KrySA Documentation

Release 0.3.7

Peter Badida

September 18, 2016

Contents

Statistical analysis for rats.

A common translation of the word “krysa” is “a rat”, mostly considered a lower creature, but in fact a really cute and intelligent little rodent.

The idea behind KrySA is to make statistical software available even for the “rats” - people who can’t or don’t want to buy a commercial tool for statistics for whatever reason. KrySA is a free, open-source tool that anyone can afford!

- KrySA is released under GNU GPL v3.0 License. Please read the LICENSE.txt file.

CHAPTER 1

Contents

1.1 Getting started

Warning: KrySA is still in pre-alpha, most of the features are buggy or not yet supported. Read *Contributing* section if you want to help speed up the process.

KrySA runs on Kivy framework, therefore it is possible to run it on any of available platforms for Kivy, mainly Windows, Linux and Mac with all required packages correctly compiled:

- Kivy
- SciPy
- NumPy
- MatPlotLib

There's no executable for KrySA yet, you'll need to install it from source and run with Python until there is a release available.

1.1.1 Minimum system requirements

RAM	At least 256 MB
Disk space	At least 400 MB free(*)
Resolution	Minimum of 800 x 600
CPU	? ? ?
GPU	Anything with OpenGL 2.0 support should be enough
Internet	Necessary for downloading requirements and updating

*if installing from scratch

1.1.2 Installation

First of all you'll need Python. To simplify the process use KivyInstaller on Windows, which will install Python together with Kivy. On other platforms use Kivy Installation page as reference.

Note: KrySA requires the latest version of Kivy. It's available either as daily-builds on `ppa` or as the `.whl` files uploaded on Google Drive. If none of those are good, compile Kivy from source.

Then it gets a little bit harder with SciPy and NumPy because those need to be compiled and it sometimes doesn't work with Windows. For this case we will use already compiled packages in `.whl` files. You can find them either on pypi or here. Choose packages for Python 2.7 (cp27). On Linux they should work without issues with `pip install <package>`.

```
pip install <path to package>.whl
```

Then install MatPlotLib. This is easy even on Windows:

```
pip install matplotlib
```

1.1.3 Getting KrySA

There are many ways how to get it, but basically you need to download it from the official repository.

1. Pip

   KrySA is available on PyPi, simply type:

   ```
   pip install krysa
   ```

   and then run it with:

   ```
   python -m krysa
   ```

2. Git

   Clone the whole repository and be able to update KrySA when a new version arrives with a simple `git pull`.

   ```
   git clone https://github.com/KeyWeeUsr/KrySA
   ```

3. Zip

   Click on the `Clone or download` button, download the zip file and unpack its contents.

When the repo clone(git/zip) of KrySA is ready, simply navigate into it and run:

```
python main.py
```

1.1.4 About docs

The documentation includes source with notes how most of the things work for example which widgets are connected, what's needed to call to make a custom *Task* and other related stuff.

Each documented class or function/method will have a little *source* link on the right side. This will send you to its place in the code. In the code there are similar *docs* links (they'll return you back) at the same place as it was for the *source* in modules' documentation.

1.2 Project

As it's obvious from the title, the main part of the application is a *Project* which is a folder with a `.krysa` file and some folders e.g. for *Data* and *Results*. You are forced to create a *Project* even with not being able to import data alone. It keeps your work at one place and makes it simplier to *Open and Save* it.

Select `File -> New -> Project`, navigate to a folder you want to save it to and KrySA will create a `<Project name>` folder there. Project's name can have only lowercase & uppercase ASCII and numbers.

Warning: Please do not manually edit any of the files, it may result in unexpected behavior and KrySA may crash

1.2.1 Open and Save

Open

To open already existing *Project* select `File -> Open Project` and then navigate to a folder with `.krysa` file. Select the file with a click or tap and press `Open`.

Save

Each new *Project* is automatically saved in the beginning with an empty *Data file* and already created folders. Please do not remove any of them even if it's nothing there.

To save changes made in a new *Project* select `File -> Save Project`, it will already know there's an active project and save it in the same folder without selecting where to save it.

Note: If there's no active project, saving does nothing.

1.2.2 Data file

KrySA creates a single `.sqlite` file which handles all the *Data* you create. Although *SQLite* doesn't limit its columns by default, KrySA uses this option to prevent crashing caused by a user's mistake of running a *Task* expecting only numbers with a value of type *TEXT*.

Data

In the beginning there's an empty *Data file*, which means we need to populate it.

Select `File -> New -> Data` and name it. Data's name can have only lowercase & uppercase ASCII and numbers. Then create columns (same rules for the names) and input new values to them. Remember, for each column you have to select a type of its values:

Type	Description
REAL	Only numbers with a single `.` symbol (`1.1`)
INTEGER	Only numbers without any special symbols
TEXT	Non-limited value converting input directly to unicode

Note: When an input box for the first value is added, the type of the column automatically locks to prevent values of different type in the single column e.g. *REAL* and *TEXT*.

Warning: Each column must have a unique name!

After the column is finished, you can `Check & Lock` the values. It'll check if the values are the same as the column type and tell you if not. You can always unlock the values later for example if the application tells you about wrong values. When you're finished, type `Run`, it'll run `Check & Lock` for each available column. If all the columns pass

the test, a new tab after the *Process Flow* tab is created and then the application export *every* present data to the *Data file*.

Each column in finished *Data* has an address you can access it later with in a *Task*.

Editing

Each cell in *Data* is clickable and editable in the limits of the column type. Press `<enter>` (`<return>`) to confirm the edit, otherwise it won't change the value and only unfocus the cell.

Note: Edited cells aren't automatically saved to *Data file*, how to save read in *Open and Save*.

Importing

Whenever you want to combine data from two or more *Project* s or just add additional tables from premade *Data file*, this is the way.

Select `File -> Import Data`, navigate to `.sqlite` file, select it with a click or tap and press `Import`. It will add another tab(s) containing the data at the end of the panel.

Warning: Before importing check if the column names don't collide, otherwise it may result in unexpected behavior and KrySA may crash.

Exporting

This will export *all* data you can see on the panel to a *Data file* which can be then accessed either with different editor or saved for later use in KrySA e.g. for combining data.

Select `File -> Export Data`, navigate to a folder you want to put the *Data file* to, select it with a click or tap and press `Export`.

1.2.3 Results

Results are by default *.png* files in a resolution of 72DPI A4 page (595px x 842px) put in the *results* folder in the *Project* folder.

Note: Making a single file with all results is still under construction.

1.2.4 Process Flow

Nothing yet.

1.3 Task

This is the most important part of KrySA, it is the way of manipulating *Data* values and reporting the result. Tasks are categorized by its purpose and/or complexity into different groups e.g. `Basic`.

1.3.1 Using a Task

Each *Task* needs values which it can use, otherwise it won't run. When the values are present, each column has an address starting with `A` for the first column. To select more than a single value, use `:` character (e.g. `B1:AB2`):

.	A	B	C	...	AA	AB	AC	...	ZZ...
1	x	x	x	x	x				
2	x	x	x	x	x				

After the values are selected, pres `Run`. Depending on the *Task*, it can create new *Data* or a page in the *Results* panel.

1.3.2 Create a Task

Note: It's good to peek in *`tasks.Task`*.

KrySA uses Python for *Task* s. Each task according to its category(file) begins with a function named like this:

```
def <category>_<task>(*args):
```

This function sets the layout that is put into the *Task* popup, sets a function that is called when user selects some options in the popup and opens it. The *Task* s layout contains an option to select which *Data* will be used in the following task, but you have to handle user's input of the address(`A1:B2`).

```
def <category>_<task>(*args):
    widget = SomeLayout()
    task = Task(title='Title', wdg=widget,
                call=['Title', <category>_<task>])
    task.run = partial(<called function>,
                       task,
                       <widget containing address>)
```

It's necessary to put into `Task()` the layout and a link to itself. Layout then can be accessed in the called function directly from arguments and the link is used to append the used *Task* to list of *Recent Tasks*.

Then it's necessary to write the `<called function>` and handle its inputs. Each *Task* must have some kind of output - new *Data*, modified *Data* or a page in the *Results*:

```
def <called_function>(task, address, *args):
```

Each `<called function>` takes at least two arguments `task` and `address`, where `task` is an instance of the main popup (so that you can access the chosen *Data*) and `address` is the widget with some kind of string property.

To get the values from user's input use the function *task.from_address()*, which is basically *Body.from_address()* accessed from within *Task*. The function takes two arguments - index of *Data* (returned in *task.tablenum* property) and string of address.

```
values = task.from_address(task.tablenum, address.text)
```

Values are returned as a simple list of everything selected no matter what the type it is. Example:

```
values = [0, 1.0, u'hi']
max(values)
>>> u'hi'
```

When you are finished, output the values e.g. into *Results* with *task.set_page*:

```
task.set_page('Count', str(len(values)), 'text')
```

Final functions would look like this:

```
def basic_count(*args):
    widget = CountLayout()
    task = Task(title='Count', wdg=widget,
                call=['Count', basic_count])
    task.run = partial(_basic_count,
                       task,
                       task.ids.container.children[0].ids.name)
    task.open()

def _basic_count(task, address, *args):
    values = task.from_address(task.tablenum, address.text)
    task.set_page('Count', str(len(values)), 'text')
```

1.4 Contributing

There are three parts of the project you can contribute to, but only two of them require at least some programming skills (mainly in Python). Each part, however, requires a fully functional KrySA application.

1.4.1 Documentation

As the project is still in the beginning, there's a lot of things to document and to make screenshots of. If you have KrySA already installed, there's a `docs` folder that contains the documentation.

The documentation is written in reStructuredText which you can test either in some online editor (referencing files won't work, obviously) or localy if you have already installed Python. KrySA uses Sphinx for converting reStructuredText to a html website. First install requirements from the `.txt` file.

```
pip install -r docs-requirements.txt
```

To build the documentation use these commands in the `docs` folder:

```
make clean && make html
```

Note: Extend the command with another `&&` to e.g. automatically open a browser with fresh `index.html` file.

Please don't break the formatting (max 79 characters in a single line) and fix the errors if any jumps out in Sphinx build.

1.4.2 Statistics

Hypotesis testing, factor analysis, averages, whatever part of statistics you think a user could find useful you can do two things:

1. Feature request

   Open an issue in the GitHub repository describing the feature and its use case.

2. Pull request

   Read the code, find out how it works and make a pull request to the GitHub repository with code that doesn't break the Test Suite together with an example of how the new feature works.

1.4.3 Application

If you think the application might find your feature useful or that some behavior needs a fix, you are welcome to make a pull request. Before each pull make sure it is written in Python's PEP8 style and that it doesn't break the Test Suite. KivyUnitTest makes running the tests easier.

1.5 License

1.5.1 The Software

KrySA is released under the GNU General Public License (GPL, or "free software").

This license grants people a number of freedoms:

- You are free to use KrySA, for any purpose
- You are free to distribute KrySA
- You can study how KrySA works and change it
- You can distribute changed versions of KrySA

The GPL strictly aims at protecting these freedoms, requiring everyone to share their modifications when they also share the software in public. That aspect is commonly referred to as Copyleft.

1.5.2 License details

The developed KrySA source code is by default licensed as GNU GPL Version 3.

KrySA also uses some modules/libraries from other projects. For example Python uses the Python License, Kivy uses the MIT License, SciPy and NumPy use the 3-clause BSD License, and MatPlotLib uses a license based on Python License.

All the components that together make KrySA are compatible under the GNU GPL Version 3. That is also the license to use for any distribution of KrySA binaries.

1.5.3 Your output

What you create with KrySA (e.g. *Data file* or *Results*) is your sole property. All your other output, meaning images/graphs, tables, etc. including the `.krysa` files and other data files Krysa can write, is free for you to use as you like.

That means the application can be used commercially by anyone to work on commercial projects, research, or for educational purposes.

KrySA's GNU GPL license guarantees you this freedom. Nobody is ever permitted to take it away, in contrast to trial or "educational" versions of commercial software that will forbid your work in commercial situations.

CHAPTER 2

Modules

- Index

2.1 KrySA

class `main.`**Body**(***kw*)

The main layout for the application. It handles menu values, their appropriate functions, filtering of user's input and functions for accessing *Data file* in `main.Table`.

New in version 0.1.0.

_export_data(*selection*, *fname*, **args*)

Exports all available *Data* (visible as tabs) as *Data file* into path selected in `Dialog`.

New in version 0.1.1.

static _extract_rows(*data*)

Extract values from `main.Table`'s dictionary into a flat list.

Example:

Data1	Data2	Data3
1	2.0	3

[u'Data1', u'Data2', u'Data3', u'1', 2.0, 3, ...]

New in version 0.1.0.

_import_data(*selection*, **args*)

Imports *Data file* from path selected in `Dialog` and puts it to `main.Table`.

New in version 0.1.0.

_new_data(**args*)

Opens a wizard for creating a new *Data* if a *Project* is available or shows a warning if it doesn't exist.

New in version 0.1.3.

_new_project(**args*)

Closes already opened *Project* if available and opens a dialog for creating a new one.

New in version 0.1.2.

_open_project(*selection*, **args*)

Opens a *Project* from path selected in `Dialog` and imports *Data file*.

New in version 0.1.7.

_save_data (*wizard*, **args*)
Gets data from the wizard, puts them into *main.Table* and exports them into *Data file*.

New in version 0.1.4.

_save_project (*selection=None*, *fname=None*, **args*)
Saves a *Project* to path selected in `Dialog` and exports *Data file*.

New in version 0.1.2.

static about (**args*)
Displays *about* page of the app and includes other credits.

New in version 0.1.0.

close_project (**args*)
Clears all important variables, removes all *Data* available in *main.Table* and switches to *main.ProcessFlow*.

New in version 0.1.0.

from_address (*table*, *address*, *extended=False*, **args*)
Gets value(s) from *main.Table* according to the address such as `A1` or `A1:B2`. Values are fetched in the way that the final list contains even empty (`u''`) values. It is not expected of user to use *Task* for strings and most of them won't even run. To get non-empty values for a *Task* use for example Python's `filter()`:

```
values = filter(lambda x: len(str(x)), values)
```

This *filter*, however, will remain values such as `None` untouched.

New in version 0.1.0.

Changed in version 0.3.5: Added extended options and a possibility to get `:all` values from data.

new (*button*, **args*)
Opens a submenu for `New` menu.

New in version 0.1.0.

set_page (*task*, *result*, *result_type='text'*, *footer='time'*)
Creates a *main.PageBox* for a result. The header consists of the *Task*'s name, the footer is by default the time when the result was created and the content depends on *result_type* which can be - text, image(path to image) or widget. If *result_type == 'widget'*, result has to be an instance of a widget (obviously containing the output), e.g.:

```
b = Button(text='my output')
set_page('MyTask', b, result_type='widget')
```

Note: When exporting pages, everything is converted into images (pngs), therefore making fancy behaving widgets is irrelevant.

New in version 0.2.0.

Changed in version 0.3.2: Added tables as a result type.

class `main.`**CreateWizard** (***kw*)
A popup handling the behavior for creating a new *Data*, i.e a wizard.

New in version 0.1.3.

class main.**Dialog**(**kw)
A dialog handling the behavior for creating or opening files e.g. *Project* or *Data*.

New in version 0.1.0.

class main.**ErrorPop**(**kw)
An error popup to let user know something is missing or typed wrong when console is disabled.

New in version 0.1.2.

class main.**ImgButton**(**kwargs)
A button with an image of square shape in the middle.

New in version 0.2.0.

class main.**KrySA**(**kwargs)
The main class of the application through which is handled the communication of other classes with getting an instance of the app via `App.get_running_app()`.

Other than that, it holds important variables of *Project*, sql blacklist for *Data file* creating and updating or the application properties themselves.

build()
Default Kivy function for getting the root widget of application.

errorcls
alias of *ErrorPop*

on_project_exists(*instance*, *exists*)
Checks change of `main.KrySA.project_exists` and if *Project* exists, schedules updating of its tree to 5 second interval.

New in version 0.3.0.

tablecls
alias of *Table*

class main.**MenuDrop**(**kw)
A list of *main.SizedButton* s displayed as a menu, where each button may create another menu depending on the function bound to it. The main menu is handled through a single instance of *main.MenuDrop* which is instantiated before `main.Krysa.build` function.

Each click/tap on the menu button then assigns a value to it from `App.menu` dictionary according to its name in *kv* file.

New in version 0.1.0.

class main.**NewDataColumn**(**kw)
A layout handling the behavior of type, values(`NewDataValue`) and some buttons for each new column in *Data*.

New in version 0.1.4.

checklock(*disable*, *coltype*, *check*, **args*)
Disables all cells in the column, then check them against a list of strings that could be used to corrupt *Data file* . If the check is done without an error, another check is made to protect against using an empty string `''` as a value, which if used inappropriately results in a crash.

New in version 0.1.4.

static free(*items*)
Frees all locked cells in the column except a column type. If a wrong type is used, removing the whole column is necessary. (protection against corrupting *Data file*)

New in version 0.1.4.

paste(*values*, *sep*)
Paste a value(s) from a user's clipboard as a column values. A user can choose what kind of separator was used on the values, for example:

```
1 2 3 4 5     # (space)
1\t2\t3\t4\t  # (tab)
1\n2\n3\n4\n  # Unix-like new line character (<enter>/<return>)
```

If in doubt and your values were copied from a column (e.g. spreadsheet), use *OS default*, which will choose between `\n` (Unix-like) or `\r\n` (Windows) new line separators.

New in version 0.3.4.

class main.**NewDataLayout**(***kw*)
A layout handling the behavior of `NewDataColumn` and some inputs for each new value in *Data*.

New in version 0.1.3.

class main.**NewDataValue**(***kw*)
A layout handling the behavior of inputs and button for each new value in *Data*.

New in version 0.1.4.

class main.**PageBox**(***kwargs*)
A layout that includes Page widget together with transparent separator. It's used for adding new results from Tasks.

New in version 0.2.0.

class main.**PaperLabel**(***kwargs*)
A label with visual properties as a paper sheet.

New in version 0.2.0.

class main.**ProcessFlow**(***kw*)
A canvas on which will be displayed actions for each *Data* related to them, such as used tasks connected with result of the tasks.

New in version 0.1.0.

(Not implemented yet)

class main.**ResultGrid**(***kwargs*)
A black gridlayout, together with *main.Wrap* makes a table container for results that need a table.

New in version 0.3.2.

class main.**SideItem**(***kwargs*)
Supposed to be a part of settings, most likely will be removed/replaced.

New in version 0.1.0.

class main.**SizedButton**(***kwargs*)
A button with width automatically customized according to text length of its siblings, which makes every sibling the same size as the one with the longest text string.

New in version 0.1.0.

class main.**Table**(***kw*)
A view handling the behavior of the inputs from *Data file*. Separates the values from *Data file* according to its *Data*'s column types into three Python categories - *int*, *float* or *unicode* and assigns an alphabetic order for each column together with row number to each value.

New in version 0.1.0.

clean(**args*)
Removes all data from *main.Table*

New in version 0.1.0.

get_letters()
Gets a list of letters the same length as *Data*'s columns.

New in version 0.1.0.

lock(*disabled=True*)
docs

New in version 0.1.0.

class main.**TableItem**(***kwargs*)
An item handling the behavior or each separate value in the *main.Table* such as updating/editing values in *Data*.

New in version 0.1.0.

on_focus(*widget*, *focused*)
Makes sure the unconfirmed value is discarded e.g. when clicked outside of the widget.

update_value(*txt*, **args*)
On `<enter>` (`return`) key updates the values `main.TableItem.text` and `main.TableItem.old_text` in *main.Table*.

New in version 0.1.0.

class main.**Wrap**(***kwargs*)
A white label with automatically wrapped text.

New in version 0.3.2.

2.2 KrySA » Tasks

2.2.1 KrySA » Tasks » Basic

class tasks.basic.**Basic**
All *Task* s categorized as *basic* under one roof.

New in version 0.1.0.

static _basic_count(*task*, *address*, **args*)
Gets the values from address and returns the count.

New in version 0.1.0.

static _basic_freq(*task*, *address*, *bins*, *limits*, *freq_type*, *intervals*, **args*)
Gets the values from address and depending on the type of values dumps them either into bins of size 1 (integers) or into bins that consist of intervals (real numbers). Then according to the size of bins and limits of the frequency creates a table for chosen types of frequency.

May return a warning if *intervals* option isn't checked for values containing real numbers:

```
IndexError: index max(<values>) + 1> is out of bounds for axis 1
with size max(<values>) + 1>
```

New in version 0.3.2.

static _basic_large(*task*, *address*, *k*, **args*)
Gets the values from address and returns the *k*-th value from the descending list of sorted values.

New in version 0.1.0.

static _basic_max(*task*, *address*, **args*)
Gets the values from address and returns a maximum.

New in version 0.1.0.

static _basic_min(*task*, *address*, **args*)
Gets the values from address and returns a minimum.

New in version 0.1.0.

static _basic_small(*task*, *address*, *k*, **args*)
Gets the values from address and returns the *k*-th value from the ascending list of sorted values.

New in version 0.1.0.

basic_count(**args*)
Opens a `tasks.Task` with a `tasks.AddressLayout` that gets from user *Data* address.

New in version 0.1.0.

basic_countifs(**args*)
Not yet implemented.

basic_freq(**args*)
(Not fully tested yet) Opens a `tasks.Task` with a `tasks.FreqLayout` that gets from user:

- *Data* address
- type of values (*interval* for real numbers)
- type of frequency (*absolute*, *relative* or *cumulative*
- number of bins (optional)
- upper and lower limit (optional)

New in version 0.3.2.

basic_large(**args*)
Opens a `tasks.Task` with a `tasks.SmallLargeLayout` that gets from user *Data* address and *k* variable representing the *k*-th value from the *Task* s output.

New in version 0.1.0.

basic_max(**args*)
Opens a `tasks.Task` with a `tasks.AddressLayout` that gets from user *Data* address.

New in version 0.1.0.

basic_min(**args*)
Opens a `tasks.Task` with a `tasks.AddressLayout` that gets from user *Data* address.

New in version 0.1.0.

basic_small(**args*)
Opens a `tasks.Task` with a `tasks.SmallLargeLayout` that gets from user *Data* address and *k* variable representing the *k*-th value from the *Task* s output.

New in version 0.1.0.

2.2.2 KrySA » Tasks » Avgs

class tasks.avgs.**Avgs**
All *Task* s categorized as *averages* under one roof.

New in version 0.2.4.

static _avgs_gen(*task*, *address*, *p*, **args*)
Gets the values from address and depending on *p* (power) value returns either exceptional case for *p == 0* (geometric mean), or value from the generalized mean's formula.

New in version 0.2.4.

avgs_gen(**args*)
Generalized mean:

$$(\frac{1}{n}\sum_{i=1}^{n} x_i^p)^{\frac{1}{p}}, where:$$

- p == -1: harmonic
- p == 0: geometric
- p == 1: arithmetic
- p == 2: quadratic
- p == 3: cubic
- etc...

New in version 0.2.4.

avgs_inter(**args*)
(Not yet implemented)

avgs_median(**args*)
(Not yet implemented)

avgs_mid(**args*)
(Not yet implemented)

avgs_mode(**args*)
(Not yet implemented)

avgs_trim(**args*)
(Not yet implemented)

2.2.3 KrySA » Tasks » Manipulate

class tasks.manipulate.**Manipulate**
All *Task* s categorized as being able to *manipulate* data. A result after manipulation is a new data.

New in version 0.3.5.

static _manip_append(*task*, *append_type*, *amount*, *overwrite*, **args*)
Gets the amount of empty rows / cols to append from user and either returns a new, altered *main.Table* of selected one, or appends directly to the selected Table.

New in version 0.3.6.

Changed in version 0.3.7: Added overwriting of selected *main.Table*

static _manip_sort(*task*, *sort_type*, **args*)
Gets the values from address, sorts each column either ascending or descending and returns a new *main.Table*

New in version 0.3.5.

manip_append(**args*)
Opens a *tasks.Task* with a *tasks.AppendLayout* that gets from user *main.Table*, type of append and an amount of empty rows / cols to append.

Note: Appending new columns don't work for now. When such an action is possible, this note will be removed.

New in version 0.3.6.

manip_filter(**args*)
(Not yet implemented)

manip_merge(**args*)
(Not yet implemented)

manip_sort(**args*)
Opens a *tasks.Task* with a *tasks.SortLayout* that gets from user the table which will be sorted and the type of sorting (*Ascending* or *Descending*).

New in version 0.3.5.

manip_split(**args*)
(Not yet implemented)

class tasks.**AddressLayout**(***kwargs*)
Simple layout that consists of single restricted input widget fetching only `[a-zA-Z0-9:]` values i.e. address.

class tasks.**AppendLayout**(***kwargs*)
A layout that consists of a spinner with two values:

- Rows
- Columns

and a restricted input that allows only integers.

New in version 0.3.6.

class tasks.**AvgsLayout**(***kwargs*)
A layout that consists of multiple restricted input widgets for address and *p* (power) value for the formula of generalized mean.

New in version 0.2.4.

floatfilter(*substring*, *from_undo*)
A function filtering everything that is not - symbol, floating point symbol(.) or a number.

class tasks.**FreqLayout**(***kwargs*)
A layout that consists of multiple checkboxes and restricted input widgets for address, type of values, type of output frequency and limits of the input values.

New in version 0.3.2.

class tasks.**SmallLargeLayout**(***kwargs*)
A layout that consists of multiple restricted input widgets for address and *k* value.

New in version 0.1.0.

class tasks.**SortLayout**(**kwargs)
A layout that consists only of a spinner with two values:

- Ascending
- Descending

The *Task* with this layout is using *tasks.manipulate.Manipulate._manip_sort*.

New in version 0.3.5.

class tasks.**Task**(**kw)
A popup handling the basic choosing of *Data* from available *Data file* in the application.

New in version 0.1.0.

Changed in version 0.2.3: Placed into a separated module.

static get_table_pos(*text*, *values*, **args*)
Returns an index of selected *main.Table* from all available in the list.

New in version 0.1.0.

recalc_height(*body*, *content*)
Recalculates the height of *tasks.Task* after a layout is added, so that the children are clearly visible without any stretching.

New in version 0.3.2.

try_run(**args*)
Tries to run a *Task* from the input a user specified and closes the popup. If no such an action is possible, it'll show a popup with an error and leave *tasks.Task* opened.

New in version 0.2.0.

2.3 KrySA » Tests

Tests run independently on each other, but not in a single python interpreter. After each test a fresh *python* is required (it won't run as casual suite), so either run them like this one by one:

```
python test_<something>.py
```

or use KivyUnitTest to do it instead of you and better.

m

t

Symbols

A

B

C

D

E

F

G

I

K

L

M

N

O

P

R

S

T

U

W