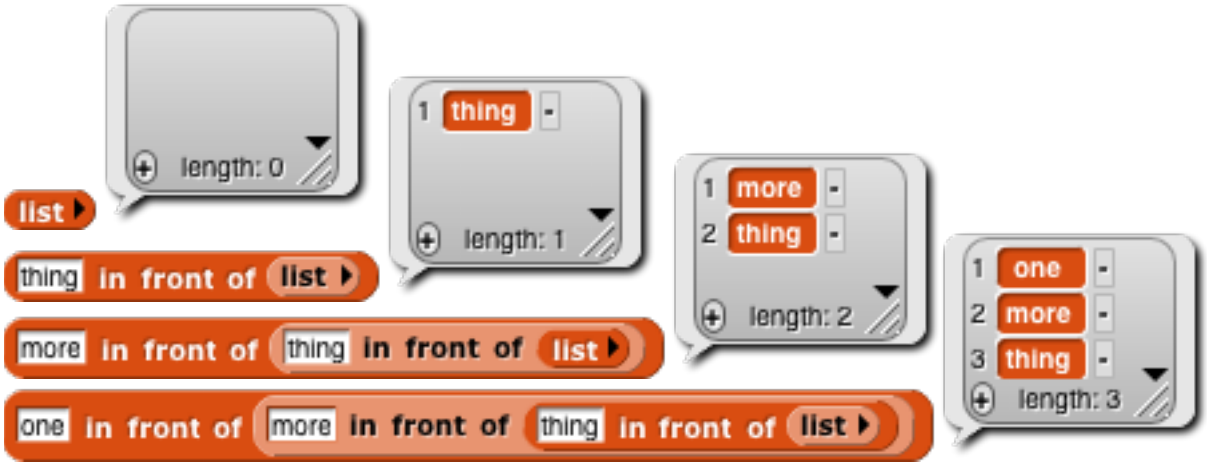
Linked Lists in Snap*!*

Jens Mönig
Mar. 16 2016

Lists in Snap are internally hybrid data structures; they can be used either as mutable dynamic arrays or as immutable linked lists. In a linked list each node only stores its first item and shares the memory of all following items with other nodes. Linked lists can accumulate data in linear time and are particularly suitable for processing through recursive functions.

In Snap an empty list is created by invoking the *LIST* reporter block. Items can be collected by creating a new list with a given first value that shares all of its following items with another list:



Lists constructed this way which are only ever accessed by side-effect free reporter blocks are internally represented as linked lists in Snap.

Mutating a list through command blocks converts its internal representation to a dynamic array.

Both modes coexist in Snap and novices need not even be aware of changing internal representations. For more advanced users, however, knowing about linked lists and sticking to a single mode can make lists more interesting and the scripts operating on them more effective.

Snap's hybrid lists have been masterminded by my friend and collaborator Brian Harvey, they have been a feature of Snap for many years, ever since the BYOB-rewrite. But only recently have they been fully integrated with Snap's list-watcher and table-view widgets and into Snap's project format.

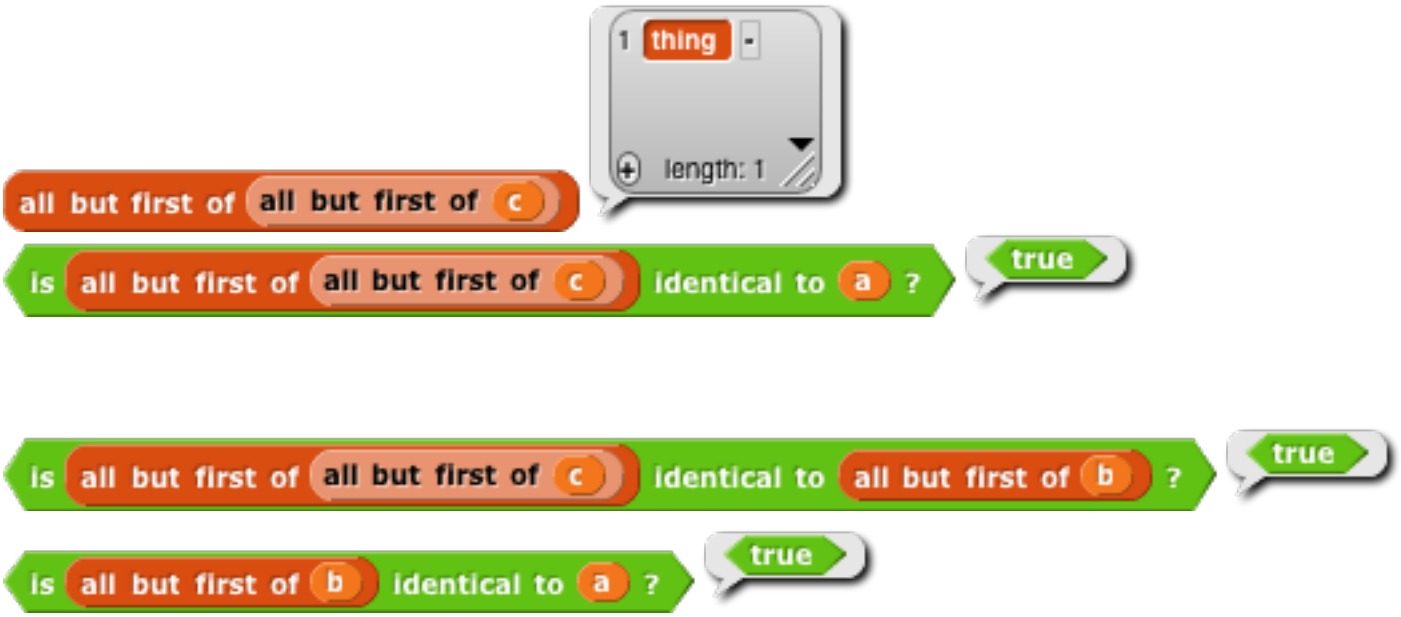
Partial Identity

The reporters

are able to operate in constant time regardless how long *"a linked list"* is, because none of these functions require enumerating or copying anything.

The following script creates a list (named *"c"*) with three elements and keeps track of the intermediate results (lists *"a"* and *"b"*):

Snap's identity testing block reveals that these partial lists are not just holding equal values, but are, in fact, identical to the sublists created by *"all but first of"*:



Saving

Saving or exporting a project in Snap also automatically saves all the project state with the exception of very few transient settings (such as whether a sprite's pen is "down"), including the values of all global and sprite-local variables. This also applies to variables holding lists, and, as of v4.0.6 also linked lists.

Opening a project with saved linked list data again automatically restores them, however, by default any variables referring to partial lists are turned into separate objects with their own identity, i.e. into copies. After saving and loading the project the identity tests shown in the example above will indicate that lists *a, b* and *c* are no longer partially identical.

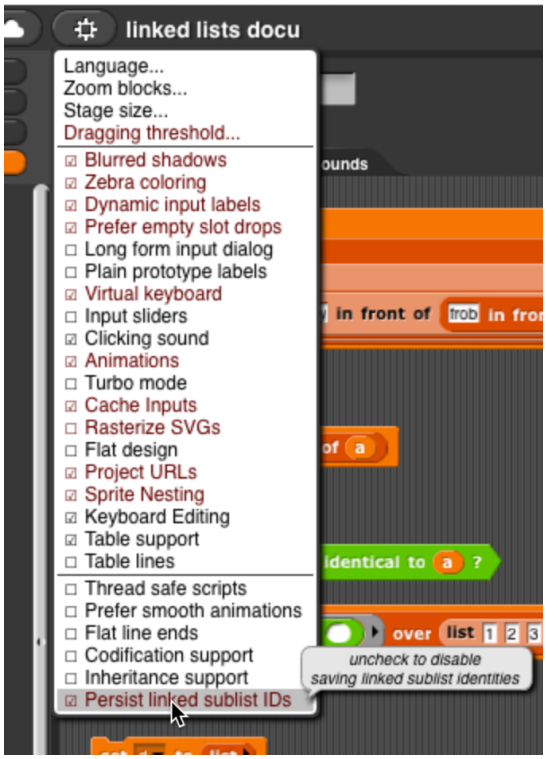
Each stored linked list gets restored as true linked list, albeit with its own identity, guaranteeing the same constant-time behavior as before, and since the functional programming paradigm generally discourages mutating lists no changes in the behavior of the project are to be expected.

```
▼<variables>
  ▼<variable name="a">
    ▼<list linked="linked" id="152">
      ▼<item>
          <l>thing</l>
        </item>
      </list>
    </variable>
  ▼<variable name="b">
    ▼<list linked="linked" id="153">
      ▼<item>
          <l>more</l>
        </item>
      ▼<item>
          <l>thing</l>
        </item>
      </list>
    </variable>
  ▼<variable name="c">
    ▼<list linked="linked" id="154">
      ▼<item>
          <l>one</l>
        </item>
      ▼<item>
          <l>more</l>
        </item>
      ▼<item>
          <l>thing</l>
        </item>
      </list>
    </variable>
</variables>
```

Persistent Sublist Identities

Note: *This is an experimental "hidden" feature that currently does not work on lists larger than about a thousand items. It is turned off by default whenever Snap starts or whenever a new project is created.*

You can turn on an experimental setting that tells Snap to preserve the unique identity of every node in every linked list in a project. To enable it press and hold the <shift> key while clicking on the settings menu indicated by the gear-button and then select the red menu item labelled "Persist linked sublist IDs":



When this - experimental! - setting is turned on saving and loading the example project with the three partially identical lists names *a, b* and *c* will preserve the identity of each as well as the shared partial identity among them.

```
▼<variables>
 ▼<variable name="a">
  ▼<list linked="linked" id="152">
   ▼<item>
      <l>thing</l>
    </item>
    <list id="153"/>
  </list>
 </variable>
 ▼<variable name="b">
  ▼<list linked="linked" id="154">
   ▼<item>
      <l>more</l>
    </item>
    <ref id="152"/>
  </list>
 </variable>
 ▼<variable name="c">
  ▼<list linked="linked" id="155">
   ▼<item>
      <l>one</l>
    </item>
    <ref id="154"/>
  </list>
 </variable>
</variables>
```

The reason why this setting is not (yet) the default one is that serializing linked lists is currently done recursively in JavaScript, and that for lists larger than roughly a thousand items the recursion level exceeds JavaScripts maximum call stack size:

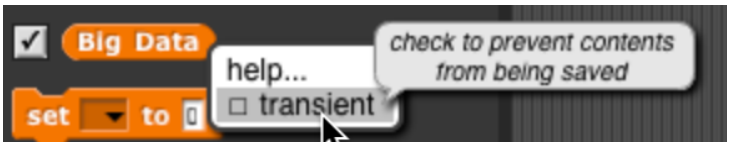
If you get this error message persisting linked sublist identities is not yet possible, until either JavaScript gets proper tail recursion or - more likely - linked lists are serialized iteratively or using Snap itself.

In this case make sure to disable the experimental setting again before saving or exporting the project. This error can occur whenever the project is serialized an deserialized, i.e. also when zooming blocks or switching to another translation language.

Transient Variables

Always saving the project state - including the contents of every variable - in the project file is very useful, because it lets you continue working on it at pretty much exactly the same state next time you open it again. However, if a project accumulates or crunches large amounts of data which it either records itself or retrieves from outside sources, e.g. from a remote web service, serializing and saving massive data sets inside the project file can be a burden, especially if these are temporary or intermediate results that won't even be processed anymore. Such data can be considered "transient" and need not be saved. Examples are weather reports obtained from a website or internally recorded mouse-trail coordinates used for a paint project.

Since v.4.0.6 you can now mark global variables and sprite-local variables, i.e. any variable that is created using the *"Make a variable"* button and shown in the palette, as "transient":



By default all variables are persistent when they are first created. You can specify for each variable in palette individually whether its contents shall be serialized and saved in the project file via its context menu. Note that this choice can only be made inside the palette.

Transient variables are saved as "named slots" in the project. The next time you open the project the variable will still exist, but its contents will have been reset.

Enjoy!
-Jens