

New Ways of Working with Blocks in Snap!

Jens Mönig
Jan. 15 2017

Snap's first update in 2017 concludes a series of improvements and enhancements to the user interface. While previous releases in 2016 focused on data and its visualization (tables, linked lists and first-class sprites) v4.0.10 revisits how you work with blocks, adding such long sought-after features as UNDO/REDO, auto-wrapping (inside-out nesting of control structures), entering formulas using the keyboard, refactoring variables (renaming all accessors in scope), browsing and previewing libraries, supporting hierarchical drop-down menus for input slots and extending the typography for block labels. In addition, this release also improves the UI for working with resources (pictures, SVGs, sounds) and projects.

Auto-Wrapping

Traditionally you can add command blocks anywhere in a script, either by attaching them at the bottom or top of the stack, or by *inserting* them *in between* two commands. As you drag a command block over a script, a horizontal feedback-halo indicates the position at which the block will be *inserted* once it is dropped:



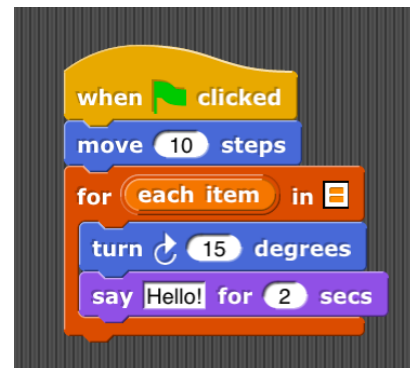
In addition to *inserting* blocks into a script the new version also lets you *wrap* control structures (i.e. C- and E-shaped blocks) *around* other commands or stacks of commands. As you drag a control structure over a script the drop-feedback alternates between a white insertion indicator and a block-colored wrapping-indicator:



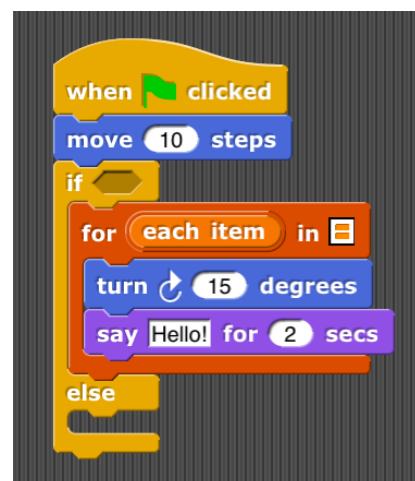
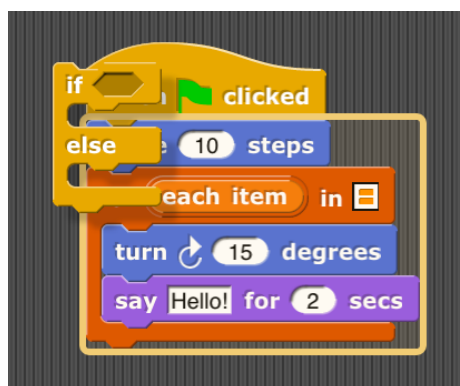
dropping the dragged block or script at a position that indicates auto-wrapping inserts those commands that are surrounded by the lasso into the C-shaped input slot. This lets you refine existing scripts with additional control structures more easily in an inside-out fashion:



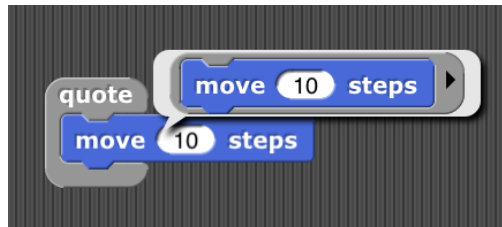
Auto-wrapping works for any block with a C-shaped input slot, including custom blocks your build yourself or control structures you import from libraries, such as WHILE or FOR loops:



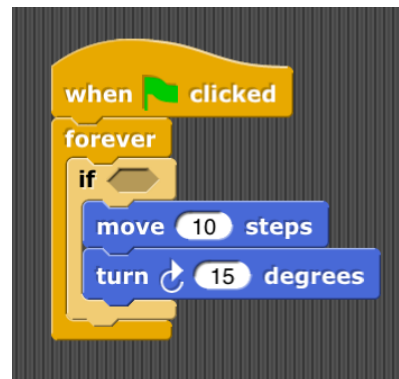
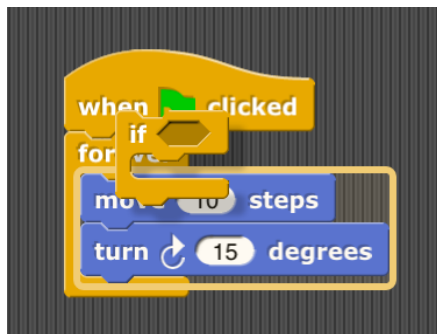
If a block has several C-shaped input slots, only the first one auto-wraps, and only if it is empty:



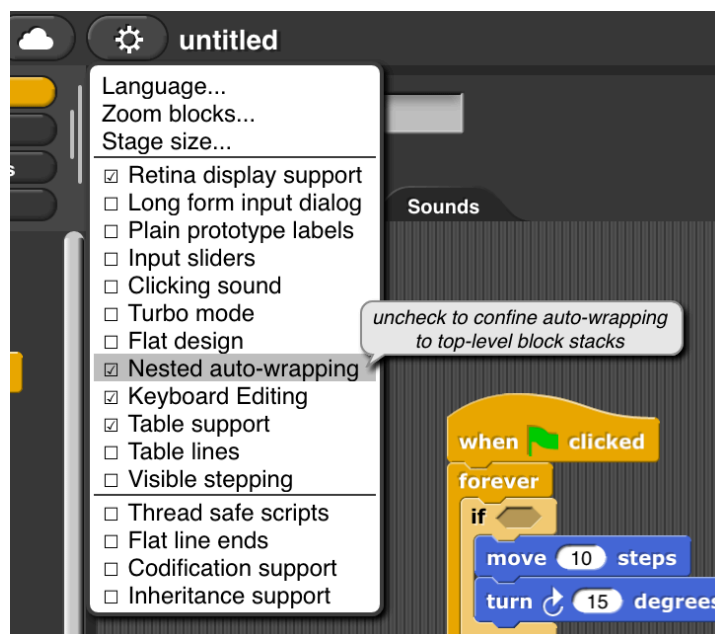
Snap! also lets you build your own C-shaped reporters (how cool is that, eh?!). Note, however, that auto-wrapping is only supported for commands, otherwise command-scripts would too easily break apart:



Auto-wrapping in Snap! is not limited to top-level stacks. You can also wrap a C-shaped statement around an already nested inner script:

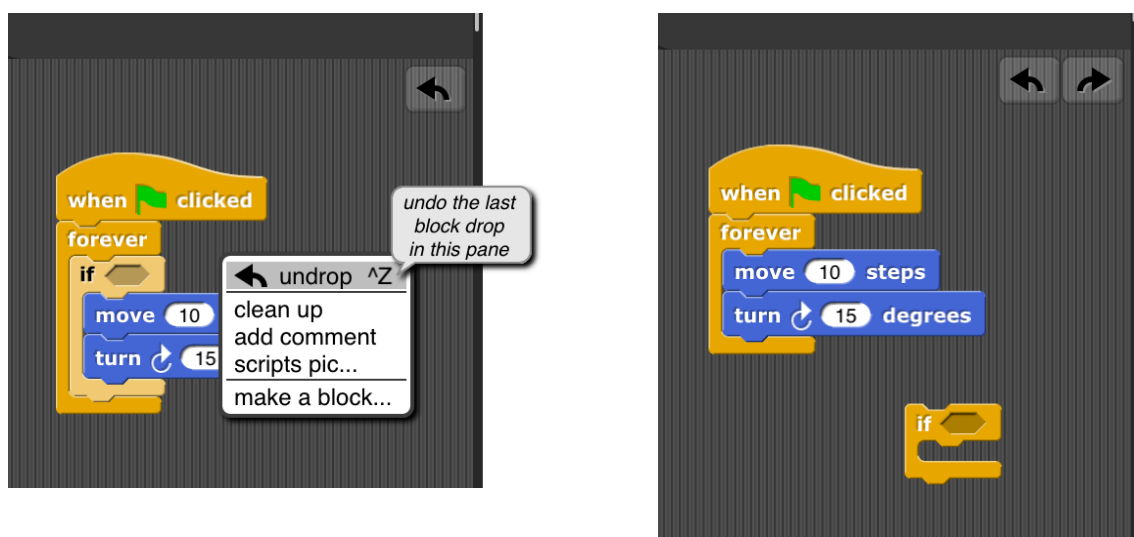


You can turn off the ability to auto-wrap around nested inner scripts in the settings menu. This can be helpful when working with beginners who are not yet used to dragging & dropping gestures or motorically impaired persons:

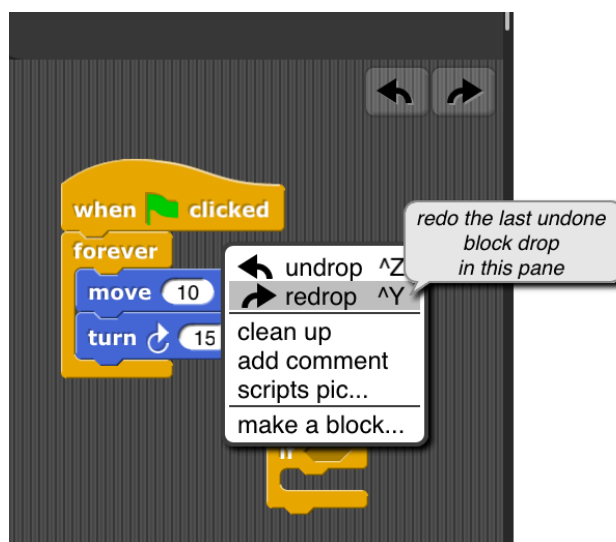


Undrop / Redrop / Undelete

Previously you could “undrop” a block from where it was last placed. This would pick up the block and restore the previous situation to how it was before the block was dropped. In the new v4.0.10 “undrop” has been extended to reach back infinite levels. Instead of picking up the last drop and placing it into the user’s “hand”, the block slides back to from it was picked up. This is essentially the same as a traditional “undo” feature, except it currently only applies to blocks (not to sprites, costumes, sounds etc.). In the future we will extend this mechanism until it reaches full “undo” functionality. The availability of “undrop” is indicated by a new “back-arrow” button in the scripting area. Additionally, “undrop” is available via the scripting area’s context menu, along with the standard keyboard shortcut:



Once the first “undroppable” action has occurred, a forward-arrow indicates that the block can be “redropped” to from where it was “undropped” before:

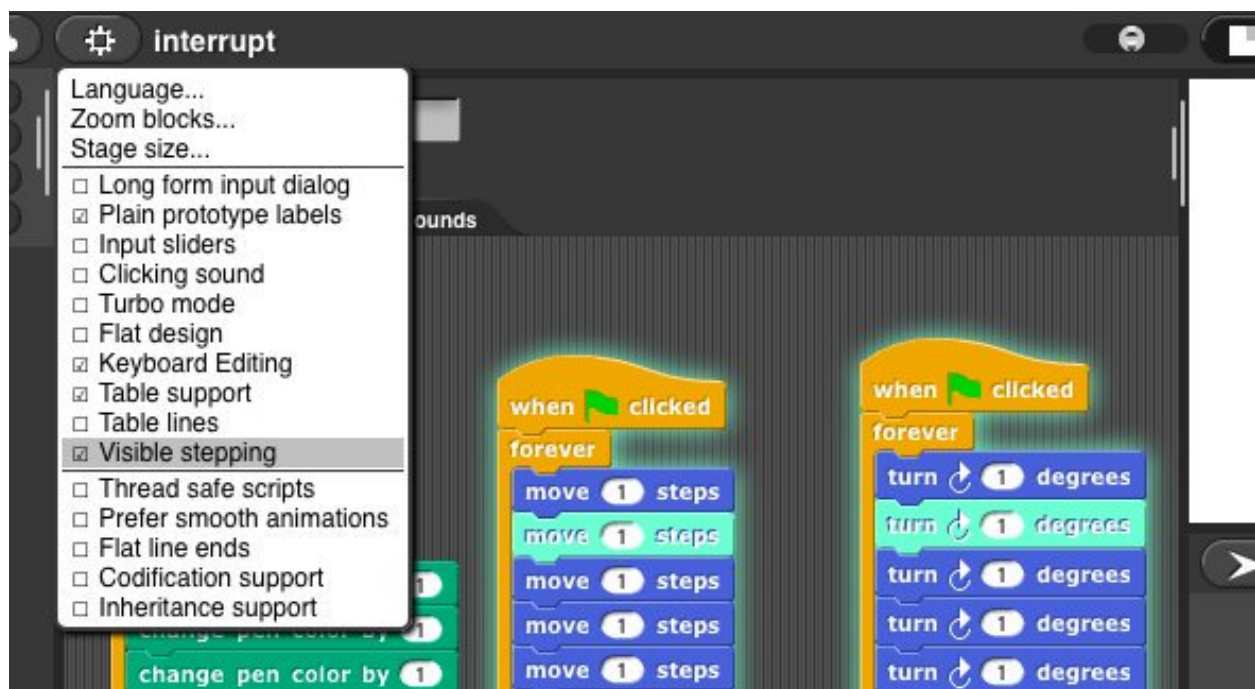


In addition to sliding blocks back and forth the “undrop/redrop” feature also brings back blocks and scripts that have been deleted by the user, either by having been dropped onto the palette or otherwise deleted via the context menu.

Note: Instead of a single global UNDO-queue Snap’s “undrop” memory is contextual, i.e. each scripting pane keeps track of its own history. This way you can work on the scripts of several different sprites and go back and forth per sprite. The same applies to custom block editor instances.

Visible Stepping

Part of working with blocks that makes Snap! interactive is that running scripts are always “live”. Out of this comes Snap’s capability to visualize which blocks and inputs are currently executed by highlighting the syntax elements as they get activated. This has already been released in v4.0.9, but so far not mentioned in any documentation. Therefore I’m including a short description of the “visible stepping” feature here:



Once “Visible Stepping” is turned on, a slider appears in the IDE’s toolbar that lets you slow down the execution speed. If that slider is moved all the way to the left the execution is paused altogether, and the “pause” button turns into a “next step” button that lets you manually step until you either again move the slider to a faster stepping-rate or again turn off visible stepping.

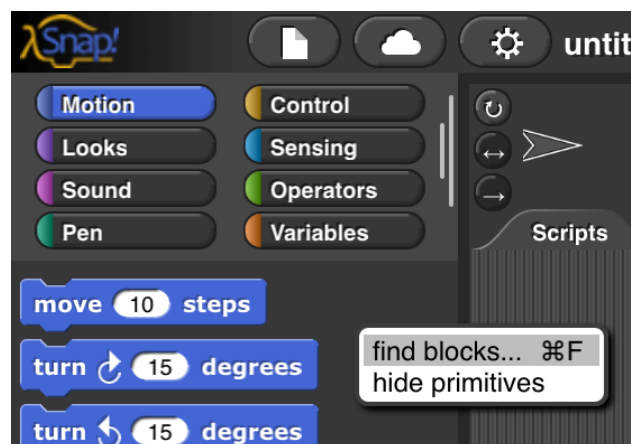
Note that while execution is paused the system is still “live” and interactive, and you can inspect and change data and scripts.

Also note that only those scripts that are visible will step. Custom block definition will only step if you open the editor (after turning on visible stepping) and stop stepping once you close the block editor again. This gives you a very fine-grained control over what you'd like to track.

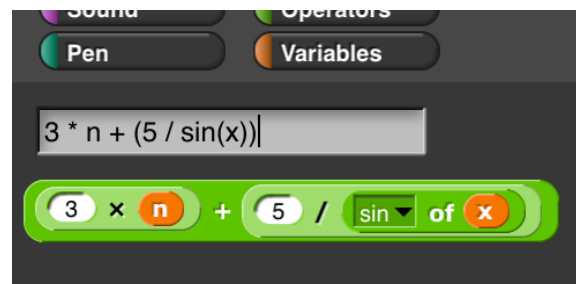
Any block editor that is already open before visible stepping is turned on needs to be re-opened in order to let it “step” visibly.

Entering Formulas on the Keyboard

Snap! V4.0.10 lets you enter simple arithmetic expressions in classic infix notation from left to right and “inside-out” using the keyboard. This mechanism is now a feature of Snap's “search for blocks” mode:



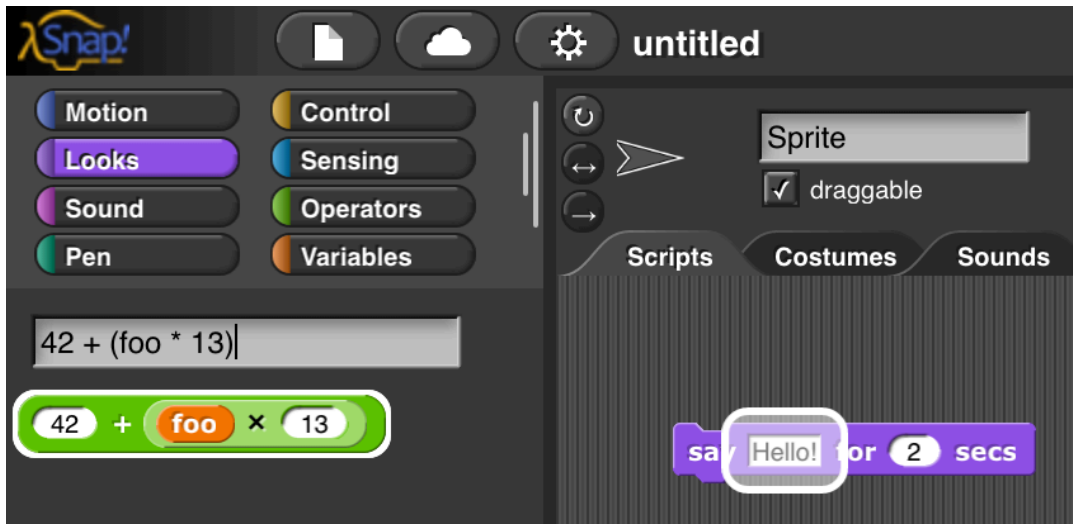
As you type an arithmetic expression into Snap's search-field type it is turned into a - nested - block that you can drag out of the palette and into your scripts:



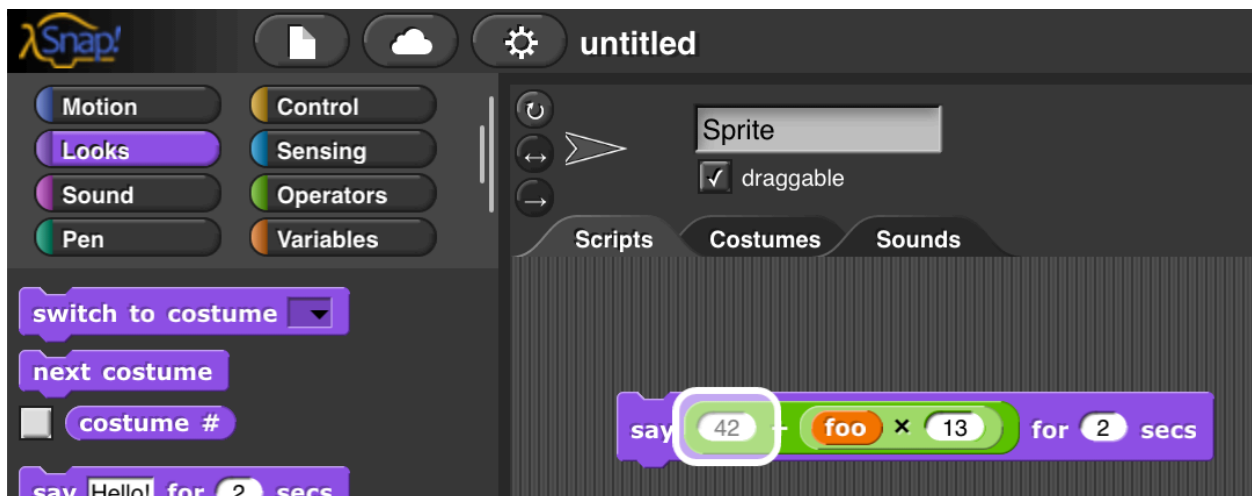
Note that there is no operator precedence, i.e. expressions are parsed strictly left-to-right, but you can use parentheses to specify the order of operations inside your expression.

Also note that you can also use Boolean operators to create predicate expressions.

Entering expressions using the keyboard can be especially powerful when you're already using Snap's keyboard-entry mode for blocks:

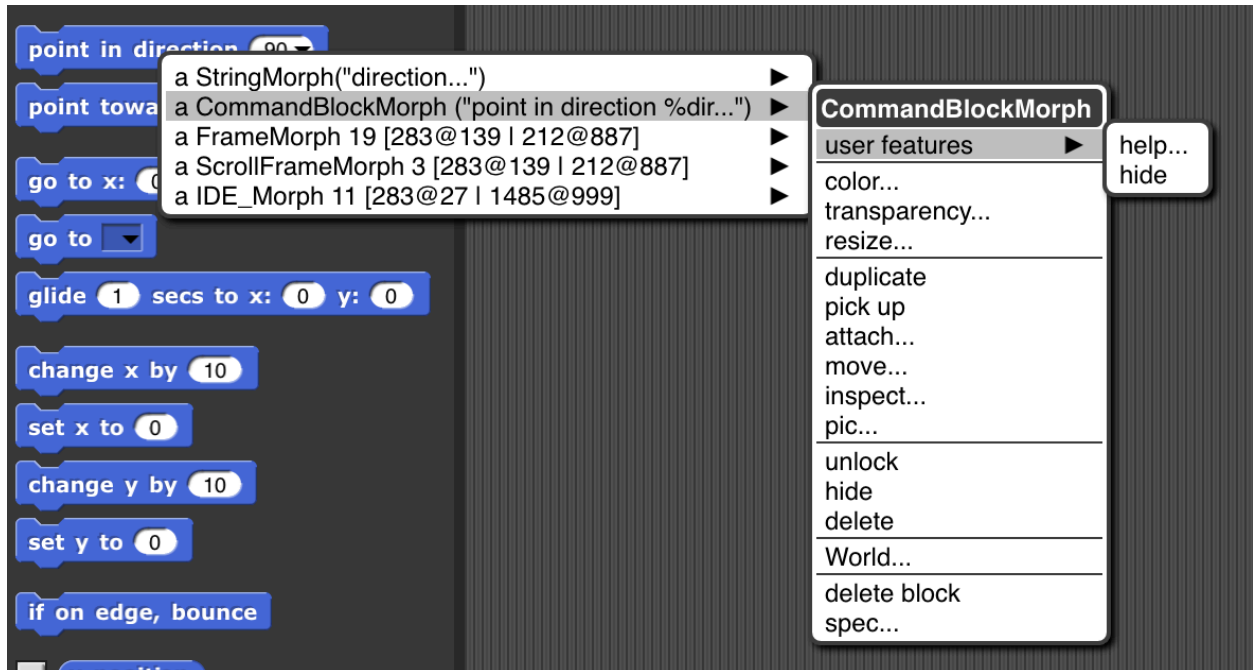


As you edit an input you can start typing a formula right away to assemble an expression of blocks, and when you're done simply press <enter> to insert it into the selected slot:

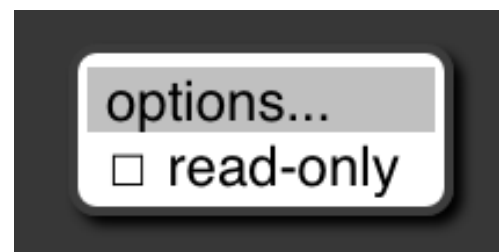
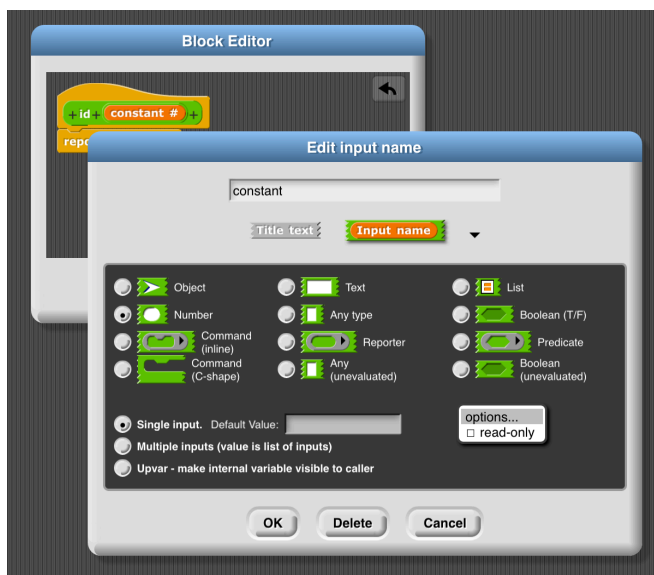


Hierarchical Menus

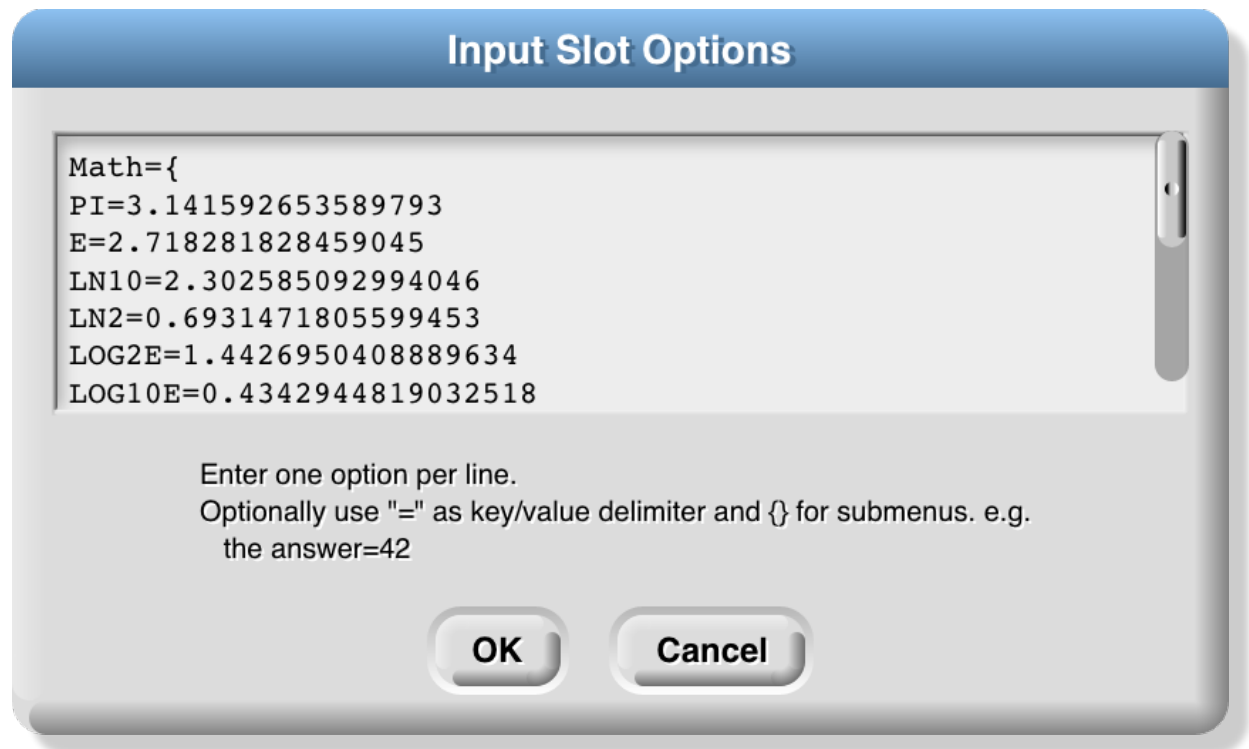
Snap's underlying Morphic kernel has been extended for v4.0.10 to support hierarchical menus. At the Morphic "developer mode" level these are now used when exploring Morphic hierarchies:



You can build your own - hierarchical - drop-down menus for input slots in Snap's - somewhat hidden - "input slot options" interface:



In the “input slot options” editor you can use curly parentheses to specify submenus:



The example code above (also shown complete below creates a menu with two submenus:

```
Math={
PI=3.141592653589793
E=2.718281828459045
LN10=2.302585092994046
LN2=0.6931471805599453
LOG2E=1.4426950408889634
LOG10E=0.4342944819032518
}
Directions={
North=0
East=90
South=180
West=-90
}
```

Here's the complete example, showing what the hierarchical drop-down menu defined in the "input slot options" editor ends up looking like:

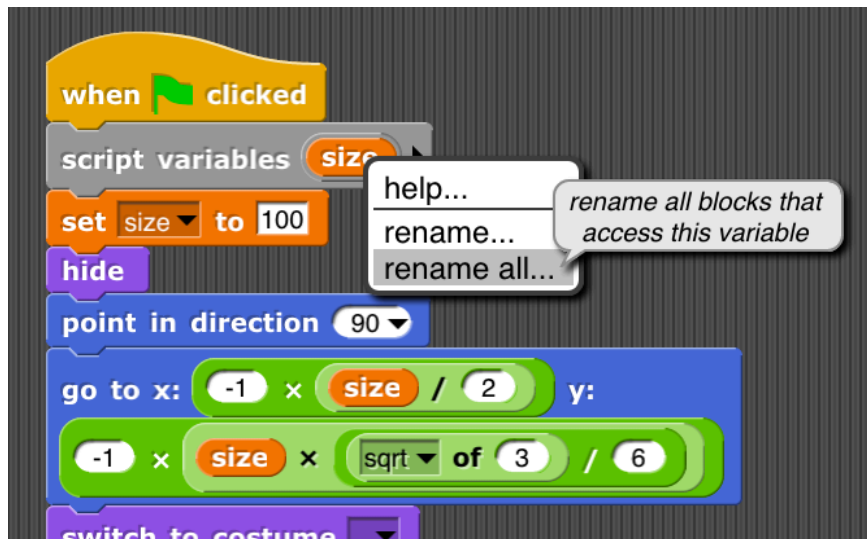


Hierarchical menus have been pioneered by Brian Harvey.

Refactoring Variables

A powerful new tool is the capability to bulk-rename all references to a variable. This is available in the context menu of every variable declaration, both local and global, including sprite-only variables:

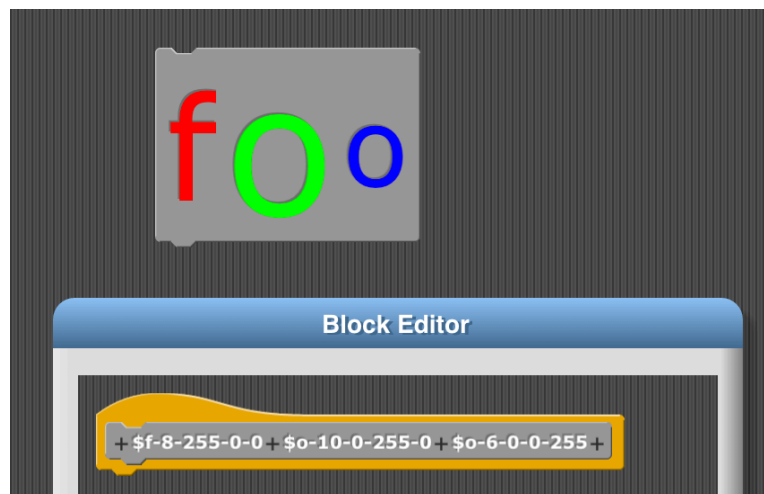
Choosing to “rename all” will rename all blocks that reference the variable within its scope.



Refactoring variables has been contributed by Bernat Romagosa, as has been:

Typography for Block Labels

You can now apply basic typography (relative size and color) to any part of a custom block label:



Browsing and Previewing Libraries

Another powerful tool is Snap's new browser that lets you explore and preview libraries of blocks-modules that you might want to include in your project:



The library browser has been contributed by Michael Ball.

Finding Projects

Bernat Romagosa also contributed a nifty search-field in Snap's project dialog that lets you quickly and easily find a particular project in your cloud-account or on the browser's local storage by entering a search phrase. You can use parts of the project's title or of its project notes:



SVG Costumes

Snap v4.0.10 enhances support for SVG costumes and now includes many of Scratch's costumes in its costumes browser:



Browsing Sounds

Also, sounds can now be imported using the same media import browser:



Summary: Release Notes

Snap! Build Your Own Blocks v4.0.10

Features:

- auto-wrapping of C-slots
- undo / redo for blocks, unlimited, but has some issues
- search field for projects, thanks, Bernat!!
- basic typography support for custom block labels, thanks, Bernat!!
- treat JS-function reporters the same as variable getters wrt rings
- new url switch #dl: for downloading raw shared projects
- new url option switch: &noExitWarning
- svg support for images from the web (svg files have been supported for a long time)
- use media dialog for browsing and importing sounds
- highly experimental infix-expression-to-reporter parser. Thanks, Bernat, for the brilliant idea to add it to the search-blocks field!
- hierarchical menus, also for custom blocks, thanks, Brian!
- variable refactoring, thanks, Bernat!
- “#run:” flags (same as “#present:”): ‘editMode’, ‘noRun’, ‘hideControls’, thanks, Brian!
- Libraries Browser, thanks, Michael!

Fixes:

- Music (play note) to work again in new and recent browser versions (Chrome 55)
- IDE layout: fixed resizing issues when the window becomes too small
- Keep left-over blocks from “relabel” around
- Evaluate the generic WHEN-hat block’s predicate and first step of the attached script in the same atom
- “go back _ layers” to work with out-of bounds numbers, thanks, Brian Broll!
- Translation updates (Russian, Polish, Danish, Portuguese, Catalan, German)

Enjoy!

-Jens