other 25 words of the sentence? Looking more closely at the dictionary entry for *think*, you find that it can serve as either a transitive verb (taking an object) or an intransitive verb (no object). So both of the following structures are OK:



This is useful, because your dictionary says that the fourth word, *that*, can introduce phrases that can replace nouns, perhaps like so:



Look back on these boxes:

- Boxes with ***dotted boundaries*** represent defined holes
- Boxes with ***solid boundaries*** represent holes filled with an object
- Boxes with ***thick solid boundaries*** represent holes filled with functions that return an object

Complicated! Fortunately, computer languages are much simpler (otherwise computers couldn't understand them), and BioBIKE Language (BBL) is about as simple as you can get and still retain the ability to express everything you need in a language. However, like human languages, computer languages increase their powers of expression by permitting forms to be placed within forms multiple times.

SQ1. Complexify the sentence in the last box even further.

SQ2. Parse the following sentence, putting the words in appropriate boxes:

"Buffalo buffalo Buffalo buffalo buffalo buffalo Buffalo buffalo"

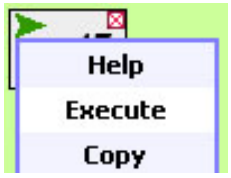
It might help to view "Buffalo" as referring to the city in New York.

B. Syntactical distinctions in BioBIKE

B.1. Atoms

Log into BioBIKE, mouse over the black **EDIT** button in the palette menu, and click **New Data Box**. You should see:

It is a dotted box, which is to say a hole, and the hole claims anything can go inside of it. OK, try clicking on the box, typing in your favorite number (I chose 47), then pressing **Enter**. Now execute the completed box, by mousing over the green action arrow, and clicking on **Execute**.



What could it possibly mean to "execute" 47? Evidently, BioBIKE thinks it makes sense, because if you examine the blue Result window, you'll see that it gave you a result: 47. You can type any literal object into the box, e.g., 47 or "banana" (including the quotes on each side), and in each case, executing the box returns the literal object. These objects are atoms: things that are indivisible. We'll see what non-atoms are momentarily.

SQ3. What happens if you put more than one atom in a box?

B.2. Literal atoms vs variable atoms

Edit the box you were working with in the previous section, by clicking within the box or by mousing over the green action arrow and clicking **Edit**. Now put in the box your favorite letters. When you put in "banana" before (and you ***did*** try it out with your own fingers, right?), you used quotes. This time, just type in the letters with no quotes. Press **Enter, Execute** as before, and ... whoops! An error message "`Attempt to take the value of the unbound variable` `` `XYZ'. ``" What's that all about?

BioBIKE, like any computer language, allows you to refer to values using symbolic names. You can have the symbol **x** refer to the value 47, or "banana", or many other types of things. Let's fix your error. Go to the **DEFINITION** menu and click on **DEFINE**. This function asks you to

provide the ***name*** of the variable and then its desired ***value***. Give as the name whatever letter combination you used that led to the error message, press **Tab,** and then give as the value your favorite number. Press **Enter**, then **Execute** the function. No fireworks result, but the symbol you defined now contains the value you gave it.

Now go back to the box with that symbol in it and re-**Execute** it. Before, executing gave an error message. Now, executing it should give the value of the variable. Variables are also atoms, and executing any atom returns the value of the variable. The value of "x" is "x", but the value of the variable **x** is whatever value you've given it.

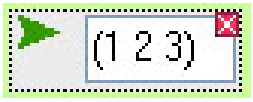
Any box that can take a literal of a certain type (e.g. a literal number) can also take a variable that contains that type of value (e.g. a variable containing a number).

SQ4. Note that once you've executed DEFINE, a new button appears, called VARIABLE. What's in that menu? What happens when you click on the thing inside? Experiment with clicking on it under different circumstances.

SQ5. What can you name variables? Try DEFINing a variable called one+one as equal to 3. Try putting other symbols into the name of a variable. What's possible and what's not?

B.3. Atoms vs aggregates

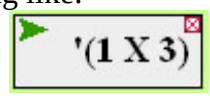
Not everything is indivisible. For example, the set of integers from 1 to 3 is composed of 3 atoms. There are many ways in BioBIKE to aggregate atoms. The most common is a ***list***, which contains an ordered sequence of elements. The elements are most often atoms but can themselves be lists. For example, go back to the box you started with in Section B.1 and edit it so that it contains a list of integers from 1 to 3. To make that list, you can simply put the numbers between parentheses.



Pressing **Enter** to accept the entry and then executing the box returns the value of the list, which is a list of the three numbers.

Recall that you defined some variable (say, **x**) a moment ago. What happens if you put the variable in the list? Try doing that and then execute the box.

…no error, but probably not the result you were hoping for. Here's a clue as to why you got the result you did. Notice that when you pressed **Enter**, the entry box closed (turned gray), and you got something like:



The list is preceded by a single quote, indicating that the list that follows is to be interpreted as literal atoms, including the ***symbol*** x, not the value of the ***variable*** x. There is no conflict with my previous claim that "*Any box that can take a literal... can also take a variable...*", because the variable is not in a box (not its own box), it's in a list! To make a list that includes the value of **x**, we need to provide a separate box for each element, using the function called **LIST**. Go to the

LISTS-TABLES menu and click on **LIST**. Then click on the options arrow to add two more items, giving:

Now yc 

of the elements of the list: 1, **x**, and 3, being sure to press **Tab** to terminate entry for each item. Once that is done, execute. If you've defined **x**, then you should produce a list containing the ***value*** of **x**.

Use (…) to make literal lists and use the **LIST** function to make lists that evaluate variables contained within them.

Bioinformatics is the art of working with huge aggregates of data. There are other kinds of aggregates that facilitate working with different kinds of data, but we'll consider them later.

SQ6. Create a list consisting of three elements: the first being the first three counting numbers, the second being the first three uppercase letters of the alphabet, and the third being the first three lowercase letters of the alphabet.

B.4. Functions

In English, a noun may be that which functions as a noun (the latter phrase being a convoluted example). Similarly, an atom of a certain type may be replaced by a function that produces a value of that type. English phrases have a bewildering number of formats. You will be pleased to learn that all BioBIKE functions have a single general format:

You might think of this as *verb – noun – modifiers*. The function box is represented with a thick solid boundary because it produces an object and therefore can fit into an object hole. The ***argument*** is an object given to the function (just like 90° may be an argument given to the sin function). An optional ***keyword clause*** contains an object, but optional ***flags*** do not. Keyword clauses and flags are represented by colored lines because they cannot be filled with objects. They're part of the function syntax. Depending on the function, there may be more than one argument, more than one keyword clause, or more than one flag.

All functions have at least the verb, the function name. Some consist ***only*** of the function name. For example:

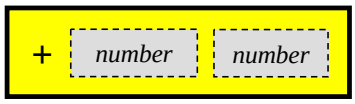
…a verb-only function that tells you who else is using the system. Some functions have only the function name and a required argument, such as:

Many functions, perhaps most, allow optional keyword clauses to supply the function with additional values and optional flags to modify the operation of the function. For example:

Like all BioBIKE functions, this one produces an object (in this case a DNA sequence) that logically fits into an object hole. When the function is evaluated, the DNA sequence replaces the function, and you're left with just that sequence, an object, which is a legal BioBIKE form.

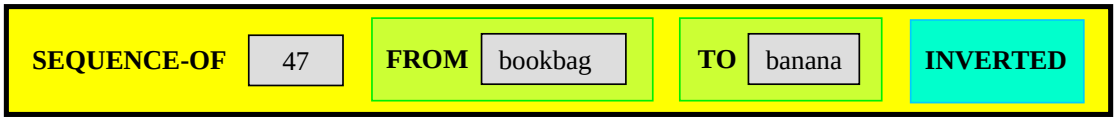
SQ6. Be sure that you actually execute all of these functions. Use simple values to figure out what they do.

It's important to see that ***all*** BBL functions have this general verb-first form, even functions you might expect to behave otherwise. For example, 1 + 1 is fit into the same form: Verb first, then arguments:



It may seem strange to write addition in this way, but at the cost of promising the computer that all functions will be rendered verb first, we gain a freedom of expression unmatched in other languages. If the verb always comes first, then whatever comes first must be a verb, and we can ***invent*** new verbs with the confidence that they will be recognized as such.

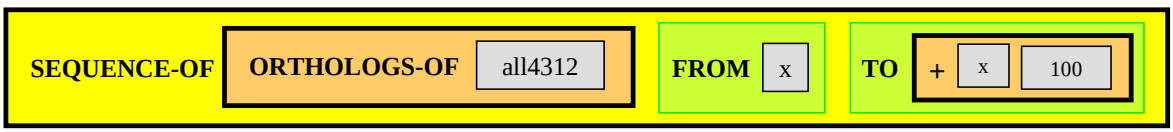
The addition example illustrates that the syntax of a function may go beyond the mere specification of object holes: it can also impose a restriction as to the type of an object, e.g. a number. Here's another example:



BioBIKE will reject this sentence because SEQUENCE-OF demands that its argument produces a sequence and that FROM and TO be followed by numbers.

It's clearly important to find out what are the syntactical requirements of any function you want to use. How do you do this? The first clue is the prompts given in holes provided by a function. **SEQUENCE-OF** (pulled down from the **GENOME** menu) specifies that it needs an *entity* (e.g. a gene or a protein). Another way is to mouse over the green action arrow and click on **Help**. This gives you a brief description of the function and a link to more information as to what are the syntactical requirements.

If all the language could do is perform a necessarily limited number of functions, it would be as limited as English would be if limited to simple subject-predicate-object sentences. But like English, BioBIKE allows you to replace any object with a function that produces an object. This makes complex expression possible. For example:



Try this out! **ORTHOLOGS-OF** is a function that returns genes that are similar to the given gene. You can find it in the **GENES-PROTEINS** menu.

SQ7. Make a function nested to three levels (a function within a function within a function).

B.4. Result vs display

If you put frozen macaroni and cheese into a microwave, provide some additional arguments (e.g. the duration of cooking), and then click *Execute*, in not too long you'll get the ***result*** of the function: a hot dinner. The microwave may also beep at you to tell you it's done, a transient ***side effect*** of the operation. You can do further computations on the result of the function (for example eating it), but ordinarily, the side effect is acknowledged (perhaps), and then it's gone forever.

Every function in BioBIKE returns a result when it concludes successfully. **LENGTH-OF** returns the length of something. **SEQUENCE-OF** returns the sequence of something. But sometimes things are more complicated. Suppose you are using an intelligent version of **SIN** (more intelligent than BioBIKE's version), found in the **ARITHMETIC** menu, **Trigonometric Functions** submenu. You provide it with an angle, say 180°. The function gives you a ***result***, -0.80115265, but at the same time it displays a message in a popup box: "*It looks like the angle you gave me is in degrees, but I'm currently set up to work in radians.*" This is a ***side effect*** of the function. If you then grab **ABS** from the **ARITHMETIC** menu and **PREVIOUS-RESULT** from the **OTHER-COMMANDS** menu and **Execute**:

getting 0.80115265. Of course. You didn't expect to get the absolute value of "*It looks like the angle you gave me…*". The **PREVIOUS-RESULT** is the previous ***result***, not the previous ***side effect***. In fact, side effects are promptly forgotten by BioBIKE. It remembers only results.

This all may seem obvious to you, but try this:

(getting **DISPLAY** from the **INPUT-OUTPUT** menu), and then:

Why did you get the result you got?

SQ8. Display the sequence of the first 1000 nucleotides of A7120.chromosome (see GENOMES menu) and verify that the length of this sequence is 1000.