KnobScripter v3.0
By Adrian Pueyo

USER GUIDE

Thank you for downloading **KnobScripter v3 (KS3)**, a free and open-source complete python script editor for Nuke**.** I hope it will help improve your scripting experience.

Tutorial video: *https://vimeo.com/adrianpueyo/knobscripter3*

Nukepedia: **http://www.nukepedia.com/python/ui/knobscripter**

GitHub repo: **https://github.com/adrianpueyo/KnobScripter**

1. Introduction

> ***KnobScripter v3.0*** *(or KS3) is a full script editor for Nuke that can script python on .py files and knobs as well as BlinkScript, with all the functionality from the default script editor in Nuke plus syntax helpers, predictions, snippets and other handy features.*

KnobScripter started in 2016 as a little python scripting widget for Nuke, and an exercise for me to learn PySide. Its goal was to provide simple **syntax highlighting**, line **numbers** and auto-indenting, and the ability to script on node **knobs** directly. In the following years I expanded its functionality, found more and more cool ideas to implement, and it eventually turned into a fully functional script editor which I released as ***KnobScripter 2***.

KS3 is the next major step for this tool, and it features a greatly optimized code, Python 3 compatibility, BlinkScript mode, a Code Gallery and many other features and fixes.

2. Overview

KnobScripter was designed as a simple python script editor for Nuke, and its main goal is to let you **interact with Nuke's python API in a more pleasant way** than Nuke's default methods. It doesn't intend to replace a fully featured IDE, but it proves enough for most common python tasks and quick work with snippets, basically anything that needs quick interaction with Nuke's python console.

With this in mind, I focused its development in the aspects that I love about IDEs regarding efficiency (such as syntax helpers), in covering everything the default Script Editor in Nuke currently offers, and last but not least, in covering the python aspects of Nuke that are generally tedious, such as adding python or blink code to nodes/knobs.

Overall features

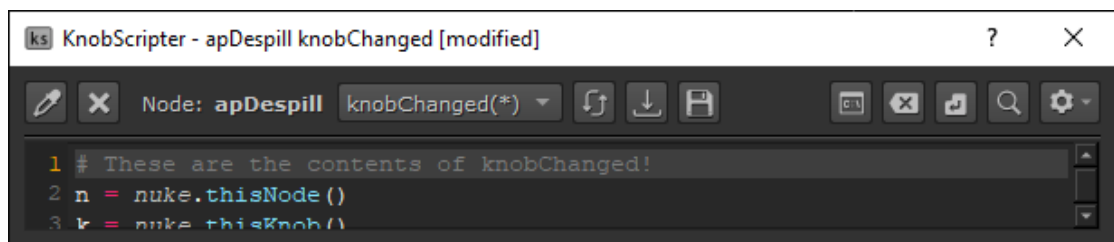Some of the general features of KnobScripter are the following:

- **Full scripting mode for .py files.**
  You can create, browse, modify or toggle between python files and folders.
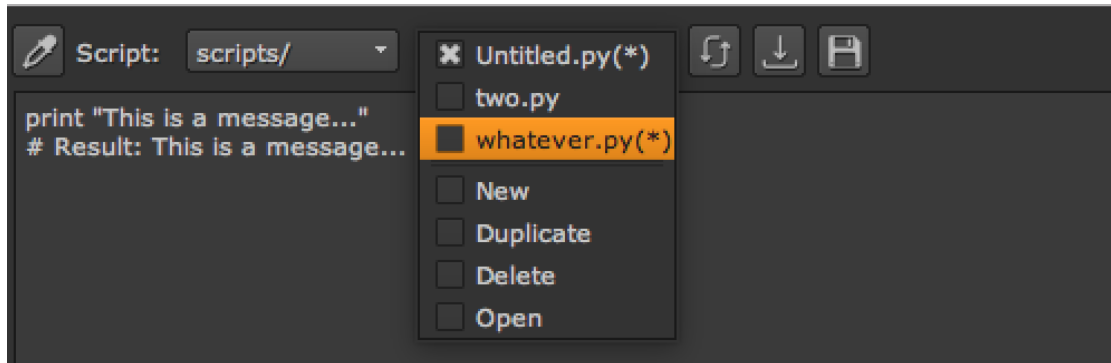
- **Node editing mode**
  In node mode, you can script directly on python buttons or callback knobs, as well as BlinkScript.
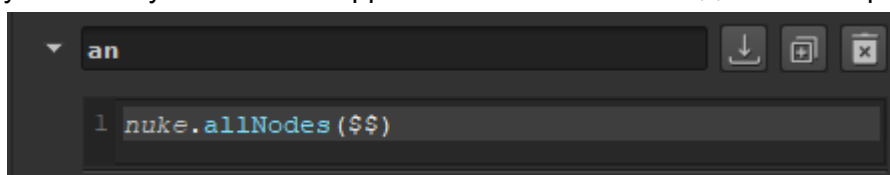

- **Python output console.**
  Same as the one from Nuke's default script editor, where you can execute any code and read the python output log.
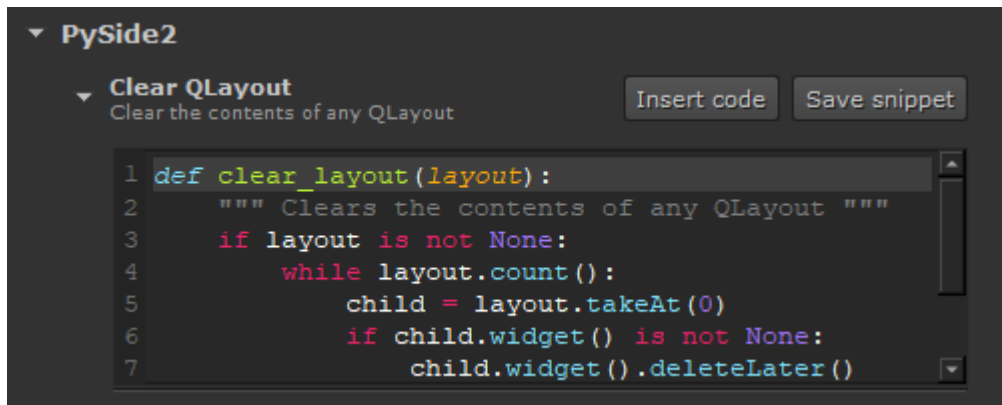
- **Find-Replace.**
  A proper find-replace widget as you'd expect in a python editor.

- **Snippets.**
  Snippets are short codes you can assign to longer pieces of code, so that by writing the short code and pressing tab you'll get the long code. You can also choose where you'll want your cursor to appear with the convenient $$ and other placeholders.

- **Python syntax highlighting, line numbers, auto-intending, auto-completer.**
  You can choose between two code highlighting styles (including one inspired by sublime's popular Monokai theme). Auto-completions on tab read from the available modules and the actual functions and variables in your script.

- **Code Gallery**
  The Code Gallery is a convenient place to store and browse codes that you might want to revisit many times.

- **Syntax helpers, multi-line commenting, moving/duplicating lines, and more!**
  Helps you open and close parenthesis, comments, etc. You can also duplicate, move or comment lines through key combinations, similar to normal IDEs.

- **Free and open source :)**
  KnobScripter started as a learning project for me and keeping it up-to-date is currently a big hobby, so I love hearing how you implemented it in your studio or adapted it to your needs. Forks, pull requests and suggestions/ideas on GitHub or Nukepedia are super welcome too.

3. Usage

Panel Types

In Nuke, you can open the *KnobScripter* both as a floating window or as a dockable pane.

- To open KnobScripter as a **floating window**, just press Alt+Z on the Node Graph.
- In order to bring the **dockable pane** you need to do the following:
  Right click on the pane selection bar, and go to `Windows/Custom/KnobScripter`
  Then, a **KnobScripter Pane** will be created. Now you can even save the workspace, so the KnobScripter will be created by default every time you open nuke.
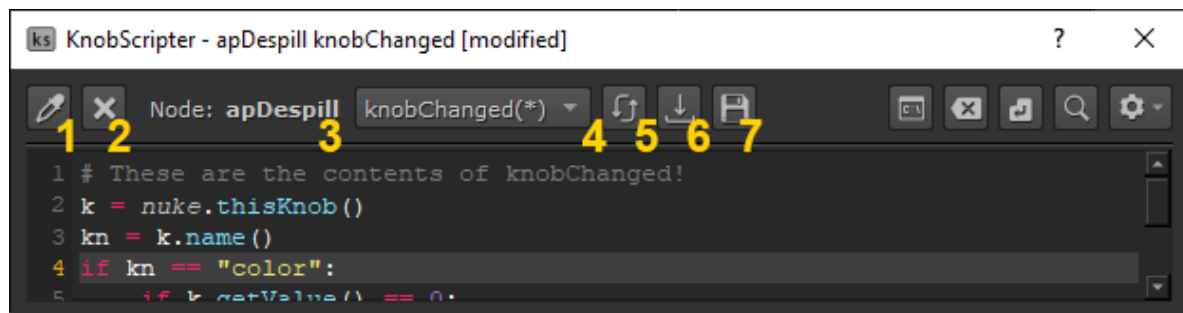
KnobScripter Modes

There are two main modes on the *KnobScripter*: Node Mode and Script Mode. There's an additional Blink mode which technically belongs to the node mode but it deserves a separate part in this documentation, as its buttons and syntax tools are completely different.

Node Mode

Lets you script on python buttons, python custom or callback knobs, on any node.

To enter the node mode, click on the **node picker** [icon] or simply open a floating *KnobScripter* (alt+Z) with a node selected. Then, you'll see the following:

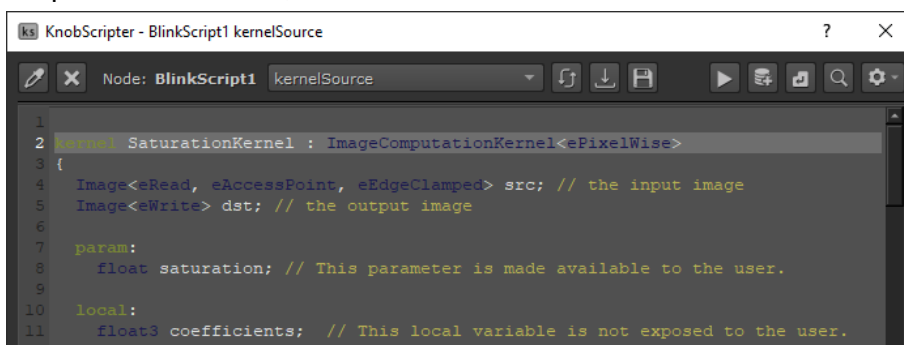|	
1	Node picker. To select a different node. If no node is selected, it lets you type the name of any node in your script. Including `root` and `preferences`.
2	Close the node mode.
3	Currently selected node, in which we are scripting.
4	Dropdown of available knobs on the node (python and callback knobs). You can display knob labels and names, or only names (change in Prefs).
5	Refresh the dropdown.
6	Reload the saved state of the knob. Will revert any changes on the editor.
7	Save the editor contents into the knob. Till you save, nothing changes.

Script Mode

Lets you script on .py files, that you can create, modify, duplicate, and you can even create different folders for them. Additionally, it lets you point to any folder in your computer.
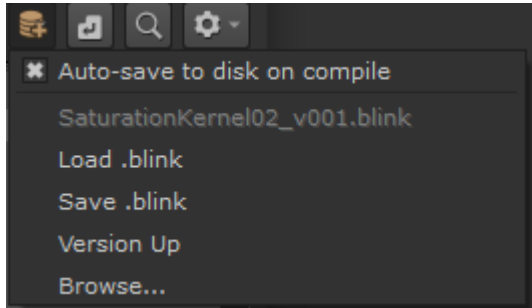
1	Node picker. To select a node, and enter the node mode.
2	Dropdown with available folders as well as custom added ones (like `.nuke`) As well as: `New`, `Open` (browse), and `Add custom` path.
3	Dropdown with all .py scripts in the chosen folder. (`Untitled.py` selected). As well as: `New`, `Duplicate`, `Delete`, and `Open` with external app.
4	Refresh the dropdowns.
5	Reload the save state of the script. Will revert any changes on the editor.
6	Save the script into the .py file. A `.py.autosave` will be created otherwise.
7	Execute the selected code (or all code if no selection) on the output panel.
8	Clear the output panel (Top part, with the lighter grey background).
9	Open the Code Gallery Panel

Blink Mode

If you select a BlinkScript node, enter node mode on the KnobScripter and then go to the kernelSource knob, you will see that the interface changes to a new appearance completely adapted to Blink code, as well as a new set of buttons:

The ▶ `Save and Recompile` button will save the code into the BlinkScript node, and then click on Recompile. As this can be dangerous (potential Nuke crashes if code isn't correct), the config options covered next will let you auto-save the blink code to disk every time you perform a `Save and Recompile`.

The 📥 `Blink auto-saves` menu gives you the options for auto-saving a scratch file to disk every time you click on `Save and Recompile`.

The `Auto-save…` checkbox will prompt you to create the .blink scratch file. Then, the other buttons will let you save and retrieve its contents.
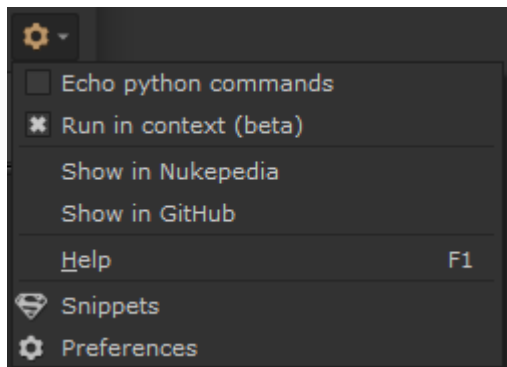
Common Features

The buttons on the right are actually common to both modes. Here's two we haven't mentioned yet:

🔍 Show or hide the **Find-Replace** widget. This is also activated through `Ctrl+F`.

The last button corresponds to the general `Settings`:

`Echo python commands`: toggles nuke echo.
`Run in context`: lets you execute code on a knob, whilst keeping the desired behaviour when you run `nuke.thisNode()` or `thisKnob()`
`Show...` and `Help`: links on your web browser.
`Snippets`: open the snippet editor (docs below!).
`Preferences` opens the Preferences panel.

Script Editor

The **KS Script Editor** is the main part of **KnobScripter** and the one in which you'll spend more time. It is a plain text editor which has been enhanced to include a full set of features for easier code writing/editing, navigation and viewing.

- **Syntax highlighting**
  The **KS Script Editor** features syntax highlighting for python and blink. Both were completely adapted to the scripting performed in Nuke. Additionally, for python scripting you can choose between two different highlighting styles/themes: Nuke Default and Monokai.

- **Duplicating lines and code**
  You can duplicate a line of code by pressing: `Ctrl (or cmd) + Shift + D`
  If you have any code selected, that code will get duplicated.

- **Moving lines up and down**
  You can move a line of code up or down pressing `Ctrl/cmd + Shift + Up/Down`
  If you select a code spanning across several lines, they will all be moved in block.

- **Auto indenting**
  When creating a new line, indenting will be added automatically.

- **Multi-line commenting and uncommenting**
  If you select several lines and press `#` (the hash key), a hash will be placed before each of the lines. If all of the lines already started with a hash, it will be removed.

- **Tab menu showing the available modules**
  If you press `Tab` after starting to type a word, if that word doesn't have a snippet associated with it, it will show Nuke's dropdown completer with the available modules and/or functions for the Script Editor.

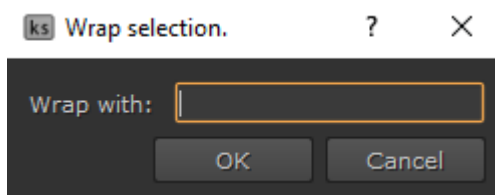- **Auto-completer and current script modules**
  Pressing `Tab` after starting to type a word will auto-complete it if a variable, function or class starting with those letters has been defined in the current script. If there's more than one match, the tab menu dropdown will appear and also include these words. In Blink mode, the autocompleter will work with most functions and keywords.

- **Other syntax helpers**
  - Opening parentheses will close them. *Brackets, braces and quotes too.*
  - Opening parentheses with text selected will open and close them around the selection. *Same with brackets, braces and quotes.*
  - I'm probably forgetting some. Will keep updating this documentation.
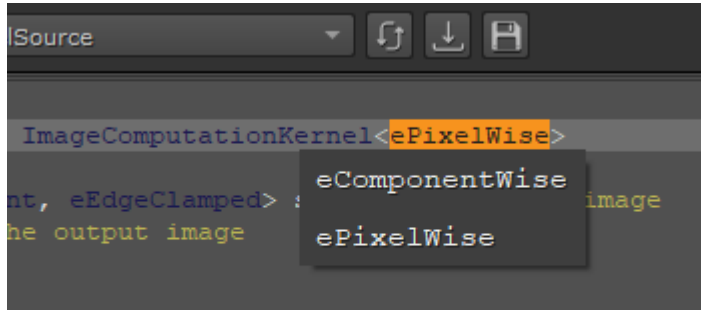
- **Parentheses function wrapper**
  If you select any part of the code on a single line and press `Tab`, a `Wrap selection` panel will appear, asking you for a function name. If you leave it blank, it will simply wrap the selection with parentheses (also selected). If you add some text on the panel, it will replace the selection with such text followed by the initial selection in parentheses, and all selected. This is really useful in the following example: any function you need help on, you select it, press tab, write help, enter, then execute the code which will execute help() on such word. Then you can do Ctrl+Z to go back. Also useful to turn `any text` into `int(any text)` or similar use cases.
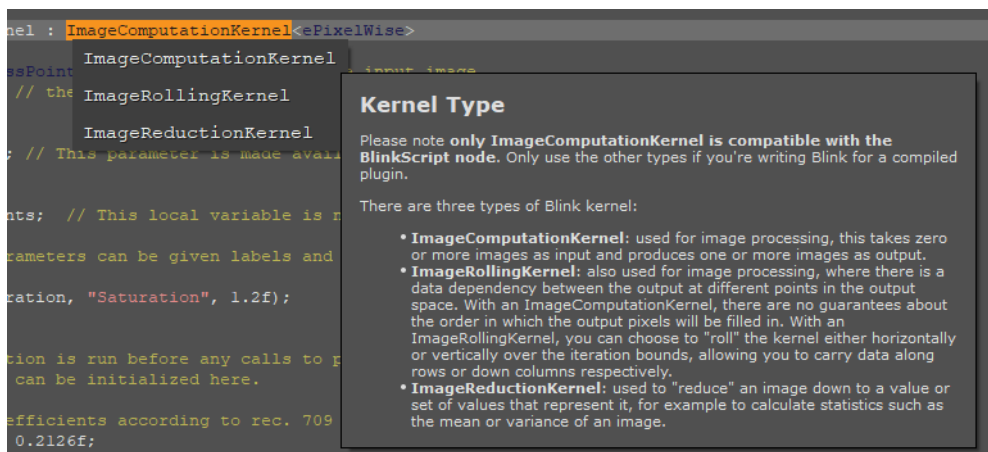
- **Keyword Hotbox**

  Currently only supported in the Blink mode: The Keyword Hotbox appears when you double click on a language-specific keyword. In the case of Blink, for example, when double-clicking on a keyword such as `ePixelWise`, the Keyword Hotbox will let you choose between the two modes of Kernel Granularity.
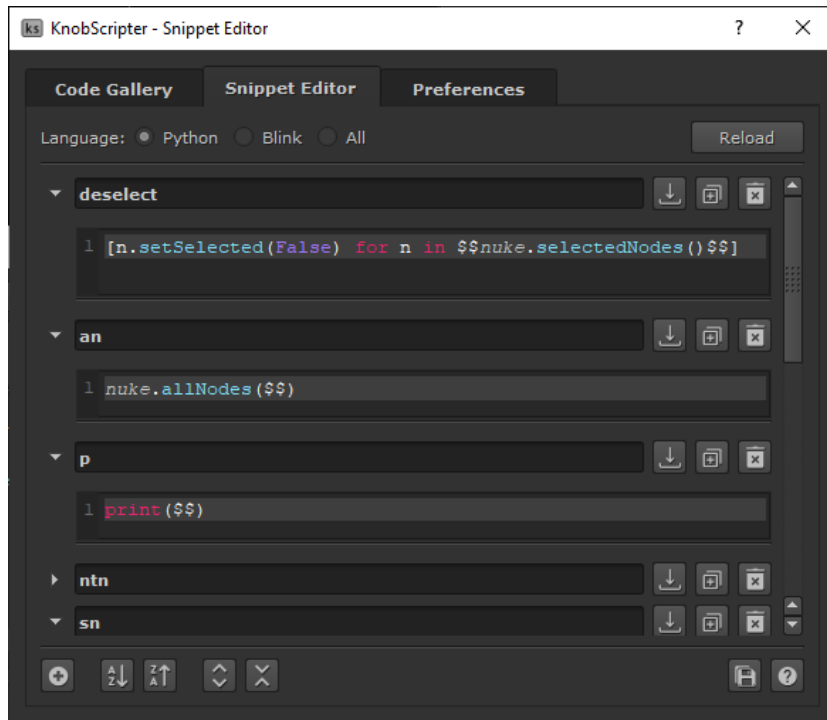

  Additionally, if you hover over the options, a full documentation tooltip will appear, helping you understand what each option means. Here's an example with the Keyboard Hotbox + Tooltip on the Kernel Type.

4. Snippet Editor and Snippets

Snippets are a convenient way to make your scripting faster, with the lines of code that you tend to write over and over again. A **snippet** is a short text that you can define, and associate a longer code to it, so that you can add the longer code to your script by simply typing the short text and pressing `Tab`.
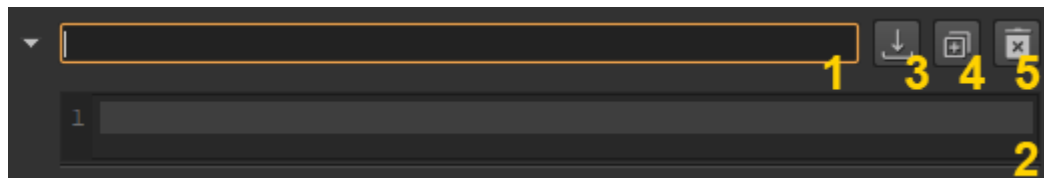
You can edit the snippets on the **Snippet Editor**, under the **Settings** dropdown button.

Create, Delete and Modify Snippets

To make a new snippet, first select the language you want the snippet to be active for (Python, Blink or All).

Click on (`Add Snippet`), and an empty snippet will be created:

New Snippet created

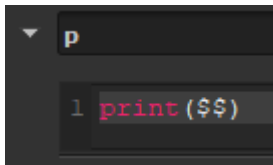|		
1	**Shortcode**: Write the short text that will activate this snippet. Example: `sns`	
2	**Code**: Write the full code that will appear when you write the shortcode and press tab. Example: `nuke.selectedNodes()` Note: The handle line on the bottom lets you expand the scripting area as needed.	
3	**Insert code into KnobScripter editor:** Add the current snippet's code into the open KnobScripter's text editor.	
4	**Duplicate Snippet:** Create a new snippet with the same shortcode and code as the current one.	
5	**Delete Snippet:** Remove the snippet from the Snippet Editor.	

Important considerations with Snippets:

- For snippets to be activated, the shortcode must be added after a separator of some kind (new line, space, (), =, etc).
- Snippets are saved on `.nuke/KS3/Snippets.txt`. You can edit that file too, if you're careful.
- Any changes you make on the Snippet Editor (creating, modifying or deleting snippets) won't take effect until you click on the following button :
  (Save all snippets)
- There is a **default snippet library** included with **KnobScripter**. It's the little collection of snippets I use the most, and I thought it would be cool to include it by default. However, the moment you add, remove or modify the snippets and save them (meaning when the tool finds your custom `KS3/Snippets.txt`), the default collection will be ignored. So if you don't like some of the default snippets, feel free to delete them and click save, and they won't be loaded anymore.

Cursor Placeholder

 Type `$$` anywhere on the Snippet code to create a cursor placeholder.
Then, the cursor will appear there after you press Tab. Check the `try` or `p` snippets included in the default library and you'll understand.

Selected text Placeholder

You can also add a selection placeholder, instead of just the cursor. It works the following way: If you create a Snippet with the code **hello $$world$$,** it will create the text **hello world**, with the word **world** selected straightaway. This will let you change it quickly afterwards or leave it as it is.

Variable Placeholders

Another cool thing you can do through the Snippets, is define variables. I explain it in detail in the tutorial video, but in short:
`$Node title$ = nuke.thisNode()`
`$Node title$_name = $Node title$.name()`
^ If you link this code to the shortcode `nodename`, for example, when you type `nodename` and press `Tab`, it will prompt you with this panel:
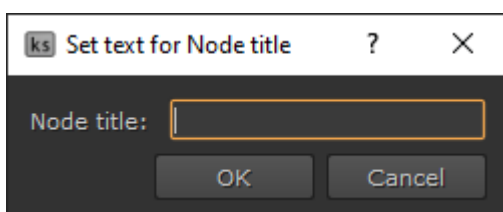
Then, after filling in the name (e.g. `blur`), the snippet code will be added to your script editor, with the word you just typed in the `$Node title$` placeholder location:

This is especially useful when you want that variable to **appear several times** on the same snippet. You can put **as many variable placeholders as you want**, or combine it with the cursor placeholder `$$` or selected text placeholder `$$text$$`.

Method Placeholders

If you add `$_$` on a snippet code text, the following will happen: if the snippet shortcode is executed after a dot (.) character, every occurrence of `$_$` will be replaced by the last word in your script right before the dot. You can find a useful example on the default snippet `cx`:

> Example snippet: `cx = xpos()+$_$.screenWidth()/2`
> Writing `johndoe.cx` and pressing `Tab` will create the following:
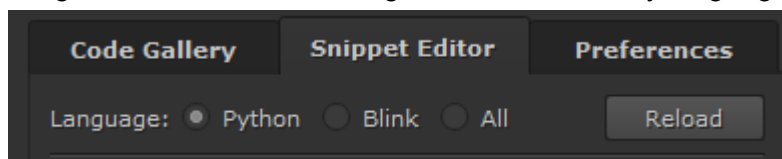> `johndoe.xpos() + johndoe.screenWidth()/2`

Lower buttons

They let you `Add` a new snippet, `sort` the snippets on the editor A->Z or Z->A, `Expand` or `contract` all snippets, and `Save` all snippets (remember, it's the only way to apply them on disk and not lose changes). Additionally, the help button will show a brief help panel.
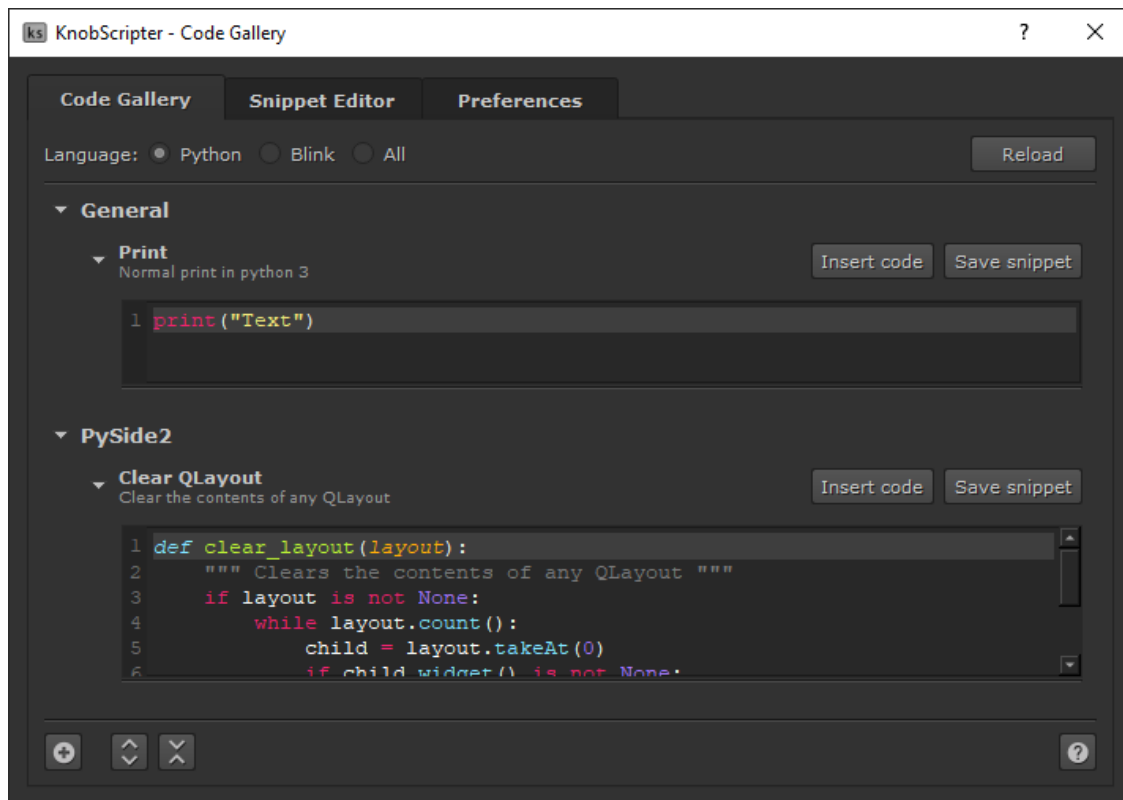

Language Selection and Reload

The language selection menu lets you switch to the snippet editor specific to a code language. The snippets linked to a language won't get activated on the others. The snippets living under All, however, will get activated in every language.
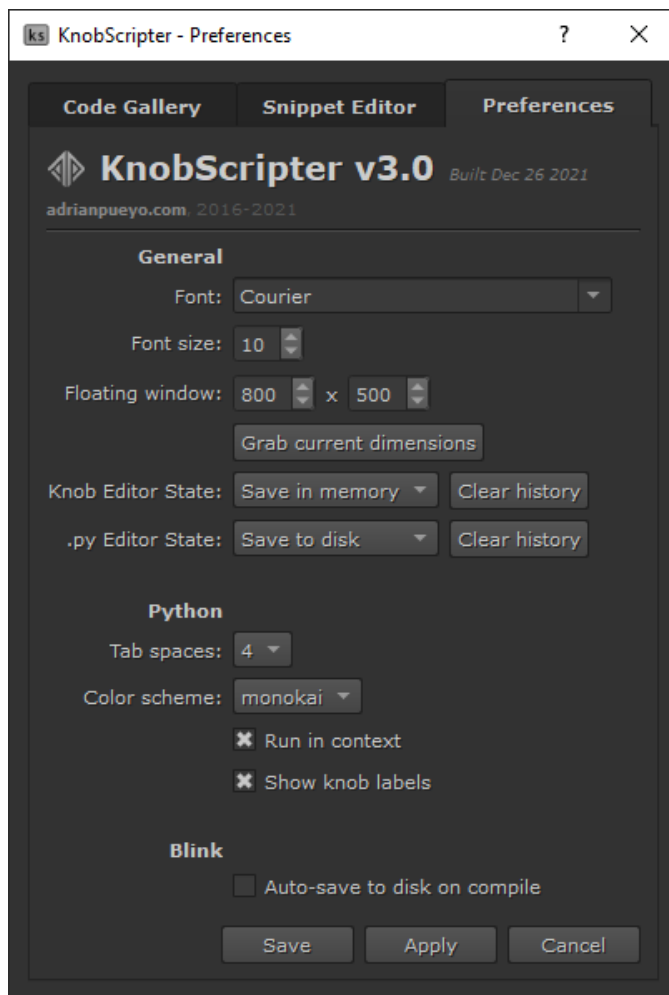
5. Code Gallery

The **Code Gallery** is a new way in which you can store pieces of code you'll want to later refer to as reference. It has a basic system of archiving, including Category, title, description and code. Then you can quickly browse and sort through the different codes you saved, while folding and unfolding categories and codes to help you browse through them more quickly.

- The 🔵 `Add new code` button opens a panel to write new codes for your gallery.
- The ⬍ ⤧ `Expand All` and `Collapse All` buttons let you quickly fold and unfold all codes and categories.
- The `Insert code` button lets you insert the current code into your KS Script Editor.
- The `Save Snippet` button opens a panel to turn your currently selected code into a new Snippet, then save all your snippets.
- I am aware this is still missing some important functionality but I still decided to roll it out and see if it proves useful for the community. If so, I'll work on further improvements on this panel and gather useful codes for the default Code Gallery.

6. Preferences

The **Preferences Panel** lets you modify the main settings for KnobScripter:

General:
`Font:` choose any font.
`Font size:` size of the editor font (pixels). Note: you can also change it dynamically via `ctrl +` and `ctrl -`

`Floating window w/h:` default dimensions for floating KnobScripters.

`Grab Current Dimensions:` set the default w/h values to the current ones.

`Knob/.py State:` where (and if) you want to save the scroll, selection and cursor values of the knobscripter on knobs and py scripts, so that when you open a new instance it goes directly to its last remembered state.
`Clear history:` clear the memory and dictionary that KS saves in disk, storing the editor state.

Python:
`Tab spaces:` number of spaces that will be created when indenting with tab.
`Color scheme:` python syntax highlighting color style.
`Run in context (beta):` whether to activate the run in context mode by default. If enabled, it lets you execute code you're writing in a knob as if it were executed from within its node. For example, if you're coding inside the knobChanged of a Blur node, and you have the code `print(nuke.thisNode().name())`, before it would print `Root` (as if you ran the code on Nuke's Script editor), but with `Run in Context` enabled it will print something like `Blur1`. Same with `nuke.thisKnob()`
`Show labels:` on node mode, display knob label and name on dropdown or just the name.

Blink:
`Auto-save to disk on compile:` whether to perform a kernel save to disk automatically every time you click compile.

7. Installation

A. Fresh install
1. Copy the *KnobScripter* folder and paste it somewhere in your Nuke plugin path. For example, inside `Users/YourUser/.nuke` directory.
2. Open with a text editor the file *menu.py* that lives next to your `KnobScripter` folder, or create one if it doesn't exist.
3. Add the following line:
   ```
   import KnobScripter
   ```
4. Restart Nuke.

B. Updating KnobScripter
1. Replace the *KnobScripter* folder with the updated one.
2. Restart Nuke.

8. Updates Log

KnobScripter v3.0 - *24 April 2022*

      New features:
1. Blinkscript mode.
2. Compatible with python 2 and 3 (Nuke 13+) and Win/Mac/Lin.
3. Code Gallery.
4. Fully redesigned Snippets, with a new Snippet Editor, default snippets and a simplified workflow.
5. Preferences Panel redone, and new settings added.
6. Keyword Hotbox for double clicking on keywords and showing alternatives.
7. Storing the knob and .py editor state, for quickly jumping back to your edit point on new KnobScripters.
8. New code helpers such as Ctrl/Cmd + and Ctrl/Cmd - for changing font size.
9. Method placeholders.
10. Parentheses wrappers.

      Bug fixes and enhancements:
1. Fixed predictor tab with custom words in Nuke 13.
2. Fixed backspace crash on indented lines.
3. Snippet behaviour improvements.
4. Fixed syntax highlighting errors.
5. Whole code refactored, and improved py import scheme.

KnobScripter v2.4 - *7 December 2020*

New features:

1. Run in Context mode.
2. "Grab current dimensions" button.

Bug fixes and enhancements:

1. Improved modal window behavior (thanks Jed Smith!)
2. Ctrl+Backspace now clears output log.
3. Improved how Nuke's Script Editor widgets are found (thanks Jed Smith!)
4. Modified (*) property now shows properly on the knob dropdown.
5. Picker ("change to Node") crashing fixed.
6. Unicode encoding works with both .py scripts and nodes now.
7. Compatible with Nuke 12.2
8. Add snippet button scrolls to top.

KnobScripter v2.3 - *24 December 2019*

Bug fixes:

1. Fixed bracket and brace closing behaviour.
2. Added QStringListModel compatibility on nuke 12.
3. Symlink error fixed hidden on Windows.

KnobScripter v2.2 - *12 August 2019*

Bug fixes:

1. Variable placeholders error fixed (bug introduced on v2.1).
2. PySide import switch including Qt.

KnobScripter v2.1 - *10 August 2019*

New features:

1. New Sublime color style. You can now choose between sublime or Nuke style in the Preferences.
2. Font selector in the Preferences, where you can set any available font.
3. Option to display the knob labels too (and not only names) on the knob dropdown.
4. Now accepting special characters (like accents or symbols) through utf-8 encoding.
5. New Snippets functionality: A snippet containing "hello $$world$$" will write "hello world" and the word "world" will be selected straightaway.
6. Auto-completer (when pressing tab) now also includes the functions, variables, classes etc live from the current script you're writing.

Bug fixes and enhancements:

1. Python syntax highlighting improved.
2. Snippet editor now includes syntax highlighting.
3. Parenthesis/brackets auto-closing behavior improved when written inside each other.
4. Fixed error that prevented opening a new KnobScripter when a Blinkscript node properties panel was open too.
5. Tested on Windows OS (Nuke 11.3) and a few minor bugs fixed.

6. Improved auto-scroll to cursor behavior when duplicating or moving lines.
7. Unindent keeps scroll value.

Guide last updated: 24 April 2022

9. License

www.adrianpueyo.com