

Lists and Files for Scratch

a Prototype

by Jens Mönig

April/May 2008

Overview

This documents proposes introducing a new, arrayed data type to MIT-Scratch, enhancing Scratch to store and retrieve such arrayed data to and from external files, and enabling Scratch projects using such file-access to be compiled into stand-alone executables under Windows whereby retaining any file references in their code, so to make them easily shareable among users together with any external files they maintain.

The prototype for this proposal is based on and derived from the official Scratch Source Code version 1.2.1 from Jan 6th 2008 by the MIT Media Lab.

List Behaviour

I am proposing an indexed collection of numerical values which can be indefinitely and randomly accessed without ever having to be dimensioned. Indices start at 1. Querying an index outside of the list's current range (e.g. negative indices) does not raise an error but returns zero per default. Setting a value at an index outside of the current range automatically re-dimensions the list if the index is a positive non-zero integer. Setting the value at an index to zero redimensions the list, if the index is currently the highest one in the list.

List Implementation

Lists are implemented internally as wrappers around a Smalltalk Dictionary, the indices corresponding to keys in the Dictionary. Setting an index to zero removes the key in the Dictionary if it exists. Querying a list's size answers the largest key in the Dictionary. Smalltalk primitives requiring list-iteration thus need only to iterate through the Dictionary's keys instead of through every (virtual) index of the list, greatly speeding up such operations.

Package

The prototype is packaged and distributed as a single zip-archieve. To run it under Windows, unzip the archive into a new folder. Two program files will appear:



To run these programs under Windows double click on the icons. The IDE prototype should also run under Mac (and Linux), if you use the *.image file in the "bin" subfolder together with a stock Squeak VM for the particular OS.

The IDE



Lists-IDE

This tiny program is just a convenient launcher for the Scratch/Squeak application in the "bin" subfolder. The only thing it does is bring up the actual application.

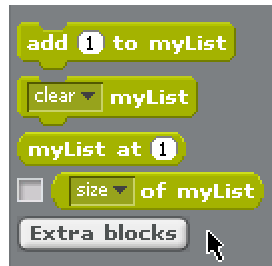
The IDE has two additional buttons in the variables category: Add a list / Remove a list.



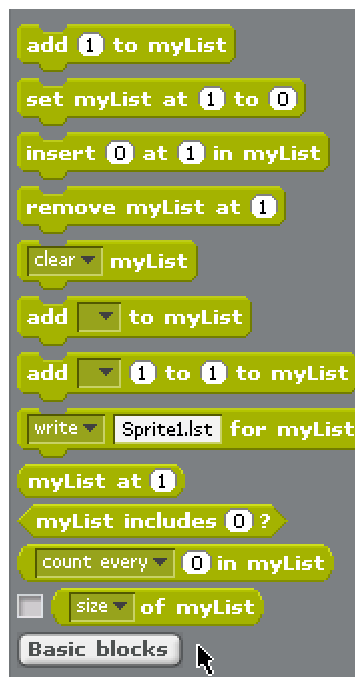
Choosing "Make a list" brings up a similar dialog to the one already used for variables, allowing for either global oder local lists:



Defining a new list will create a corresponding set of command blocks in the variables panes. Per default, only the most basic operations are displayed:



Below the set of blocks for each list is a button letting the user expand / collapse additional "extra" blocks:



The user's choice of collapsed/expanded list-blocks is remembered by the IDE, so the user does not have to customize his preferences each time a different sprite is edited.

The Basic Blocks



Add-block: This block appends the argument number to the end of the specified list.

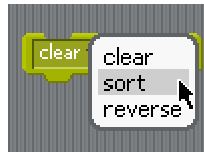
It is actually the same as:





Rearrange-block: This block rearranges the specified list. Selecting "clear" removes all values from the list, leaving it empty.

A drop-down list lets the user select other operations:

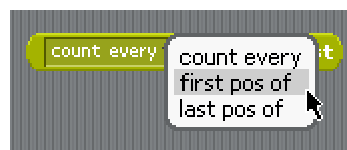


sort rearranges the values in the list in ascending order.

reverse rearranges the values in the list backwards.



Reporter-block: This block reports numerical data from and about the list. It features the following options, which again can be chosen via a drop-down list:



In the following example the reporter block is used in two ways to remove every instance of the number 7 from myList:

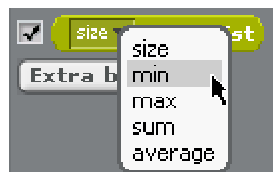




Stats-block: This block reports statistical numerical data about the list. Any of its operations can be toggled as a watcher onstage by marking the checkbox on its left side:



The drop-down list of this stats-block features the following operations:



size - reports the highest index for a non-zero value in the list.

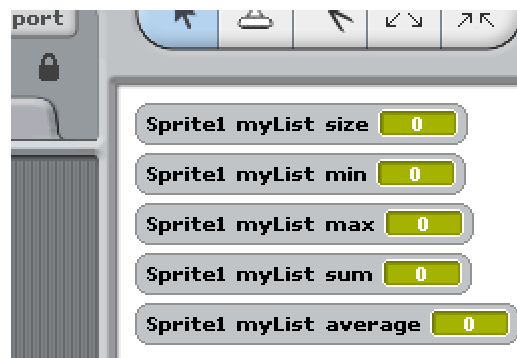
min - reports the minimum value stored in the list.

max - reports the maximum value stored in the list.

sum - reports all values of the list added together.

average - reports the average of the list's values (same as sum/size).

In order to save screen-real-estate each of these operations can be toggled into a separate watcher onstage, depending on which operation the user selects:



The Extra Blocks



A Scratch 'setter' block with a green background. The text reads 'set myList at 1 to 0'. The number '1' is in a red box and '0' is in a white box.

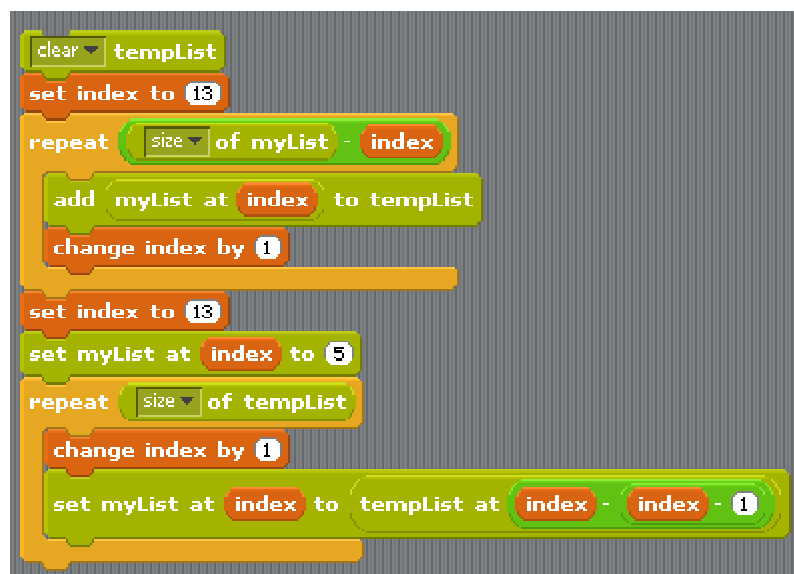
Setter-block: This block is the low-level setter block, allowing random setting of indexed numerical data. Lists do not have to be dimensioned and do not have to be accessed sequentially.

Internally I have implemented lists as Smalltalk Dictionaries, so my Smalltalk-primitives need only iterate over the indices actually used ignoring all empty slots in between.



A Scratch 'inserter' block with a green background. The text reads 'insert 5 at 13 in myList'. The number '5' is in a red box and '13' is in a white box.

Inserter-block: This block moves every value following the specified index one slot towards the end of the list and then assigns the specified value to that slot. Its functionality corresponds with the following example of how this operation would otherwise have to be coded:



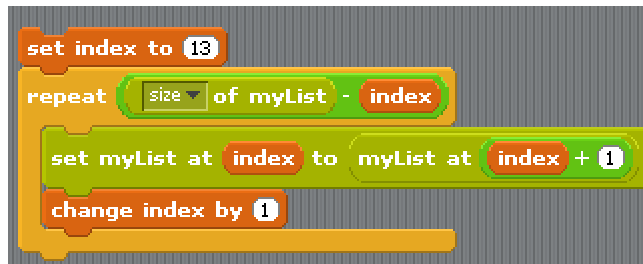
```
clear tempList
set index to 13
repeat (size of myList - index)
  add myList at index to tempList
  change index by 1
set index to 13
set myList at index to 5
repeat (size of tempList)
  change index by 1
  set myList at index to tempList at (index - index - 1)
```

The image shows a Scratch code snippet on a grey background. It starts with 'clear tempList', followed by 'set index to 13'. A 'repeat' block with 'size of myList - index' contains 'add myList at index to tempList' and 'change index by 1'. Then 'set index to 13' and 'set myList at index to 5'. Another 'repeat' block with 'size of tempList' contains 'change index by 1' and 'set myList at index to tempList at (index - index - 1)'.



Remover-block: This block removes the value stored at the specified index of the list and moves all following values one slot towards the beginning of the list.

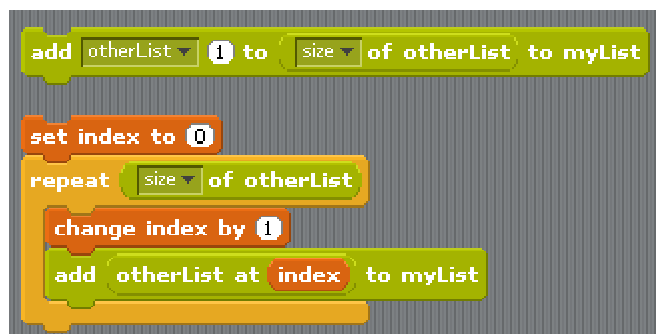
It corresponds to this example:



Copy-block: This block copies all values of *another* (source) list and adds them to the specified (target) list. The source list can be chosen via a drop-down list:



Its functionality corresponds to both of the following two examples:



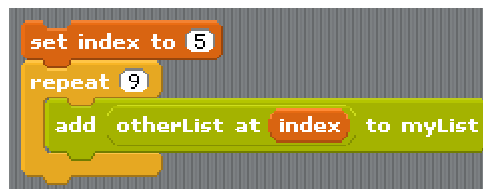


Range-block: This block copies every value within the specified index-range of the target list and appends it to the source list.

The source list can be selected via a drop-down menu:



This block corresponds to the following example:

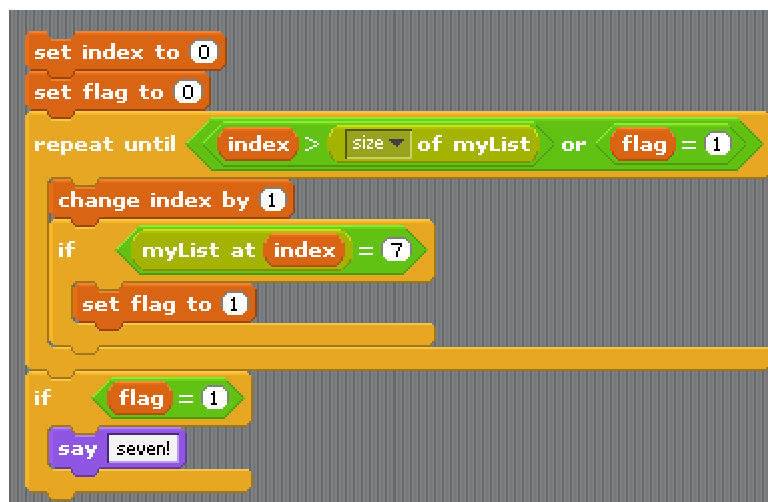


Tester-block: This Boolean block answers if a specified value is stored in the given list.

The following example



would otherwise have to be coded like this:



File access

persistant data storage is implemented "on top of" the list design:



File-block: This blocks writes and reads the given list to and from a user-specified file in the same directory as the project code.



This way, a project can be easily shared together with any files it maintains.

The filedir can also be specified in the command line using the new filedir:pathname option. This option is exploited by the fake compiler as will be shown later.

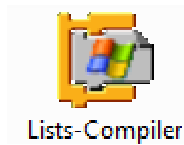
The range-block and the reporter-block can be especially useful to parse several sub-lists from a single file (or to simulate multi-dimensional lists), delimited by a certain numerical value:



Compatibility

The prototype can read/import and write compatible Scratch projects as well as projects containing lists. If a project contains a list it is marked internally as being incompatible with Scratch and will produce a "bad header" error when opened in Scratch.

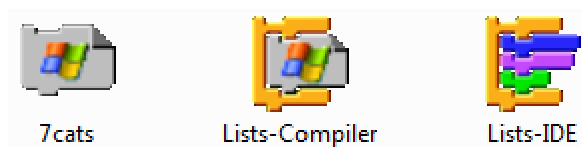
The Compiler



The fake compiler packages the project together with the prototype's vm, image and plug-ins into a sort of self-extracting archive (exe). The resulting exe extracts all of its internal files into a temporary folder and calls the vm with the image and the project file in presentation mode, specifying the current folder of the exe as the directory for any external list files. Once the user exits presentation mode the program exits directly back to the OS and removes all temporary files, but leaves any list-files, thus enabling data-persistence.

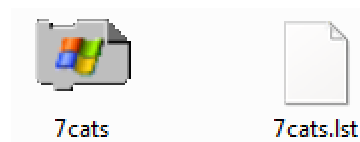
Running the compiler brings up a dialog box letting the user select any Scratch project for compilation, and another one letting the user optionally specify an icon.

The output exe-file will be written to the directory containing the other launch-executables:



It can be shared as stand-alone exe, without Scratch or the prototype.

Running the exe will play the project in presentation mode and exit back to Windows without bringing up the IDE.



Any files created or maintained by the compiled project will be written/read to and from the same folder the exe file is in.

Since the prototype is 100% backwards compatible with Scratch the compiler can also compile any Scratch project.