

# Writing Scratch/BYOB code on paper

You might be asked to write Scratch/BYOB code on exams, so we've developed a technique for writing it on paper. There are a few key things to notice:

- We write variables in **UPPERCASE**.
- We change spaces between words in block names to dashes (this makes it much easier to read).
- Parentheses mark the start and end of a parameter list, and we separate consecutive parameters by commas
- We use indentation just as Scratch/BYOB does, to help us understand what is "inside" the **if**, **else**, and other Control structures.
- When you want to write a list of things, write them with an open parenthesis, then the first item, second item, etc (separated by spaces) and when you're done, put a closed parenthesis. If any of your items are a sentence, you have to put quotes around the sentence. So, for example, the following list of three things would be written as the equivalent 3-element-list:



- `(life liberty "pursuit of happiness")`.

- Similarly, a nested list just shows up as a nested set of parenthesis. So the following would be written as

- `((Love 5) (Hate 4) (The 10))`.

- If you want to pass in a function as argument, you have two options in BYOB: use the grey-border or the more verbose `the( )block` green block. Here are three new conventions:

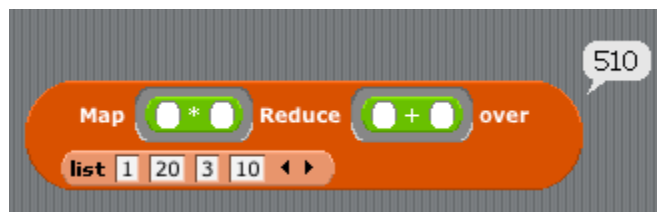
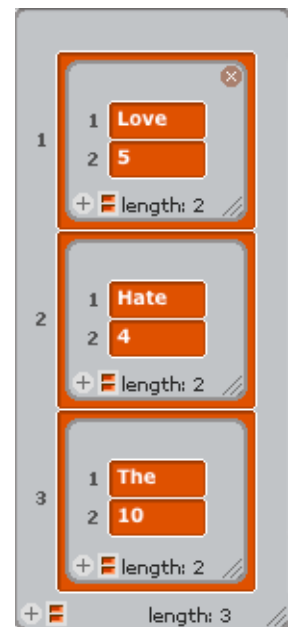
- The grey border is written as square brackets: `[ ]`
- Blanks are written as parenthesis with underscore `_` in the middle, but common blocks that are passed in to HOFs can be simplified by just their name (and not the parens and underscores)
- Return values are written as `==> value`

- So the following would be written as:

- `Map[ (_)*(_) ]Reduce[ (_)+( _) ]over( (1 20 3 10) ) ==> 510`

- or, in the more simplified (and preferred) format:

- `Map[ * ]Reduce[ + ]over( (1 20 3 10) ) ==> 510`

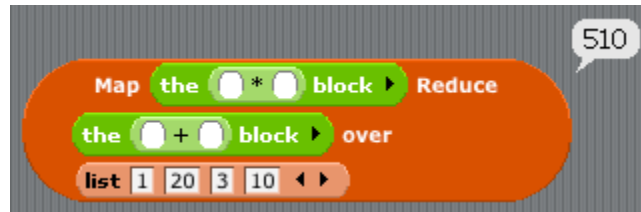


- If you prefer to use the `the( )block` green block, it could also be written:

- `Map(the( (_)*(_) )block)Reduce(the( (_)+( _) )block)over( (1 20 3 10) ) ==> 510`

- or, in the more simplified (again, preferred) format:

- `Map(the(*)block)Reduce(the(+)block)over( (1 20 3 10) ) ==> 510`



Here's a sample (and a familiar piece of BYOB code):



...and here's how we would write it on an exam using our technique:

```
downup(WORD)
  if length-of(WORD) < 2
    report(WORD)
  else
    report(join-words(WORD, downup(all-but-first-letter-of(WORD)), WORD))
```

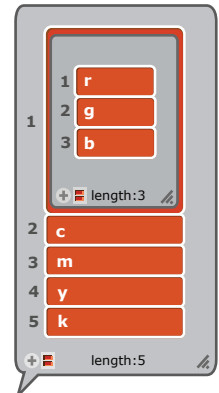
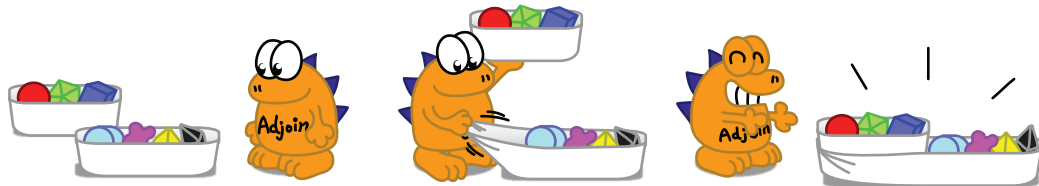
Here's how you could write the `factorial-of` block from lab.

```
factorial-of(NUM)
  if NUM = 1
    report(1)
  else
    report(NUM * factorial-of(NUM - 1))
```

# List Constructors

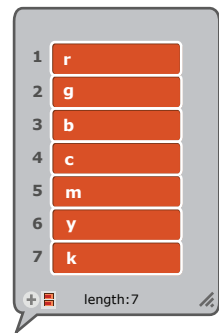
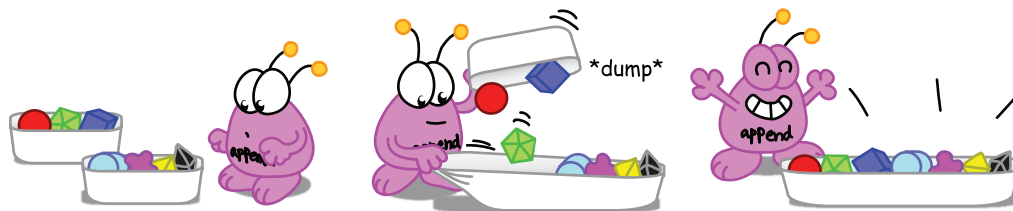
When given two lists...

adjoin



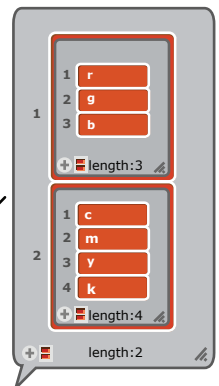
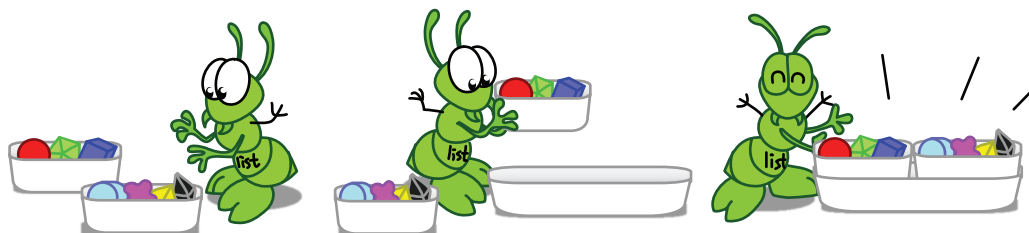
```
adjoin list r g b ◀ ▶ to list c m y k ◀ ▶
```

append



```
append list r g b ◀ ▶ list c m y k ◀ ▶ ◀ ▶
```

list



```
list list r g b ◀ ▶ list c m y k ◀ ▶ ◀ ▶
```

**!** While it appears that list constructors `adjoin` and `append` modify (i.e., stretch) the second input list, they don't change their input lists at all. Instead, they return a new structure.



Adjoin takes exactly 2 arguments while `append` and `list` take as many arguments as you want.



# List Constructors




These are three very common list constructors in the **Variables** menu. Each returns a new list and doesn't modify the input arguments. This is meant to supplement the "List Constructors" CS Illustrated handout.

## ADJOIN

- Available via **ToolSprite** sprite or `tools.ypr` project, and takes exactly two arguments
- In BYOB it is shown as "adjoin **ITEM** to **LIST**", i.e., 
- Reports a new list, the result of attaching the **ITEM** to the front of **LIST**, which is shown on the left if we're writing BYOB code on paper (as below), and in BYOB visually as the new first element in the top (1) slot.
- There is a similar block "adjoin to **LIST** this item **NEW** on the right", i.e.,  that attaches **NEW** to the end (right) of **LIST** if we're writing BYOB code on paper, and in BYOB is shown visually in the last slot on the bottom.




	ITEM	LIST	
<code>adjoin(</code>	<code>x,</code>	<code>z</code>	<code>) ==&gt; ERROR, 2nd argument to adjoin has to be a list</code>
<code>adjoin(</code>	<code>()</code>	<code>z</code>	<code>) ==&gt; ERROR, 2nd argument to adjoin has to be a list</code>
<code>adjoin(</code>	<code>(r g b)</code>	<code>z</code>	<code>) ==&gt; ERROR, 2nd argument to adjoin has to be a list</code>
<code>adjoin(</code>	<code>x</code>	<code>()</code>	<code>) ==&gt; (x)</code>
<code>adjoin(</code>	<code>()</code>	<code>()</code>	<code>) ==&gt; (() ;; a list w/1 element in it, an empty list</code>
<code>adjoin(</code>	<code>(r g b)</code>	<code>()</code>	<code>) ==&gt; ((r g b)) ;; a list of the 3-element list (r g b)</code>
<code>adjoin(</code>	<code>x</code>	<code>(c m y k)</code>	<code>) ==&gt; (x c m y k)</code>
<code>adjoin(</code>	<code>()</code>	<code>(c m y k)</code>	<code>) ==&gt; (() c m y k)</code>
<code>adjoin(</code>	<code>(r g b)</code>	<code>(c m y k)</code>	<code>) ==&gt; ((r g b) c m y k) ;; as in CS Illustrated handout</code>

## APPEND

- \* Available via **ToolSprite** sprite or `tools.ypr` project, and takes any number of arguments (even 0)
- \* In BYOB it looks like  or  or  or ...
- \* Reports the result of merging all input lists into a single list. If the inner lists have lists, it doesn't merge those.

	LIST1	LIST2	
<code>append(</code>	<code>x</code>	<code>z</code>	<code>) ==&gt; ERROR, all arguments to append must be lists</code>
<code>append(</code>	<code>()</code>	<code>z</code>	<code>) ==&gt; ERROR, all arguments to append must be lists</code>
<code>append(</code>	<code>(r g b)</code>	<code>z</code>	<code>) ==&gt; ERROR, all arguments to append must be lists</code>
<code>append(</code>	<code>x</code>	<code>()</code>	<code>) ==&gt; ERROR, all arguments to append must be lists</code>
<code>append(</code>	<code>()</code>	<code>()</code>	<code>) ==&gt; ()</code>
<code>append(</code>	<code>(r g b)</code>	<code>()</code>	<code>) ==&gt; (r g b)</code>
<code>append(</code>	<code>x</code>	<code>(c m y k)</code>	<code>) ==&gt; ERROR, all arguments to append must be lists</code>
<code>append(</code>	<code>()</code>	<code>(c m y k)</code>	<code>) ==&gt; (c m y k)</code>
<code>append(</code>	<code>(r g b)</code>	<code>(c m y k)</code>	<code>) ==&gt; (r g b c m y k) ;; as in CS Illustrated handout</code>

## LIST

- Built-in to BYOB, and takes any number of arguments (even 0)
- In BYOB it looks like  or  or  or ...
- Reports the result of wrapping all the input elements in a list, in the same order they came in.

	ELEMENT1	ELEMENT2	
<code>list(</code>	<code>x</code>	<code>z</code>	<code>) ==&gt; (x z)</code>
<code>list(</code>	<code>()</code>	<code>z</code>	<code>) ==&gt; (() z)</code>
<code>list(</code>	<code>(r g b)</code>	<code>z</code>	<code>) ==&gt; ((r g b) z)</code>
<code>list(</code>	<code>x</code>	<code>()</code>	<code>) ==&gt; (x ())</code>
<code>list(</code>	<code>()</code>	<code>()</code>	<code>) ==&gt; (() ())</code>
<code>list(</code>	<code>(r g b)</code>	<code>()</code>	<code>) ==&gt; ((r g b) ())</code>
<code>list(</code>	<code>x</code>	<code>(c m y k)</code>	<code>) ==&gt; (x (c m y k))</code>
<code>list(</code>	<code>()</code>	<code>(c m y k)</code>	<code>) ==&gt; (() (c m y k))</code>
<code>list(</code>	<code>(r g b)</code>	<code>(c m y k)</code>	<code>) ==&gt; ((r g b) (c m y k)) ;; as in CS Illustrated handout</code>