mathmaker Documentation

Release 0.7.1dev5

Nicolas Hainaux <nh.techn@gmail.com>

May 05, 2017

Contents

CHAPTER 1

User's guide

Contents:

Overview

Mathmaker creates elementary maths worksheets with detailed solutions.

The output documents can be compiled into pdf files by lualatex. Examples of available themes are: first degree equations, pythagorean theorem, fractions calculation...

It can run from command line, but can be controlled via http requests too.

License

Documentation (master release)

Documentation (latest development release).

Quickstart

Install

Required (non python) dependencies are python >=3.4, euktoeps >=1.5.4, xmllint, msgfmt, lualatex, luaotfload-tool and a bunch of LaTeX packages(1)

To install them:

- on Ubuntu, python3.4 is already installed, so:

```
$ sudo apt-get install eukleides libxml2-utils gettext texlive-latex-base
```

And for the yet missing LaTeX packages: you can either install the complete texlive distribution (takes some room on the hard disk): run `$ sudo apt-get install texlive-full`, or install only the necessary packages:

```
$ sudo apt-get install texlive-luatex texlive-latex-recommended texlive-xetex␣
↪texlive-pstricks texlive-font-utils texlive-latex-extra texlive-base texlive-
↪science texlive-pictures texlive-generic-recommended texlive-fonts-recommended␣
↪texlive-fonts-extra
```

- on FreeBSD(2):

```
$ sudo pkg install python34 py34-sqlite3 gettext eukleides libxml2 texlive-full
$ rehash
```

Note: Check how to fix eukleides install in the complete documentation

Once you're done, you can proceed installing mathmaker:

```
$ pip3 install mathmaker
```

(this will automatically install three extra python3 libraries too: polib, PyYAML and python-daemon).

Note: FreeBSD users, check how to fix mathmaker install if it stops with an error in python-daemon install, in the complete documentation

Basic use

```
$ mathmaker pythagorean-theorem-short-test > out.tex
$ lualatex out.tex
```

or directly:

```
$ mathmaker pythagorean-theorem-short-test --pdf > out.pdf
```

Get the list of all provided sheets(3):

```
$ mathmaker list
```

Some settings

Check `mathmaker --help` to see which settings can be changed as command line arguments.

Some more settings can be overriden by user defined values in `~/.config/mathmaker/user_config.yaml`. Read the complete documentation for more information.

Advanced use

It's possible to create your own sheets in xml (only for the mental calculation theme yet). Read the complete documentation for more information.

Contribute

You can contribute to mathmaker:

As a wordings contributor

Mathmaker needs contexts for problems wordings. There are already some, but the more there is, the better. Existing wordings can be found here. You can submit any new idea as an enhancement proposal there (should be written in english, french or german).

Any question can be sent to nh dot techn (hosted at gmail dot com).

As a translator

You can help translating mathmaker to your language (or any language you like, if you have enough elementary maths vocabulary for that).

If the translation to your language isn't started yet, there are several pot files to get here (see explanations about their respective roles there). You can use an editor like poedit or any other you like better, to create po files from them and start to translate.

If you want to add missing translations, or to correct some, you can find the po files in the subdirectories here.

Once you're done, you can make a pull request here.

Any question can be sent to nh dot techn (hosted at gmail dot com).

As a developer

Before submitting a PR, please ensure you've had a look at the writing rules.

More details can be found in the documentation for developers.

Any question can be sent to nh dot techn (hosted at gmail dot com).

Contributors

Development

- Lead developer: Nicolas Hainaux
- Developers: Vaibhav Gupta
- Clever advices: Olivier Cecillon

Translation

- French: Nicolas Hainaux

Problems wordings

Nicolas Hainaux

Patience and chocolate cakes

Sophie Reboud

Changelog

New in version 0.7.1dev5 (2017-05-04)

- Issue warnings instead of exceptions when the version of a dependency could not be determined.

New in version 0.7.1dev4 (2017-05-03)

- New sheets about trigonometry: - vocabulary in the right triangle - write the correct formulae - calculate a length - calculate an angle

New in version 0.7.1dev3 (2016-10-21)

- New sheets: - intercept theorem: “butterfly” configuration - intercept theorem: converse

New in version 0.7.1dev2 (2016-10-13)

- New sheets: - expansion of simple brackets (declined in two versions) - clever multiplications (mental calculation) - intercept theorem: write the correct quotients’ equalities - intercept theorem: solve simple exercises

These sheets and all future sheets will be defined in xml files.

New in version 0.7.1dev1 (2016-09-14)

- A new sheet (declined in two versions): expansion of double brackets. Defined in an xml sheet as for mental calculation sheets.

New in version 0.7.0-6 (2016-08-19)

- Added a setting to let the user change mathmaker’s path (to be used by the daemon)

New in version 0.7.0-5 (2016-08-19)

- Bugfix

New in version 0.7.0-4 (2016-08-19)

- If an IP address is passed as parameter to mathmaker’s daemon, it will return a 429 http status code (too many requests) if the last request from the same address is not older than 10 seconds.

New in version 0.7.0-3 (2016-07-18)

- Fixed the install of locale files and font listing file

New in version 0.7 (2016-07-15)

- Standardized structure (`mathmaker` becomes pip3-installable, available on PyPI and github; its documentation is hosted on readthedocs; tests are made with py.test)
- A daemon is added (`mathmakerd`) to provide communication with `mathmaker` through http connections.
- A bunch of mental calculation sheets
- The use of XML frameworks for the sheets (yet only for mental calculation, so far)

Footnotes:

1. Complete list of recommended LaTeX packages:

CTAN Package Name	Package name (Ubuntu 14.04)
fontspec	texlive-latex-recommended
polyglossia	texlive-xetex
geometry	texlive-latex-base
graphicx	texlive-pstricks
epstopdf	texlive-font-utils
tikz	texlive-latex-extra
amssymb	texlive-base
amsmath	texlive-latex-base
siunitx	texlive-science
cancel	texlive-pictures
array	texlive-latex-base
ulem	texlive-generic-recommended
textcomp	texlive-latex-base
eurosym	texlive-fonts-recommended
lxfonts	texlive-fonts-extra
multicol	texlive-latex-base

2. Using `pkg`, you'll have to install `texlive-full`; if you wish to install only the relevant LaTeX packages, you'll have to browse the ports, I haven't done this yet so cannot tell you exactly which ones are necessary.
3. Complete list of provided sheets:

Theme	Subtheme	Directive name
algebra		algebra-balance-01
algebra		algebra-binomial-identities-expansion
algebra		algebra-expression-expansion
algebra		algebra-expression-reduction
algebra		algebra-factorization-01
algebra		algebra-factorization-02
algebra		algebra-factorization-03
algebra		algebra-mini-test-0
algebra		algebra-mini-test-1
algebra		algebra-short-test

Continued on next page

Table 1.1 – continued from previous page

Theme	Subtheme	Directive name
algebra		algebra-test-2
algebra	equations	equations-basic
algebra	equations	equations-classic
algebra	equations	equations-harder
algebra	equations	equations-short-test
algebra	equations	equations-test
geometry	right triangle	converse-and-contrapositive-of-pythagorean-theorem-short-test
geometry	right triangle	pythagorean-theorem-short-test
mental_calculation	lev11_1	divisions
mental_calculation	lev11_1	mini_problems
mental_calculation	lev11_1	multi_11_15_25
mental_calculation	lev11_1	multi_decimal
mental_calculation	lev11_1	multi_hole_any_nb
mental_calculation	lev11_1	multi_hole_tables2_9
mental_calculation	lev11_1	multi_reversed
mental_calculation	lev11_1	ranks
mental_calculation	lev11_1	tables2_9
mental_calculation	lev11_1	test_11_1
mental_calculation	lev11_2	multi_divi_10_100_1000
mental_calculation	lev11_2	operations_vocabulary
mental_calculation	lev11_2	polygons_perimeters
mental_calculation	lev11_2	rectangles
mental_calculation	lev11_2	test_11_2
numeric calculation	fractions	fraction-simplification
numeric calculation	fractions	fractions-product-and-quotient
numeric calculation	fractions	fractions-sum

Advanced features

User settings

The default settings are following:

```
PATHS:
    EUKTOEPS: euktoeps
    XMLLINT: xmllint
    LUALATEX: lualatex
    LUAOTFLOAD_TOOL: luaotfload-tool
    MSGFMT: msgfmt
    OUTPUT_DIR: ./

LOCALES:
    # Available values can be checked in the locale directory.
    LANGUAGE: en_US
    ENCODING: UTF-8
    # Values can be 'euro', 'sterling', 'dollar'
    CURRENCY: dollar

LATEX:
    FONT:
```

```
    ROUND_LETTERS_IN_MATH_EXPR: False
```

Some explanations:

- The `PATHS:` section is here to provide a mean to change the paths to `euktoeps`, `lualatex` and `xmllint` mainly. In case one of them is not reachable the way it is set in this section, you can change that easily.
- The `PATHS:` section contains also an `OUTPUT_DIR:` entry. This is the directory where `mathmaker` will store the possible picture files (.euk and .eps). Default value is "current directory". You can set another value, at your liking.
- The entries under `LOCALES:` allow to change the language, encoding, and default currency used.
- The `LATEX:` section contains an entry to set the font to use (be sure it is available on your system). The `ROUND_LETTERS_IN_MATH_EXPR:` entry is disabled by default (set to False). If you set it to True, a special font will be used in math expressions, that will turn all letters (especially the 'x') into a rounded version. This is actually the `lxfonts` LaTeX package. It doesn't fit well with any font. Using "Ubuntu" as font and setting `ROUND_LETTERS_IN_MATH_EXPR:` to True gives a nice result though.

Your settings file must be `~/.config/mathmaker/user_config.yaml`.

Command-line options

Several command-line options correspond to settings that are defined in `~/.config/mathmaker/user_config.yaml`. Any option redefined in command-line options will override the setting from the configuration file.

Type `mathmaker --help` to get information on these command-line options.

http server (mathmakerd)

Once everything is installed, it's possible to run a server to communicate with mathmaker through a web browser.

Run the server:

```
$ mathmakerd start
```

Then go to your web browser and as url, you can enter:

```
http://127.0.0.1:9999/?sheetname=<sheetname>
```

and replace `<sheetname>` by an available sheet's name (from `mathmaker list`), for instance:

```
http://127.0.0.1:9999/?sheetname=pythagorean-theorem-short-test
```

`mathmaker` will create the new sheet, compile it and return the pdf result to be displayed in the web browser.

At the moment, `mathmakerd stop` doesn't work correctly, you'll have to `kill` it directly (`ps aux | grep mathmakerd` then `kill` with the appropriate pid).

It's possible to pass an IP address in an extra parameter named `ip`, like:

> http://127.0.0.1:9999/?sheetname=pythagorean-theorem-short-test&ip=127.0.0.1

In this case, `mathmakerd` will check if the last request is older than 10 seconds (this is hardcoded, so far) and if not, then a http status 429 will be returned. In order to do that, `mathmakerd` uses a small database that it erases when the last request is older than one hour (also hardcoded, so far).

xml sheets

As a directive to mathmaker it is possible to give a path to an xml file.

Creating a new xml file that can be used as a model by `mathmaker` is more for advanced users, though it's not that difficult.

At the moment (version 0.7), it's only possible to create mental calculation sheets this way. So there's not much to change from a raw model.

Let's have a look at `mathmaker/data/frameworks/mental_calculation/lev11_1/divisions.xml`:

```
<?xml version="1.0" encoding="UTF-8"?>

<sheet header="" title="Mental calculation" subtitle="Divisions" text="" answers_
↪title="Answers">

    <!-- Default values: type="std" unit="cm" font_size_offset="0" -->
    <!-- Available layout types: std|short_test|mini_test|equations|mental -->
    <layout type="mental" font_size_offset="-1">
            <exc>
                    <line nb="None">
                            <exercises>all</exercises>
                    </line>
            </exc>
            <ans>
                    <line nb="None">
                            <exercises>all</exercises>
                    </line>
            </ans>
    </layout>

    <!-- Default value: id='generic'
             No default for kind and subkind, they must be given -->
    <!-- Available kinds for mental calculation: tabular, slideshow -->
    <exercise id="mental_calculation" kind="tabular">

            <!--No default for kind and subkind, they must be given -->
            <question kind="divi" subkind="direct">
                    <nb source="intpairs_2to9">20</nb>
            </question>

    </exercise>

</sheet>
```

The `<sheet>` tag has attributes that let you easily change the title of the sheet, a subtitle etc.

The `<layout>` part can't be changed (yet) except the `unit` and `font_size_offset` attributes. The later one is especially practical to resize the whole sheet at once.

The `<exercise>` part is the one you can change alot. Keep the `id="mental_calculation"` and `kind="tabular"` attributes though (they can't be changed yet) but you can put the questions you like inside.

Each question is defined this way:

```
<question kind="divi" subkind="direct">
    <nb source="intpairs_2to9">20</nb>
</question>
```

You must set at least a `kind` and a `subkind` attributes. Then inside the question, you set at least one numbers' source. This question says: "I want 20 questions about direct division (i.e. each one will be of the form a ÷ b = ?) the numbers being integers between 2 and 9". (For divisions the pair of integers will be b and the solution; mathmaker will compute a automatically).

Another example, taken from `mathmaker/data/frameworks/mental_calculation/lev11_1/mini_problems.xml`:

```
<question kind="addi" subkind="direct" context="mini_problem">
    <nb source="intpairs_5to20">5</nb>
</question>
```

You see you can set the lower and upper values as you like. Just respect the syntax (if you write `intpairs_5_to_20` this won't work). And this time a context is added to the question. So it means "I want 5 simple additive problems, the numbers being integers between 5 and 20".

Note that you can put several different numbers' sources inside one `<question>`. For instance:

```
<question kind="multi" subkind="direct">
    <nb source="intpairs_2to9">1</nb>
    <nb source="table_11">1</nb>
    <nb source="decimal_and_one_digit">1</nb>
</question>
```

This means there will be three questions, all being direct multiplications, but one pair of numbers will be integers between 2 and 9; one pair will be from the table of 11 (like 34 × 11), and one will be a decimal number and a one digit number (like 150,3 × 0.01).

Last explained feature: in some sheets you'll find `<mix>` sections, like this one, taken from `mathmaker/data/frameworks/mental_calculation/lev11_2/test_11_2.xml`:

```
<mix>
    <question kind="area" subkind="rectangle" picture="true"></question>
    <question kind="multi" subkind="direct"></question>
    <question kind="multi" subkind="direct"></question>
    <question kind="vocabulary" subkind="multi"></question>
    <nb source="table_15">1</nb>
    <nb source="table_11">1</nb>
    <nb source="intpairs_2to9" variant="decimal1">1</nb>
    <nb source="intpairs_2to9" variant="decimal2">1</nb>
</mix>
```

It means the numbers' sources will be randomly attributed to the questions. Each time a new sheet is generated from this framework, the numbers from table of 15 will be attributed . The rules to follow for a `<mix>` section are:

- Put as many numbers' sources as there are questions. For instance in the example above we could have written this too:

```
<mix>
    <question kind="area" subkind="rectangle" picture="true"></question>
    <question kind="multi" subkind="direct"></question>
    <question kind="multi" subkind="direct"></question>
    <question kind="vocabulary" subkind="multi"></question>
    <nb source="table_15">3</nb>
    <nb source="intpairs_2to9" variant="decimal1">1</nb>
</mix>
```

- Any numbers' source must be assignable to any of the questions.

Now the question is: how to know about the questions kinds and subkinds, and the possible contexts, variants or whatever other attributes? Well it is planned to add an easy way to know that (like a special directive) but there's nothing yet. The better, so far, may be to look at the provided sheets in `mathmaker/data/framworks/mental_calculation/` and see what's in there.

FreeBSD notes

eukleides fix

`eukleides` currently does not work out of the box. The pkg-installed version has a functional euktoeps script, it is required, so keep it somewhere. Then do `pkg remove eukleides` and re-install it from source:

- get the 1.5.4 source from http://www.eukleides.org/, for instance `wget http://www.eukleides.org/files/eukleides-1.5.4.tar.bz2`
- then `tar xvzf eukleides-1.5.4.tar.bz2`
- then possibly modify the prefix for install in the `Config` file, at your liking
- remove the making of documentation and manpages from the `install` target in the `Makefile` (they cause errors)
- install the required dependencies to compile eukleides: `pkg install bison flex gmake gcc`
- do `gmake` and then `gmake install`. This will provide functional binaries.
- replace the euktoeps script by the one you did get from the pkg installed version.
- if necessary (if `lualatex` complains about not finding `eukleides.sty`), reinstall `eukleides.sty` and `eukleides.tex` correctly:

```
# mkdir /usr/local/share/texmf-dist/tex/latex/eukleides
# cp /usr/local/share/texmf/tex/latex/eukleides/eukleides.* /usr/local/
↪share/texmf-dist/tex/latex/eukleides/
# mktexlsr
```

python-daemon error at install

You might get an error before the end of `mathmaker`'s installation:

```
/usr/local/venv/mm0/lib/python3.4/site-packages/setuptools/dist.py:294: UserWarning:␣
↪The version specified ('UNKNOWN') is an invalid version, this may not work as␣
↪expected with newer versions of setuptools, pip, and PyPI. Please see PEP 440 for␣
↪more details.
  "details." % self.metadata.version
creating /usr/local/venv/mm0/lib/python3.4/site-packages/python_daemon-UNKNOWN-py3.4.
↪egg
Extracting python_daemon-UNKNOWN-py3.4.egg to /usr/local/venv/mm0/lib/python3.4/site-
↪packages
Adding python-daemon UNKNOWN to easy-install.pth file
Installed /usr/local/venv/mm0/lib/python3.4/site-packages/python_daemon-UNKNOWN-py3.4.
↪egg
error: The 'python-daemon>=2.1.1' distribution was not found and is required by␣
↪mathmaker
[mm0] root@testmm0:/usr/local/src/mathmaker #
```

Fix it this way:

```
# pip3 install python-daemon --upgrade
```

And finish the install:

```
# pip3 install mathmaker
```

CHAPTER 2

Developer's documentation

Guided tour

Foreword

This code has been developed chunk by chunk over more than 10 years now, starting with python2.3 or 4. Now it is a python3.4 software and my python skills have fortunately improved over the years. Problem is that several old parts, even after big efforts to correct the worst pieces, are not very idiomatic, especially the core.

Luckily there are unit tests, and whatever one might think about them, it's not difficult to admit that they're extremely useful to check nothing got broken when the core parts are written anew or debugged.

The documentation has been originately written using doxygen. Despite the fact it is an excellent documentation software, I have decided to move to Sphinx because it corresponds closer to python best practices. So, all doxygen-style comments will be turned into docstrings so mathmaker can use Sphinx to build the documentation. At the moment this work is just started, so the auto-generated Sphinx documentation is quite uncomplete now.

So, a part of the work to do is surely to bring new features, but another part, more annoying, is to turn ugly old parts into the right pythonic idioms. That's why at places you'll see that this or this module is deprecated and should be "reduced", or rewritten. A list of such things to do is available on sourceforge. The 1.0 milestone inventories many things.

The issue

It is utmost important to understand that mathmaker is not a software intended to compute mathematical stuff, but to display it. For instance, resolving a first-degree equation is not in itself a goal of mathmaker, because other softwares do that already (and we don't even need any software to do it). Instead, mathmaker will determine and display the steps of this resolution. Sometimes, mathmaker solutions will even try to mimic the pupils' way of doing things.

For instance, it won't automatically simplify a fraction to make it irreducible in one step, but will try to reproduce the steps that pupils usually need to simplify the fraction. So the GCD is only used to check when the fraction is irreducible and for the cases where there's no other choice, but not as the mean to simplify a fraction directly (not before pupils learn how to use it, at least).

Another example is the need of mathmaker to control the displaying of decimal and integer numbers perfectly. Of course, most of the time, it doesn't matter if a computer tells that 5.2×5.2 = 27.040000000000003 or 3.9×3.9 = 15.209999999999999 because everyone knows that the correct results are 27.04 and 15.21 and because the difference is not so important, so in many situations, this precision will be sufficient. But, can mathmaker display to pupils that the result of 5.2×5.2 is 27.040000000000003 ?

Also, the human rules we use to write maths are full of exceptions and odd details we don't notice usually because we're familiar to them. We would never write

$+2x^2 + 1x - 1(+5 - 1x)$

but instead

$2x^2 + x - (5 - x)$

There are many conventions in the human way to write maths and many exceptions.

These are the reasons why the core is quite complex: re-create these writing rules and habits on a computer and let the result be readable by pupils is not an easy thing.

Workflow

Mathmaker creates Sheets of maths Exercises.

Each Exercise contains Questions.

Each Question uses objects from the core, that embbed enough information to compute and write the text of the Question and also the answer.

The main executable (`entry_point()` in `mathmaker/cli.py`) performs following steps:

- Load the default settings from configuration files.
- Setup the main logger.
- Check that the correct dependencies are installed.
- Parse the command-line arguments, updates the settings accordingly.
- Install the language and setup shared objects, like the database connection.
- If the main directive is `list`, it just write the directives list to stdout
- Otherwise, it checks that the directive matches a known sheet (either a xml file or a sheet's name that mathmaker provides) and writes the result to the output (`stdout` or a file)

The directories

At the root can be found:

- The usual `docs/` and `tests/` directories
- `mathmaker/` contains the actual python source code
- `tools/` contains several standalone scripts that are useful for developers only (not users)
- Several usual files (`.flake8` etc.)

`mathmaker/`'s content:

- `data/` this is where the database is stored, but also xml files containing additional wordings, translations etc.
- `lib/` contains all useful classes and submodules (see below).

- `locale/` contains all translation files.
- `settings/` contains the functions dedicated to setup the settings and also the default settings files themselves.

`lib/`'s content:

- `common/` contains several constants (should be reduced or disappear; `pythagorean.py` is especially meant to be included in the database)
- `core/` contains all mathematical objects, numeric or geometric
- `machine/` contains the "typewriter"
- `sheet/` contains all sheets, exercices and questions. A big part of it is obsolete (should be replaced by generic objects that take their data from xml files)
- `tools/` contains modules relating to a special feature (configuration, database handling, etc.)
- `error.py` contains mathmaker's own errors (actually still contains a lot of useless Exceptions, needs to be reduced and reorganized)
- `is_.py` contains special functions to determine the type of certain objects (this should be removed)
- `list_sheets.py` contains the functions used to build the complete list of the sheets that mathmaker provides
- `maths_lib.py` contains some extra mathematical functions
- `randomly.py` contains some functions dealing with randomness (should be heavily reduced because the standard random module already provides almost everything)
- `shared.py` contains objects and variables that need to be shared (except settings), like the database connection
- `sources.py` contains the functions to interact with numbers' or wordings' sources
- `startup_actions.py` contains several functions called at startup

Overview of the main classes

A Machine is like a typewriter: it turns all printable objects (Sheets, and everything they contain) into LaTeX. It knows how to turn a mathematical expression in LaTeX format. It knows how to draw figures from the geometrical objects (using eukleides).

The Sheet objects given to a Machine contain guidelines for the Machine: the layout of the Sheet and what Exercises it contains.

The Exercise objects contains Questions and also layout informations that might be specific to the exercise (for instance, display the equations' resolutions in two columns).

The Question objects contains the mathematical objects from the core and uses them to compute texts and answers.

The objects from the core are all different kinds of mathematical objects, like Sums, Products, Equations or Triangles, Tables... For instance, a Question about Pythagora's theorem would embed a RightTriangle (which itself embeds information on its sides, vertices, angles; and enough methods to create a picture of it) but also fields telling if the figure should be drawn in the Question's text or if only a description of the figure should be given; if the hypotenuse should be calculated or another side; if the result should be a rounded decimal and how precise it should be etc.

When a new Sheet is created, all objects it contains are created randomly, following some rules, though, to avoid completely random uninteresting results.

More details about the core objects a little bit below, in the paragraph about *The core*.

Start working on mathmaker

Short version

Install dependencies:

- Ubuntu:

```
$ sudo apt-get install eukleides libxml2-utils gettext texlive-full
```

- FreeBSD:

```
$ sudo pkg install python34 py34-sqlite3 gettext eukleides libxml2 texlive-full
$ rehash
```

And FreeBSD users should check the *eukleides fix*

To install mathmaker in dev mode in a venv, get to the directory where you want to work, and (assuming git and python3.4 are installed):

- Ubuntu:

```
$ pyvenv-3.4 dev0
$ source dev0/bin/activate
(dev0) $ pip3 install pytest tox flake8 pydocstyle sphinx sphinx-autodoc-
↪annotation sphinx-rtd-theme
(dev0) $ mkdir mathmaker
(dev0) $ cd mathmaker/
(dev0) $ git clone https://github.com/nicolashainaux/mathmaker.git
(dev0) $ python3 setup.py develop
```

- FreeBSD:

```
$ pyvenv-3.4 dev0
$ source dev0/bin/activate.csh
[dev0] $ sudo pip3 install pytest tox flake8 pydocstyle sphinx sphinx-autodoc-
↪annotation sphinx-rtd-theme
[dev0] $ mkdir mathmaker
[dev0] $ cd mathmaker/
[dev0] $ git clone https://github.com/nicolashainaux/mathmaker.git
[dev0] $ python3 setup.py develop
```

Usage: get to an empty directory and:

```
(dev0) $ mathmaker test_11_2 > out.tex
(dev0) $ lualatex out.tex
```

You can check `out.pdf` with the pdf viewer you like.

Run the tools:

```
(dev0) $ cd path/to/mathmaker/tools/
(dev0) $ ./build_db.py
(dev0) $ ./update_pot_files
```

Most of the tests are stored under `tests/`. Some others are doctests. Any new test or doctest will be added automatically to the tests run by `py.test` or `tox`.

Run the tests:

```
(dev0) $ py.test
(dev0) $ tox
```

Tox will ignore missing python interpreters.

Edit the settings:

```
(dev0) $ cd path/to/mathmaker/settings/
(dev0) $ mkdir dev/
(dev0) $ cp default/*.yaml dev/
```

In `dev/logging.yaml` you can set the `__main__` logger to `INFO` (take care to define log rotation for `/var/log/mathmaker`). Set the dbg logger to `DEBUG`.

Each debugging logger can be enabled/disabled individually in `debug_conf.yaml` (by setting it to `DEBUG` or `INFO`).

See *Loggers: main, daemon, debugging* for more details on how to setup new loggers (and debugging loggers).

You can override settings in `dev/user_config.yaml` to your liking.

Before starting, you should read at least the *Auxiliary tools* and *Writing rules* sections. It is certainly worth also to have a look at *Advanced features*.

Hope you'll enjoy working on mathmaker!

Detailed version

Dev environment

Install external dependencies

You'll need to install the same dependencies as users do (see *Install*). In addition, `xgettext` is required to extract the gettext messages from py files. In Ubuntu 14.04 it's in the `gettext` package.

Get mathmaker's source code from github repo

In the folder of your choice:

```
$ git clone https://github.com/nicolashainaux/mathmaker.git
```

Setup a python virtual environment

It is strongly advised to install mathmaker in develop mode inside of a python virtual environment. This allows to install the required libraries without conflicting with other projects or python software on the same computer. So, in addition to the packages required for mathmaker to work (see Quickstart), you'd better install `python3.4-venv` and work in a virtual environment dedicated to mathmaker. So, just get to the directory of your choice, and to create a virtual environment named `dev0`, you type:

```
$ pyvenv-3.4 dev0
```

From there, you can activate it:

on Ubuntu:

```
$ source dev0/bin/activate
```

on FreeBSD:

```
$ source dev0/bin/activate.csh
```

Install mathmaker

Once your virtual environment is activated, go to mathmaker's root:

```
(dev0) $ cd path/to/mathmaker/
```

You should see something like:

```
(dev0) $ ls
CHANGELOG.rst  docs  LICENSE  MANIFEST.in  mathmaker README.md  README.rst
↪requirements.txt  setup.py  tests  tools  tox.ini
```

There you can install mathmaker in developer mode:

```
(dev0) $ python3 setup.py develop
```

It's possible to clean the project's main directory:

```
(dev0) $ python3 setup.py clean
```

Run mathmaker and tools

From now on, it is possible to run `mathmaker` from your virtual environment. As `mathmaker` is installed in developer mode, any change in the source files will be effective when running `mathmaker`. Go to a directory where you can leave temporary files (each sheet requiring pictures will produce picture files, by default), and test it:

```
(dev0) $ cd path/to/garbage/directory/
(dev0) $ mathmaker test_11_2 > out.tex
(dev0) $ lualatex out.tex
```

You can check `out.pdf` with the pdf viewer you like.

You can also run the tools:

```
(dev0) $ cd path/to/mathmaker/
(dev0) $ cd tools/
(dev0) $ ./build_db.py
(dev0) $ ./update_pot_files
```

Somewhat below, more informations about the *Auxiliary tools*.

Once you're done working with mathmaker, you can deactivate the virtual environment:

```
(dev0) $ deactivate
$
```

Note that it is possible to run `mathmaker` outside the virtual environment this way:

```
$ cd path/to/mathmaker/
$ python3 -m mathmaker.cli
```

But it requires to have installed the python dependencies yourself on the host system (e.g. the computer) and maybe also to have set `$PYTHONPATH` correctly (and exported it).

Other dependencies

Linters

It is recommended to install linters for PEP 8 and PEP 257 (see *Writing rules*):

```
(dev0) $ pip3 install flake8
(dev0) $ pip3 install pydocstyle
```

Test dependencies

In addition you should install at least `py.test`, and also `tox` if you intend to run tox tests:

```
(dev0) $ pip3 install pytest
(dev0) $ pip3 install tox
```

Below is more information about *testing*.

Documentation dependencies

You'll need to install these dependencies in the virtual environment:

```
(dev0) $ pip3 install sphinx sphinx-rtd-theme
```

`sphinx-rtd-theme` is the theme used for mathmaker's documentation. It's the readthedocs theme.

Note: `sphinx-autodoc-annotation` makes writing docstrings lighter when using python3 annotations. Problem is, this package currently has a bug that prevents to build the doc on readthedocs.

Below is more information about *documentation*.

Dev settings

You can make a copy of the default configuration files:

```
(dev0) $ cd path/to/mathmaker/
(dev0) $ cd settings/
(dev0) $ mkdir dev/
(dev0) $ cp default/*.yaml dev/
```

Then you can edit the files in `mathmaker/settings/dev/` to your liking. Any value redefined there will override all other settings (except the options from the command line).

In `logging.yaml` the loggers part is interesting. I usually set the `__main__` logger to `INFO` (this way, informations about starting and stopping mathmaker are recorded to `/var/log/mathmaker`, take care to define the log rotation if you do so) and the dbg logger to `DEBUG`. This second setting is important because it will allow to enable debugging loggers in `debug_conf.yaml`.

`debug_conf.yaml` allows to trigger each debugging logger individually by setting it to `DEBUG` instead of `INFO`.

And in `user_config.yaml` it is especially nice to define an output directory where all garbage files will be stored, but also to set the language, the font etc.

For instance, my `settings/dev/user_config.yaml` contains this:

```
# SOFTWARE'S CONFIGURATION FILE

PATHS:
    OUTPUT_DIR: /home/nico/dev/mathmaker/essais/poubelle/dev2/

LOCALES:
    LANGUAGE: fr_FR
    CURRENCY: euro

LATEX:
    FONT: Ubuntu
    ROUND_LETTERS_IN_MATH_EXPR: True
```

See *Settings* to learn more about the way settings are handled by `mathmaker`.

Testing

Run the tests

The testing suite is run by py.test this way:

```
(dev0) $ py.test
```

or this way:

```
(dev0) $ python3 setup.py test
```

Where do they live?

Most of the tests belong to `tests/`. Any function whose name starts with `test_` written in any python file whose name also starts with `test_` (and stored somewhere under `tests/`) and will be automatically added to the tests run by `py.test`.

Some more tests are written as doctests (see also pytest documentation about doctests) in the docstrings of the functions. It's possible to add doctests, especially for simple functions (sometimes it is redundant with the tests from `tests/`, but this is not a serious problem). The configuration for tests is so that any new doctest will be automatically added to the tests run by `py.test`.

Tox

To test `mathmaker` against different versions of python, you can run tox this way:

```
(dev0) $ tox
```

or this way:

```
(dev0) $ python3 setup.py tox
```

Be sure you have different versions of python installed correctly on your computer before starting this. The missing versions will be skipped anyway. Note that it is not a purpose of `mathmaker` to run under a lot of python versions (several python3 versions are OK, but no support for python2 is planned, unless someone really wants to do that).

Loggers: main, daemon, debugging

See *Dev settings* to know how to use the settings files and enable or disable logging and debugging.

The two interesting loggers are `__main__` and `dbg`.

Main logger

`__main__` is intended to be used for messages relating to `mathmaker` general working. In particular, it should be used to log any error that forces `mathmaker` to stop, before it stops.

In order to use this `__main__` logger, you can write this at the start of any function (assuming you have imported settings at the top of the file):

```
log = settings.mainlogger
```

And then inside this function:

```
log.error("message")
```

(or `log.warning("message")` or `log.critical("message")` depending on the severity level).

If an Exception led to stop `mathmaker`, then the message should include its Traceback (if you notice this is not the case somewhere, you can modify this and make a pull request). For instance in `cli.py`:

```
try:
    shared.machine.write_out(str(sh))
except Exception:
    log.error("An exception occured during the creation of the sheet.",
              exc_info=True)
    shared.db.close()
    sys.exit(1)
```

Daemon logger

This logger is intended to be used by the daemon script. Works the same way as the main logger.

Debugging logger

`dbg` is the logger dedicated to debugging and ready to use. No need to write `sys.stderr.write(msg)` anywhere.

If there's no logger object in the function you want to print debugging messages, you can create one this way:

- Add the matching entry in `debug_conf.yaml` (both the `settings/default/` and `settings/dev/` versions, but set to `INFO` in the `settings/default/` version). For short modules, you can add only one level, and for modules containing lots of functions of classes, two levels should be added, like the example of the extract below:

```
dbg:
    db: INFO
    wording:
        merge_nb_unit_pairs: INFO
        setup_wording_format_of: INFO
        insert_nonbreaking_spaces: INFO
    class_or_module_name:
        fct: DEBUG
```

- Import the settings at the top of the file, if it's not done yet:

```
from mathmaker import settings
```

- Create the logger at the start of the function (i.e. locally):

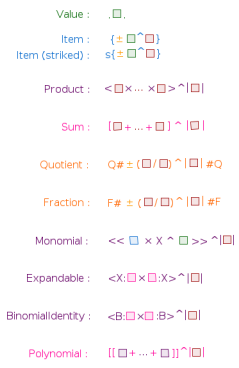
```
def fct():
    log = settings.dbg_logger.getChild('class_or_module_name.fct')
```

- Then where you need it, inside `fct`, write messages this way:

```
log.debug("the message you like")
```

Later when you need to disable this logger, you just set it to `INFO` instead of `DEBUG` in `settings/dev/debug_conf.yaml`. See *Dev settings* for information on these files.

A summary of the conventions used to represent the different core objects (i.e. what their `__repr__()` returns):



System log configuration

Systems using `rsyslog`, like Ubuntu

Ensure `/etc/rsyslog.conf` contains:

```
$IncludeConfig /etc/rsyslog.d/*.conf
```

Then create (if not created yet) a 'local' configuration file, like: `/etc/rsyslog.d/40-local.conf` and put (or add) in it:

```
#  Local user rules for rsyslog.
#
#
local5.*                        /var/log/mathmaker.log
local6.*                        /var/log/mathmakerd.log
```

Then save it and:

```
# service rsyslog restart
```

Warning: Do not create `/var/log/mathmaker.log` yourself with the wrong rights, otherwise nothing will be logged.

To format the messages in a nicer way, it's possible to add this in /etc/rsyslog.conf:

```
$template MathmakerTpl,"%$now% %timegenerated:12:23:date-rfc3339% %syslogtag%%msg%\n"
```

and then, modify /etc/rsyslog.d/40-local.conf like:

```
local5.*                          /var/log/mathmaker.log;MathmakerTpl
local6.*                          /var/log/mathmakerd.log;MathmakerTpl
```

Tools to check everything's fine: after having restarted rsyslog, enable some more informations output:

```
# export RSYSLOG_DEBUGLOG="/var/log/myrsyslogd.log"
# export RSYSLOG_DEBUG="Debug"
```

and running the configuration validation:

```
# rsyslogd -N2 | grep "mathmaker"
```

should show something like (errorless):

```
rsyslogd: version 7.4.4, config validation run (level 2), master config /etc/rsyslog.
↪conf
2564.153590773:7f559632b780:   ACTION 0x2123160 [builtin:omfile:/var/log/mathmaker.
↪log;MathmakerTpl]
2564.154126386:7f559632b780:   ACTION 0x2123990 [builtin:omfile:/var/log/mathmakerd.
↪log;MathmakerTpl]
2564.158461309:7f559632b780:   ACTION 0x2123160 [builtin:omfile:/var/log/mathmaker.
↪log;MathmakerTpl]
2564.158729012:7f559632b780:   ACTION 0x2123990 [builtin:omfile:/var/log/mathmakerd.
↪log;MathmakerTpl]
rsyslogd: End of config validation run. Bye.
```

Once you've checked this works as expected, do not forget to configure your log rotation.

Documentation

Current state

As stated in the *Foreword*, the documentation is being turned from doxygen to Sphinx, so there are missing parts .

Any new function or module has to be documented as described in PEP 257.

The doxygen documentation for version 0.6 is here. The core parts are still correct, so far.

Format

This documentation is written in ReStructured Text format.

There are no newlines inside paragraphs. Set your editor to wrap lines automatically to your liking.

Make html

To produce the html documentation:

```
(dev0) $ cd docs/
(dev0) $ make html
```

Auxiliary tools

Several standalone scripts live in the `tools/` directory under root. They can be useful for several tasks that automate the handling of data.

The two most useful ones are both meant to be run from the `tools/` directory. They are:

- `build_db.py`, what is used to update the database when there are new entries to add in it. If new words of 4 letters are added to any po file, `build_db.py` should be run, it will add them to the database. If new wordings are entered in `mathmaker/data/wordings/*.xml`, then it should be run too. See details in the docstring. And if a new table is required, it should be added in this script. For instance, the pythagorean triples should live in the database and will be added to this list soon or later.
- `update_po_files`, what is a shell script making use of `xgettext` and of the scripts `merge_py_updates_to_main_pot_file` and `merge_xml_updates_to_pot_file`. Run `update_po_files` to update `locale/mathmaker.pot` when new strings to translate have been added to python code (i.e. inside a call to `_()`) or new entries have been added to any xml file from `mathmaker/data` (only entries matching a number of identifiers are taken into account, see DEFAULT_KEYWORDS in the source code to know which ones exactly).

`import_msgstr` and `retrieve_po_entries` are useful on some rare occasions. See their docstrings for more explanations. They both have a `--help` option.

`pythagorean_triples_generator` shouldn't be of any use any more (later on maybe a part of its code will be incorporated to `build_db.py`, that's why it's still around here)

Writing rules

It is necessary to write the cleanest code possible. It has not been the case in the past, but the old code is updated chunk by chunk and **any new code portion must follow python's best practices**, to avoid adding to the mess, and so, must:

- Use idioms (to learn some, it is recommended to read Jeff Knupp's Writing Idiomatic Python)
- Conform to the PEP 8 – Style Guide for Python
- Conform to the PEP 257 – Docstring Conventions

And of course, all the code is written in english.

As to PEP 8, mathmaker 's code being free from errors, the best is to use a linter, like `flake8`. They also exist as plugins to various text editors or IDE (see *Atom packages* for instance). Three error codes are ignored (see `.flake8`):

- E129 because it is triggered anytime a comment is used to separate a multiline conditional of an `if` statement from its nested suite. A choice has been made to wrap multiline conditions in `()` and realize the separation with next indented block using a `#  ___` comment (or any other comment if it's necessary) and this complies with PEP 8 (second option here):

  Acceptable options in this situation include, but are not limited to:

  ```
  # No extra indentation.
  if (this_is_one_thing and
      that_is_another_thing):
      do_something()

  # Add a comment, which will provide some distinction in editors
  # supporting syntax highlighting.
  if (this_is_one_thing and
      that_is_another_thing):
      # Since both conditions are true, we can frobnicate.
      do_something()
  ```

- W503 because PEP 8 does not compel to break before binary operators (the choice of breaking *after* binary operators has been done).
- E704 because on some occasions it is OK to put several *short* statements on one line in the case of `def`. It is the case in several test files using lines like `def v0():  return Value(4)`

Other choices are:

- A maximum line length of 79
- Declare _ as builtin, otherwise all calls to `_()` (i.e. the translation function installed by gettext) would trigger flake8's error F821 (undefined name).
- No complexity check. This might change in the future, but the algorithms in the core are complex. It's not easy to make them more simple (if anyone wants to try, (s)he's welcome).
- Name modules, functions, instances, and other variables in lower case, whenever possible using a single `word` but if necessary, using `several_words_separated_with_underscores`.
- Name classes in CapitalizedWords, like: `SuchAWonderfullClass` (don't use mixedCase, like `wrongCapitalizedClass`).
- All `import` statements must be at the top of any module. It must be avoided to add `from ... import ...` at the top of some functions, but sometimes it's necessary. A solution to avoid this is always preferred.
- All text files (including program code) are encoded in UTF-8.

As to PEP 257, this is also a good idea to use a linter, but lots of documentation being written as doxygen comments, the linter will detect a lot of missing docstrings. Just be sure the part you intend to push does not introduce new PEP 257 errors (their number must decrease with time, never increase).

The text of any docstring is marked up with reStructuredText.

The module *mathmaker.lib.tools.wording* can be considered as a reference on how to write correct docstrings. As an example, the code of two functions is reproduced here.

Note: The use of python3's annotations and `sphinx-autodoc-annotation` would automatically add the types (including return type) to the generated documentation. If `sphinx-autodoc-annotation`'s bug is corrected,

the `:type ...:  ...` and `:rtype:  ...` lines will be removed.

```
def cut_off_hint_from(sentence: str) -> tuple:
    """
    Return the sentence and the possible hint separated.

    Only one hint will be taken into account.

    :param sentence: the sentence to inspect
    :type sentence: str
    :rtype: tuple

    :Examples:

    >>> cut_off_hint_from("This sentence has no hint.")
    ('This sentence has no hint.', '')
    >>> cut_off_hint_from("This sentence has a hint: |hint:length_unit|")
    ('This sentence has a hint:', 'length_unit')
    >>> cut_off_hint_from("Malformed hint:|hint:length_unit|")
    ('Malformed hint:|hint:length_unit|', '')
    >>> cut_off_hint_from("Malformed hint: |hint0:length_unit|")
    ('Malformed hint: |hint0:length_unit|', '')
    >>> cut_off_hint_from("Two hints: |hint:unit| |hint:something_else|")
    ('Two hints: |hint:unit|', 'something_else')
    """
    last_word = sentence.split()[-1:][0]
    hint_block = ""
    if (is_wrapped(last_word, braces='||')
        and last_word[1:-1].startswith('hint:')):
        # __
        hint_block = last_word
    if len(hint_block):
        new_s = " ".join(w for w in sentence.split() if w != hint_block)
        hint = hint_block[1:-1].split(sep=':')[1]
        return (new_s, hint)
    else:
        return (sentence, "")


def merge_nb_unit_pairs(arg: object):
    r"""
    Merge all occurences of {nbN} {\*_unit} in arg.wording into {nbN\_\*_unit}.

    In the same time, the matching attribute arg.nbN\_\*_unit is set with
    Value(nbN, unit=Unit(arg.\*_unit)).into_str(display_SI_unit=True)
    (the possible exponent is taken into account too).

    :param arg: the object whose attribute wording will be processed. It must
      have a wording attribute as well as nbN and \*_unit attributes.
    :type arg: object
    :rtype: None

    :Example:

    >>> class Object(object): pass
    ...
    >>> arg = Object()
    >>> arg.wording = 'I have {nb1} {capacity_unit} of water.'
```

```
    >>> arg.nb1 = 2
    >>> arg.capacity_unit = 'L'
    >>> merge_nb_unit_pairs(arg)
    >>> arg.wording
    'I have {nb1_capacity_unit} of water.'
    >>> arg.nb1_capacity_unit
    '\\SI{2}{L}'
    """
```

Atom packages

This paragraph lists useful packages for atom users (visit the links to have full install and setup informations):

- `flake8` linter provider: linter-flake8 (Note: you should let the settings as is, except for the "Project config file" entry where you can write ".flake8" to use `mathmaker` project's settings.)
- `pydocstyle` linter provider: linter-pydocstyle
- python3's highlighter: MagicPython (MagicPython is able to highlight correctly python3's annotations. You'll have to disable the language-python core package.)
- To edit rst documentation: language-restructuredtext and rst-preview-pandoc

A deeper look in the source code

Settings

Everything happens in `mathmaker/settings/__init__.py` (it would be better to have everything happening rather in something like `mathmaker/settings/settings.py`, so this will most certainly change).

This module is imported by the main script at start, that run its `init()` function. After that, any subsequent `from mathmaker import settings` statement will make `settings.*` available.

The values shared as `settings.*` are: the paths to different subdirectories of the project, the loggers and the values read from *configuration* files. (Plus at the moment, two default values that should move to some other place).

None of these values is meant to be changed after it has been set by the main script, what calls `settings.init()` and then corrects some of them depending on the command-line options. Once this is done, these values can be considered actually as constants (they are not really constants as they are setup and corrected, so no UPPERCASE naming).

`tests/conftest.py `` uses the ``settings` module the same way `mathmaker/cli.py` does.

Configuration

This is handled by `mathmaker/lib/tools/config.py`. It works the same way for any of the `*.yaml` files. It first loads the default values from `mathmaker/settings/default/filename.yaml`. Then it updates any value found redefined in any of these files: `/etc/mathmaker/filename.yaml`, `~/.config/mathmaker/filename.yaml` and `mathmaker/settings/dev/filename.yaml`. Any missing file is skipped (except the first one: the default settings are part of the code, are shipped with it and must be present).

An extended dict class is used to deal easier with dicts created from yaml files. See `mathmaker/lib/tools/ext_dict.py`.

The daemon

It's a daemonized web server that allows to communicate with mathmaker through http requests. See *http server (mathmakerd)*.

Shared

Three resources are shared: *the database*, the LaTeX machine and the sources.

`mathmaker/lib/shared.py` works a similar way as the `settings` module. It is initialized once in the main script and then its resources are used.

The database

The aim of the database is to avoid having to create a lot of randomly values and store them in lists or dicts everytime we need something.

It is considered as a source among others.

The sources

They concern as well numbers as words or letters or anything one can think of. So far, they are used only for mental calculation, but they should be used for any kind of question.

When random numbers are required, most of the time, we don't need complete random. For instance if we want a pair of integers for the multiplication tables between 2 and 9, we don't want to ask the same question twice in a row.

The sources manage this randomness. Anytime we need to use a source, we can use its `next()` method to get the next random data, without worrying in the same time whether it's the same as the previous one or not.

So we have sources for names, for words having a limited number of letters, for different kinds of numbers but also for mini-problems wordings.

So far, there are two kinds of sources: the ones that are provided by the database, and the ones that are provided by the python function `generate_values()` from `mathmaker/lib/sources.py`.

All sources are initialized in `mathmaker/lib/shared.py`. There you can see which one has its values provided by the database, which are the other ones.

The database provides an easy way to ensure the next value will be different from the last one: we simply "stamp" each drawn value and the next time we draw a value among the yet unstamped ones. When they're all stamped, we reset all stamps and redraw. There's a tiny possibility to draw two times in a row the same value, so far, but it's so tiny we can safely ignore it. (This could be improved later). The values drawn from `generate_values()` are so different the ones from the others that it's very unlikely to draw the same ones two times in a row.

The real and the fake translation files

`mathmaker/locale/mathmaker.pot` is a real translation file.

The other `mathmaker/locale/*.pot` files are "fake" ones. They are used to get random words in a specific language, but the words do not need to be the exact translation from a source language.

For instance, `w4l.pot` contains words of four different letters. It wouldn't make sense to translate the english word "BEAN" into a word of the same meaning AND having exactly four different letters, in another language. This

wouldn't work for french, for instance. In general this would only work for rare exceptions (like "HALF" can be translated to "DEMI" in french).

The same applies to `feminine_names.pot` and `masculine_names.pot`. These files are used to get random names, but we don't need to *translate* them.

So, the entries in these "fake" translation files are only labeled entries, with nothing to translate.

A translator only needs to provide a number of entries (at least 10) in each of these files. No matter how many, no matter which `msgid` do they match. So: in `masculine_names.po` are several masculine names required, in `feminine_names.po` are several feminine names required and in `w4l.po` are several words of four unique letters required. Each time, at least 10, and then, the more the better.

The sheets, exercises and questions

I won't describe thoroughly all objects under `lib/sheet`, `lib/sheet/exercise` and `lib/sheet/exercise/question` because most of them are the old-style way of implementing this all.

Now the sheets should be frameworks stored as xml files (under `data/frameworks/`). Under `lib/sheet`, The classes `S_Structure` and `S_Generic` will be kept. `S_Structure` handles the layout of the sheet depending on the `SHEET_LAYOUT` dict you can find at the top of any sheet module, and that is built by `S_Generic` from the `<layout>` section of any xml framework.

There are no `X_Generic` nor `Q_Generic` classes yet, but there will be. They will replace the old-style `X_*` and `Q_*` classes. The way `X_MentalCalculation` and `Q_MentalCalculation` classes are written is a prefiguration of the future `X_Generic` and `Q_Generic` classes.

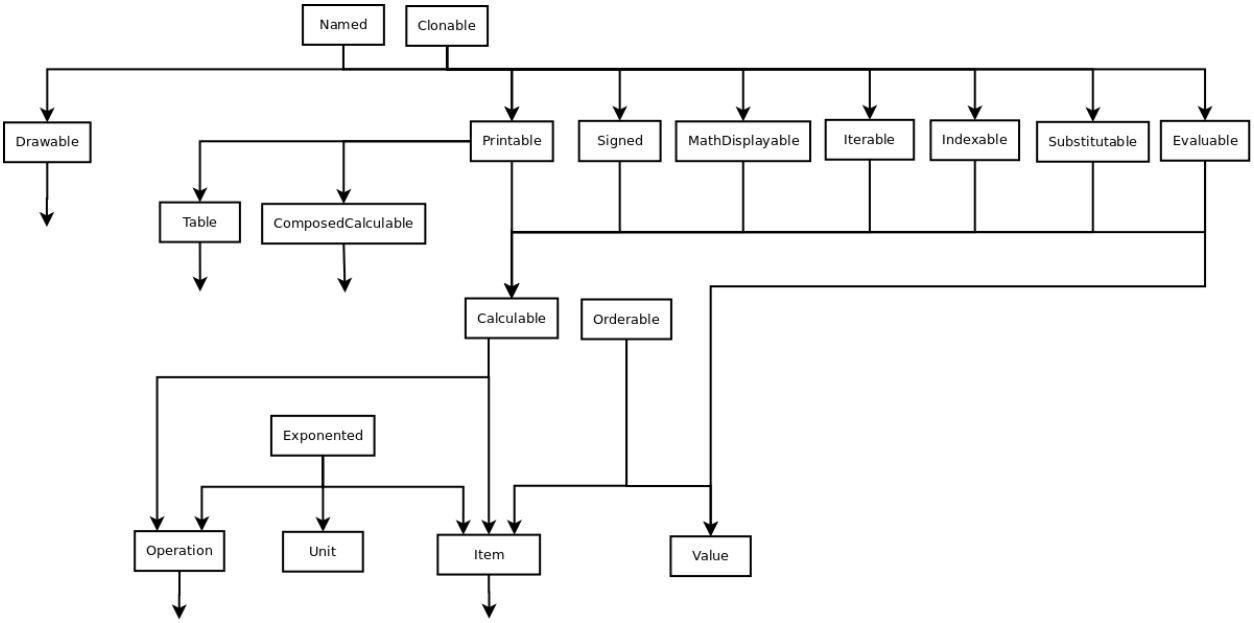
The `Q_MentalCalculation` class actually leaves the work to a `sub_object` that is written in one of the mental calculation modules (under `mc_modules/`). This allow a great variety of questions distributed in many files instead of one long file for all questions. These `sub_object``s also have a mother class (defined in ``mc_modules/mc_module.py`) and can be organized in subclasses (like vocabulary questions what all inherit from `vocabulary_questions.structure`).

The core

Diagram

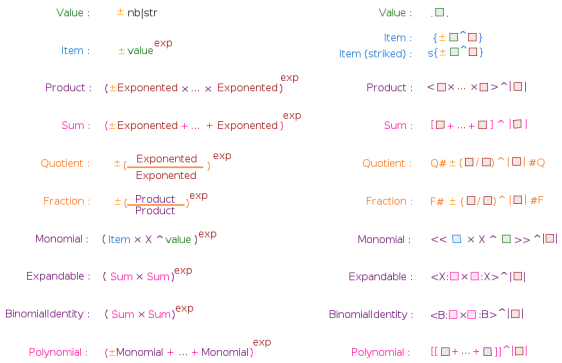
You can check the 0.6 version (i.e. from doxygen) of the top of the core diagram, though it will be somewhat changed later, it still can be used as reference for some time.

Unfinished draft of future plans:



Core objects' summary

Objects at left; associated `__repr()` at right:



Core objects' details

The "old" doc for 0.6 version is available here and mainly still correct for 0.7 version. When things will have settled down to something more stable, an updated documentation will be published chunk by chunk.

What can be done?

See the tickets on sourceforge and especially the ones for the 1.0 version.

mathmaker package

Subpackages

mathmaker.lib package

Subpackages

mathmaker.lib.common package

Submodules

mathmaker.lib.common.alphabet module

mathmaker.lib.common.cst module

mathmaker.lib.common.latex module

mathmaker.lib.common.pythagorean module

mathmaker.lib.common.pythagorean.**get_legs_matching_given_hypotenuse**(*side_length*)

mathmaker.lib.common.pythagorean.**get_legs_matching_given_leg**(*side_length*)

Module contents

mathmaker.lib.core package

Submodules

mathmaker.lib.core.base module

class mathmaker.lib.core.base.**Clonable**
Bases: object

clone()

class mathmaker.lib.core.base.**Drawable**
Bases: *mathmaker.lib.core.base.NamedObject*

eps_filename

euk_filename

into_euk(***options*)

into_pic(***options*)

name

class mathmaker.lib.core.base.**NamedObject**
Bases: *mathmaker.lib.core.base.Clonable*

name

class mathmaker.lib.core.base.**Printable**
Bases: *mathmaker.lib.core.base.NamedObject*

into_str(**options*)

printed
Shortcut for self.into_str(force_expression_begins=True)

This returns the string of the Printable object, assuming it starts the expression.

mathmaker.lib.core.base_calculus module

class mathmaker.lib.core.base_calculus.**AngleItem**(*raw_value=None*, *copy_this=None*, *from_this_angle=None*)
Bases: *mathmaker.lib.core.base_calculus.Item*

Represent Angles' names, like widehat{ABC} (handled as Items).

into_str(**options*)
Return the printed version of the AngleItem.

If the AngleItem is literal, it will be displayed like a literal Item yet wrapped in a 'wide hat'. If it is not numeric, the unit will be automatically discarded.

class mathmaker.lib.core.base_calculus.**BinomialIdentity**(*arg*, ***options*)
Bases: *mathmaker.lib.core.base_calculus.Expandable*

a
Gets the 'a' term of the BinomialIdentity

b
Gets the 'b' term of the BinomialIdentity

expand()

expand_and_reduce_next_step(**options*)

get_a()

get_b()

get_kind()

into_str(**options*)

kind
kind of BinomialIdentity it, e.g. 'sum_square'|'difference_square'|'squares_difference'

class mathmaker.lib.core.base_calculus.**CommutativeOperation**
Bases: *mathmaker.lib.core.base_calculus.Operation*

append(*elt*)

compact_display
compact_display field of a CommutativeOperation

contains_a_rounded_number()

contains_exactly(*objct*)

evaluate(**options*)

get_compact_display()

get_first_letter()

get_info()

get_sign()

info
info field of a CommutativeOperation

is_displ_as_a_single_int()

is_displ_as_a_single_neutral(*neutral_elt*)

is_displ_as_a_single_numeric_Item()

remove(*elt*)

set_compact_display(*arg*)

set_info(*arg*)

set_sign(*arg*)

throw_away_the_neutrals()

class mathmaker.lib.core.base_calculus.**Expandable**(*arg*, ***options*)
Bases: *mathmaker.lib.core.base_calculus.Product*

expand(***options*)

expand_and_reduce_()

expand_and_reduce_next_step(***options*)

is_expandable()

class mathmaker.lib.core.base_calculus.**Fraction**(*arg*, *ignore_1_denominator=False*, ***options*)
Bases: *mathmaker.lib.core.base_calculus.Quotient*

calculate_next_step(***options*)

completely_reduced()

evaluate(***options*)

expand_and_reduce_next_step(***options*)

get_same_deno_reduction_in_progress()

get_simplification_in_progress()

get_status()

is_a_decimal_number()

is_reducible()

replace_striked_out()

same_deno_reduction_in_progress
Fraction's same_deno_reduction_in_progress field

set_down_numerator_s_minus_sign()

set_same_deno_reduction_in_progress(*arg*)

set_status(*arg*)

simplification_in_progress
Fraction's simplification_in_progress status

simplification_line()

simplified()

status
Fraction's status

class mathmaker.lib.core.base_calculus.**Function**(*copy_this=None*, *name='f'*, *var= {+x (unit='') ^.1.}*, *fct=<function Function.<lambda>>*, *num_val=.1.*, *display_mode='literal'*, *inv_fct=None*, *unlocked=False*)
Bases: *mathmaker.lib.core.base_calculus.Item*

Represent the image of a number under a function like f(x) or cos(ABC).

argument

calculate_next_step(***options*)
Will only swap to numeric argument, no automatic evaluation.

contains_a_rounded_number()
f(...) neither its argument never have any reason to be rounded.

contains_exactly(*objct*)
True if the looked for object is self.

evaluate(***options*)
Return the value of f(number).

expand_and_reduce_next_step(***options*)
Same as calculate_ntext_step(), for Functions.

fct
The lambda function to use for evaluation.

get_first_letter()
Return the first letter of Function's name.

get_iteration_list()
Return [variable, exponent].

get_minus_signs_nb()
1 if Function object has a negative sign and no even exponent, else 0.

image_notation

into_str(***options*)
Creates the str version of the Function.

inv_fct
The lambda function to use for evaluation.

is_displ_as_a_single_0()
f(x) is never a single 0.

is_displ_as_a_single_1()
f(x) is never a single 1.

is_displ_as_a_single_int()
f(x) is never a single int.

is_displ_as_a_single_minus_1()
f(x) is never a single -1.

is_displ_as_a_single_neutral(*elt*)
f(x) is never a single neutral element.

is_displ_as_a_single_numeric_Item()
f(x) is never a single numeric Item (like any single number).

is_expandable()
f(x) is not expandable.

is_literal(*displ_as=False*) → bool
Return True if Function is to be considered literal.

Parameters **displ_as** – if displ_as is True, it's about knowing whether the object should be considered literal for display, otherwise, it's about knowing wether it can be numerically evaluated (directly, without replacing its variable by a Value).

is_null()
True if self evaluates to 0.

is_numeric(*displ_as=False*)
Return True if current display mode is numeric.

Parameters **displ_as** – if displ_as is True, it's about knowing whether the object should be considered numeric for display, otherwise, it's about knowing wether it can be numerically evaluated (directly, without replacing its variable by a Value).

multiply_symbol_is_required(*objct*, *position*)
True if × is required between self and next objct in a Product.

num_val

requires_brackets(*position*)
True if Function requires brackets inside a Product.

requires_inner_brackets()
Return True for cases like (-f(x))^2

set_literal_mode()
Set the display mode to 'literal'.

set_numeric_mode()
Set the display mode to 'numeric'.

substitute(*subst_dict*)
Substitute the argument by its value, if available in subst_dict.

unlocked

var

class mathmaker.lib.core.base_calculus.**Item**(*arg*, ***options*)
Bases: *mathmaker.lib.core.root_calculus.Exponented*

calculate_next_step(***options*)

contains_a_rounded_number()

contains_exactly(*objct*)

digits_number()

evaluate(***options*)

expand_and_reduce_next_step(***options*)

force_display_sign_once
Item's force_display_sign_once field

get_first_letter()

get_force_display_sign_once()

get_is_out_striked()

get_iteration_list()

get_minus_signs_nb()

get_raw_value()

get_unit()

get_value_inside()

into_str(***options*)

is_displ_as_a_single_0()

is_displ_as_a_single_1()

is_displ_as_a_single_int()

is_displ_as_a_single_minus_1()

is_displ_as_a_single_neutral(*elt*)

is_displ_as_a_single_numeric_Item()

is_expandable()

is_literal(*displ_as=False*) → bool
Return True if Item is to be considered literal.

Parameters **displ_as** – not applicable to Items

is_null()

is_numeric(*displ_as=False*)

is_out_striked
Item's is_out_striked field

multiply_symbol_is_required(*objct*, *position*)

needs_to_get_rounded(*precision*)

raw_value
Item's raw value

requires_brackets(*position*)

requires_inner_brackets()

round(*precision*)

set_force_display_sign_once(*arg*)

set_is_out_striked(*arg*)

set_unit(*arg*)

set_value_inside(*arg*)

turn_into_fraction()

unit
Unit of the Item

value_inside
Item's Value

class mathmaker.lib.core.base_calculus.**Monomial**(*arg*, ***options*)
Bases: *mathmaker.lib.core.base_calculus.Product*

coeff
Monomial's coefficient

degree
Monomial's degree

get_coeff()

get_degree()

get_first_letter()

get_raw_value()

get_sign()

get_value_inside()

is_negative()

is_null()

is_numeric(*displ_as=False*)

is_positive()

letter
Monomial's letter

raw_value
0-degree-Monomial's value

set_coeff(*arg*)

set_degree(*arg*)

set_letter(*letter*)

sign
Monomial's sign

value_inside
0-degree Monomial's Value inside

class mathmaker.lib.core.base_calculus.**Operation**
Bases: *mathmaker.lib.core.root_calculus.Exponented*

element
element field of Operation

get_element()

get_iteration_list()

get_neutral()

get_symbol()

is_expandable()

is_literal(*displ_as=False*) → bool
Return True if Operation is to be considered literal.

Parameters **displ_as** – not applicable to Operations

is_numeric(*displ_as=False*)

neutral
neutral field of Operation

operator(*arg1*, *arg2*)

reset_element()

set_element(*n*, *arg*)

set_symbol(*arg*)

symbol
symbol field of Operation

class mathmaker.lib.core.base_calculus.**Polynomial**(*arg*)
Bases: *mathmaker.lib.core.base_calculus.Sum*

degree
Real degree of the Polynomial

get_degree()

get_max_degree()

class mathmaker.lib.core.base_calculus.**Product**(*arg*)
Bases: *mathmaker.lib.core.base_calculus.CommutativeOperation*

calculate_next_step(***options*)

expand_and_reduce_next_step(***options*)

factor
To access the factors of the Product.

get_factors_list(*given_kind*)

get_factors_list_except(*objct*)

get_first_factor()

get_minus_signs_nb()

into_str(***options*)

is_displ_as_a_single_0()

is_displ_as_a_single_1()

is_displ_as_a_single_minus_1()

is_null()

is_reducible()

multiply_symbol_is_required(*objct*, *position*)

operator(*arg1*, *arg2*)

order()

reduce_()

requires_brackets(*position*)

requires_inner_brackets()

set_factor(*n*, *arg*)

class mathmaker.lib.core.base_calculus.**Quotient**(*arg*, *ignore_1_denominator=False*, ***options*)

Bases: *mathmaker.lib.core.base_calculus.Operation*

calculate_next_step(***options*)

contains_a_rounded_number()

contains_exactly(*objct*)

denominator
denominator field of Quotient

evaluate(***options*)

expand_and_reduce_next_step(***options*)
If possible, expands Quotient's numerator and/or denominator.

get_denominator()

get_iteration_list()

get_minus_signs_nb()

get_numerator()

get_sign()

into_str(***options*)

invert()

is_displ_as_a_single_0()

is_displ_as_a_single_1()

is_displ_as_a_single_int()

is_displ_as_a_single_minus_1()

is_displ_as_a_single_neutral(*elt*)

is_displ_as_a_single_numeric_Item()

is_null()

multiply_symbol_is_required(*objct*, *position*)

numerator
numerator field of Quotient

operator(*arg1*, *arg2*)

requires_brackets(*position*)

requires_inner_brackets()

set_denominator(*arg*)

set_numerator(*arg*)

class mathmaker.lib.core.base_calculus.**SquareRoot**(*arg*, ***options*)
Bases: *mathmaker.lib.core.base_calculus.Function*

calculate_next_step(***options*)

contains_a_rounded_number()

contains_exactly(*objct*)

expand_and_reduce_next_step(***options*)

force_display_sign_once
Item's force_display_sign_once field

get_force_display_sign_once()

get_iteration_list()

get_minus_signs_nb()

into_str(***options*)

is_displ_as_a_single_0()

is_displ_as_a_single_1()

is_displ_as_a_single_int()

is_displ_as_a_single_minus_1()

is_displ_as_a_single_neutral(*elt*)

is_displ_as_a_single_numeric_Item()

is_expandable()

is_literal(*displ_as=False*)
Return True if SquareRoot is to be considered literal.

Parameters **displ_as** – not applicable to SquareRoots

is_null()

is_numeric(*displ_as=False*)

multiply_symbol_is_required(*objct*, *position*)

requires_brackets(*position*)

requires_inner_brackets()

set_force_display_sign_once(*arg*)

turn_into_fraction()

class mathmaker.lib.core.base_calculus.**Sum**(*arg*)
Bases: *mathmaker.lib.core.base_calculus.CommutativeOperation*

calculate_next_step(***options*)

expand_and_reduce_next_step(***options*)

force_inner_brackets_display
force_inner_brackets_display field of a Sum

get_force_inner_brackets_display()

get_literal_terms()

get_minus_signs_nb()

get_numeric_terms()

get_terms_lexicon()

intermediate_reduction_line()

into_str(***options*)

is_displ_as_a_single_0()

is_displ_as_a_single_1()

is_displ_as_a_single_minus_1()

is_null()

is_reducible()

multiply_symbol_is_required(*objct*, *position*)

next_displayable_term_nb(*position*)

numeric_terms_require_to_be_reduced()

operator(*arg1*, *arg2*)

reduce_()

requires_brackets(*position*)

requires_inner_brackets()

set_force_inner_brackets_display(*arg*)

set_term(*n*, *arg*)

term
To access the terms of the Sum.

mathmaker.lib.core.base_geometry module

class mathmaker.lib.core.base_geometry.**Angle**(*arg*, ***options*)
Bases: *mathmaker.lib.core.base.Drawable*, *mathmaker.lib.core.base.Printable*

into_str(***options*)

label

label_display_angle

mark

measure

points

vertex

class mathmaker.lib.core.base_geometry.**Point**(*arg*, ***options*)
Bases: *mathmaker.lib.core.base.Drawable*

name

rotate(*center*, *angle*, ***options*)

x

x_exact

y

y_exact

class mathmaker.lib.core.base_geometry.**Ray**(*arg*, ***options*)
Bases: *mathmaker.lib.core.base.Drawable*

class mathmaker.lib.core.base_geometry.**Segment**(*arg*, ***options*)
Bases: *mathmaker.lib.core.base.Drawable*

invert_length_name()
Swap points' names in the length name. E.g. AB becomes BA.

label
Label of the Segment (the displayed information).

label_into_euk()
Return the label correctly positionned along the Segment.

length
Fake length of the Segment (the one used in a problem).

length_has_been_set
Whether the (fake) length has been set or not.

length_name
Length's name of the Segment, like AB.

mark

points

real_length
Real length (build length) of the Segment.

setup_label(*flag*)

class mathmaker.lib.core.base_geometry.**Vector**(*arg*, ***options*)
Bases: *mathmaker.lib.core.base_geometry.Point*

bisector_vector(*arg*)
Return a vector colinear to the bisector of self and another vector.

Parameters **arg** (Vector) – the other vector

norm
Return the norm of self.

orthogonal_unit_vector(*clockwise=True*)
Return a unit vector that's (default clockwise) orthogonal to self.

If clockwise is set to False, then the anti-clockwise orthogonal vector is returned.

slope
Return the slope of self.

unit_vector()
Return the unit vector built from self

mathmaker.lib.core.calculus module

class mathmaker.lib.core.calculus.**ComposedCalculable**
Bases: *mathmaker.lib.core.base.Printable*

class mathmaker.lib.core.calculus.**CrossProductEquation**(*arg*)
Bases: *mathmaker.lib.core.calculus.Equation*

get_variable_obj()

get_variable_position()

solve_next_step(***options*)

variable_obj
Variable object of the Equation

variable_position
Variable position in the Equation

class mathmaker.lib.core.calculus.**Equality**(*objcts*, *subst_dict=None*, ***options*)
Bases: *mathmaker.lib.core.calculus.ComposedCalculable*, *mathmaker.lib.core.root_calculus.Substitutable*

content
The content to be substituted (list containing literal objects).

elements
Elements of the object

equal_signs
Equal signs of the object

get_elements()

get_equal_signs()

into_str(***options*)

class mathmaker.lib.core.calculus.**Equation**(*arg*, ***options*)
Bases: *mathmaker.lib.core.calculus.ComposedCalculable*

auto_resolution(***options*)

get_left_hand_side()

get_number()

get_right_hand_side()

get_variable_letter()

into_str(***options*)

left_hand_side
Left hand side of the Equation

name

number
Number of the Equation

right_hand_side
Right hand side of the Equation

set_hand_side(*left_or_right*, *arg*)

set_number(*arg*)

solve_next_step(***options*)

variable_letter
Variable letter of the Equation

class mathmaker.lib.core.calculus.**Expression**(*integer_or_letter*, *objct*)
Bases: *mathmaker.lib.core.calculus.ComposedCalculable*

auto_expansion_and_reduction(***options*)

get_right_hand_side()

into_str(***options*)

right_hand_side
Right hand side of the object

set_right_hand_side(*arg*)

class mathmaker.lib.core.calculus.**QuotientsEquality**(*arg*, *displ_as_qe=True*, *ignore_1_denos=True*, *subst_dict=None*)
Bases: *mathmaker.lib.core.calculus.Table*

A shortcut to create Tables as quotients equalities.

class mathmaker.lib.core.calculus.**Table**(*arg*, *displ_as_qe=False*, *ignore_1_denos=None*, *subst_dict=None*)
Bases: *mathmaker.lib.core.base.Printable*, *mathmaker.lib.core.root_calculus.Substitutable*

class SubstitutableList(**args*, *subst_dict=None*)
Bases: list, *mathmaker.lib.core.root_calculus.Substitutable*

A list that can call substitute() on its elements.

content
The content to be substituted (list containing literal objects).

Table.**auto_resolution**(*col0=0*, *col1=1*, *subst_dict=None*, ***options*)

Table.**cell**
t.cell is the complete Table t.cell[i][j] is a cell

Table.**content**

Table.**cross_product**(*col*, *x_position*, *remove_1_deno=True*, ***options*)

Table.**displ_as_qe**

Table.**get_cell**()

Table.**ignore_1_denos**

Table.**into_crossproduct_equation**(*col0=0*, *col1=1*) → mathmaker.lib.core.calculus.CrossProductEquation
Create a CrossProductEquation from two columns.

Ensure there is only one literal among the four cells before using it.

Parameters

- **col0** – the number of the first column to use
- **col1** – the number of the second column to use

Table.**into_str**(*as_a_quotients_equality=None*, *ignore_1_denos=None*, ***options*)

Table.**is_numeric**(*displ_as=False*)

class mathmaker.lib.core.calculus.**Table_UP**(*coeff*, *first_line*, *info*, *displ_as_qe=False*)
Bases: *mathmaker.lib.core.calculus.Table*

coeff
the coefficient of the Table_UP

crossproducts_info
infos about the cross products

get_coeff()

get_crossproducts_info()

into_crossproduct_equation(*arg*)

mathmaker.lib.core.geometry module

class mathmaker.lib.core.geometry.**InterceptTheoremConfiguration**(*points_names=None*, *butterfly=False*, *sketch=True*, *build_ratio=None*, *build_dimensions=None*, *rotate_around_isobarycenter='no'*)
Bases: *mathmaker.lib.core.geometry.Triangle*

butterfly

chunk

enlargement_ratio

into_euk(***options*)
Create the euk file content, as a str

point

ratios_equalities() → mathmaker.lib.core.calculus.Table
Return a Table matching the ratios equalities.

ratios_equalities_substituted() → mathmaker.lib.core.calculus.Table_UP
Return the ratios equalities containing known numbers.

It is returned as a Table_UP object.

ratios_for_converse() → mathmaker.lib.core.calculus.Table
Return a Table matching the ratios equality for converse.

set_lengths(*lengths_list*, *enlargement_ratio*)
Set all ("fake") lengths of the figure.

The given lengths' list matches the three small sides. The ratio will be used to compute all other segments' sides. As these lengths are the "fake" ones (not the ones used to draw the figure, but the ones that will show up on the figure), this ratio is the "fake" one (not the same as self.ratio).

Parameters

- **lengths_list** (*a list (of Values)*) – the list of the lengths for small0, small1, small2
- **enlargement_ratio** (*any Evaluable*) – the enlargement ratio of the exercise.

small

u

v

class mathmaker.lib.core.geometry.**Polygon**(*arg*, ***options*)
Bases: *mathmaker.lib.core.base.Drawable*

angle

filename

into_euk(***options*)

lengths_have_been_set

name

nature

perimeter

rename(*n*)

rotation_angle

set_lengths(*lengths_list*)

setup_labels(*flags_list*, *segments_list=None*)
Tells what to display along each segment of the list.

If no segments' list is provided, it defaults to the Polygon's sides' list. It is expected that both the flags' and segments' lists have the same length. Meaning of the flags' list: - a '?' will be displayed for each Segment flagged as None or '?' - its length will be displayed if it's flagged as anything else

evaluating to True

•nothing will be displayed it it's flagged as anything else evaluating to False

Parameters

- **flags_list** (*list*) – the list of the flags
- **segments_list** (*list (of Segments)*) – the list of the Segments to flag

side

vertex

work_out_euk_box(*vertices=None*)

class mathmaker.lib.core.geometry.**Rectangle**(*arg*, ***options*)
Bases: *mathmaker.lib.core.geometry.Polygon*

area

length

set_lengths(*lengths_list*)

width

class mathmaker.lib.core.geometry.**RightTriangle**(*arg*, ***options*)
Bases: *mathmaker.lib.core.geometry.Triangle*

hypotenuse

leg

pythagorean_equality(***options*)

pythagorean_substequality(***options*)

right_angle

setup_for_trigonometry(*angle_nb=None*, *trigo_fct=None*, *angle_val=None*, *up_length_val=None*, *down_length_val=None*, *length_unit=None*, *only_mark_unknown_angle=False*, *mark_angle='simple'*)

Setup labels, determine subst_dict and stores configuration details.

Exactly one parameter among the three *_val ones must be left to None. According to the chosen trigo_fct and this parameter, this method will create the correct subst_dict.

Parameters

- **angle_nb** (*int*) – must be either 0 or 2 (index of an acute angle)
- **trigo_fct** (*str*) – must belong to ['cos', 'sin', 'tan']
- **angle_val** (Value *(or leave it to None to use it as the unknown value to calculate)*) – the angle's Value
- **up_length_val** (Value *(or leave it to None to use it as the unknown value to calculate)*) – the length's Value of the side that's at the numerator of the trigonometric formula
- **down_length_val** (Value *(or leave it to None to use it as the unknown value to calculate)*) – the length's Value of the side that's at the denominator of the trigonometric formula
- **length_unit** (*anything that can be used as argument for Units*) – the length's unit to use for lengths

side_adjacent_to(*angle=None*)

Return the side adjacent to given angle.

Parameters **angle** (*must be self.angle[0] or self.angle[2]*) – one of the acute angles

side_opposite_to(*angle=None*)

Return the side opposite to given angle.

Parameters **angle** (*must be self.angle[0] or self.angle[2]*) – one of the acute angles

trigonometric_equality(*angle=None*, *trigo_fct=None*, *subst_dict=None*, *autosetup=False*)

Return the required trigonometric equality.

Parameters

- **angle** (Angle) – the acute Angle to use
- **trigo_fct** (*str*) – either 'cos', 'sin' or 'tan'
- **subst_dict** (*dict*) – a correct substitution dictionary
- **autosetup** (*bool*) – if enabled, will take the angle, trigo_fct and subst_dict from pre-configured values (requires to have called setup_for_trigonometry() previously).

trigonometric_ratios()

Definitions of the three standard trigonometric ratios.

class mathmaker.lib.core.geometry.**Square**(*arg*, ***options*)

Bases: *mathmaker.lib.core.geometry.Polygon*

area

set_lengths(*lengths_list*)

set_marks(*arg*)

set_side_length(*side_length*)

side_length

class mathmaker.lib.core.geometry.**Triangle**(*arg*, ***options*)
Bases: *mathmaker.lib.core.geometry.Polygon*

mathmaker.lib.core.root_calculus module

class mathmaker.lib.core.root_calculus.**Calculable**
Bases: *mathmaker.lib.core.root_calculus.Evaluable*

calculate_next_step(***options*)

expand_and_reduce_next_step(***options*)

get_iteration_list()

is_displ_as_a_single_0()

is_displ_as_a_single_1()

is_displ_as_a_single_int()

is_displ_as_a_single_minus_1()

is_displ_as_a_single_neutral(*elt*)

is_displ_as_a_single_numeric_Item()

multiply_symbol_is_required(*objct*, *position*)

requires_brackets(*position*)

requires_inner_brackets()

substitute(*subst_dict*)

class mathmaker.lib.core.root_calculus.**Evaluable**
Bases: *mathmaker.lib.core.base.Printable*

alphabetical_order_cmp(*other_objct*)

contains_a_rounded_number()

contains_exactly(*objct*)

evaluate(***options*)

get_first_letter()

is_literal()

is_null()

is_numeric(*displ_as=False*)

class mathmaker.lib.core.root_calculus.**Exponented**
Bases: *mathmaker.lib.core.root_calculus.Signed*

exponent
Exponent of the Function

exponent_must_be_displayed()

get_exponent()

set_exponent(*arg*)

class mathmaker.lib.core.root_calculus.**Signed**
Bases: *mathmaker.lib.core.root_calculus.Calculable*

get_minus_signs_nb()

get_sign()

is_negative()

is_positive()

set_opposite_sign()

set_sign(*arg*)

sign
Sign of the object

class mathmaker.lib.core.root_calculus.**Substitutable**(*subst_dict=None*)
Bases: object

Any object whose (literal) value(s) can be substituted by numeric ones.

Any Substitutable must define a content property, should include an optional subst_dict argument in its __init__() method and must ensure that a _subst_dict is defined (an easy way to do this is calling Substitutable.__init__(self, subst_dict=subst_dict). The substitute() method is redefined by some Substitutable objects.

content
The content to be substituted (list containing literal objects).

subst_dict
Get the default dictionary to use for substitution.

substitute(*subst_dict=None*)
If a subst_dict has been defined, it is used for literals substitution.

class mathmaker.lib.core.root_calculus.**Unit**(*arg*, ***options*)
Bases: *mathmaker.lib.core.root_calculus.Exponented*

exponent

into_str(***options*)

name

class mathmaker.lib.core.root_calculus.**Value**(*arg*, *text_in_maths=True*, ***options*)
Bases: *mathmaker.lib.core.root_calculus.Signed*

abs_value

calculate_next_step(***options*)

contains_a_rounded_number()

contains_exactly(*objct*)

digits_number()

evaluate(***options*)

get_first_letter()

get_has_been_rounded()

get_iteration_list()

get_minus_signs_nb()

get_sign()

get_unit()

has_been_rounded
'has been rounded' state of the Value

into_str(***options*)

is_a_perfect_square()

is_an_integer()

is_displ_as_a_single_0()

is_displ_as_a_single_1()

is_displ_as_a_single_int()

is_displ_as_a_single_minus_1()

is_displ_as_a_single_numeric_Item()

is_literal(*displ_as=False*) → bool
Return True if Value is to be considered literal.

Parameters **displ_as** – not applicable to Values

is_null()

is_numeric(*displ_as=False*)

needs_to_get_rounded(*precision*)

raw_value

round(*precision*)

set_has_been_rounded(*arg*)

set_opposite_sign()

set_sign(*arg*)

set_unit(*arg*)

sign
Sign of the Value

sqrt()

substitute(*subst_dict*)

unit
Unit of the Value

mathmaker.lib.core.utils module

mathmaker.lib.core.utils.**check_lexicon_for_substitution**(*objcts*, *subst_dict*, *how_many*)

mathmaker.lib.core.utils.**gather_literals**(*xpr*)
Return all literal Values|AngleItems of an expression.

Parameters **xpr** (Calculable) – the expression to iter over

mathmaker.lib.core.utils.**put_term_in_lexicon**(*provided_key*, *associated_coeff*, *lexi*)

mathmaker.lib.core.utils.**reduce_literal_items_product**(*provided_list*)

Module contents

mathmaker.lib.machine package

Submodules

mathmaker.lib.machine.LaTeX module

class mathmaker.lib.machine.LaTeX.**LaTeX**(*language*, *create_pic_files=True*, ***options*)
Bases: *mathmaker.lib.machine.Structure.Structure*

addvspace(*height='30.0pt'*, ***options*)
Add a vertical space.

create_table(*size*, *content*, ***options*)

insert_dashed_hline(***options*)

insert_nonbreaking_space(***options*)

insert_picture(*drawable_arg*, ***options*)

insert_vspace(***options*)

reset_exercises_counter()

set_font_size_offset(*arg*)

set_redirect_output_to_str(*arg*)

translate_font_size(*arg*)

type_string(*objct*, ***options*)

write(*given_string*, ***options*)

write_document_begins()

write_document_ends()

write_document_header()

write_exercise_number()

write_jump_to_next_page()

write_layout(*size*, *col_widths*, *content*, ***options*)

write_math_style1(*given_string*)

write_math_style2(*given_string*, ***kwargs*)

write_new_line(*check=''*, *check2=''*)

write_new_line_twice(***options*)

write_out(*latex_document: str*, *pdf_output=False*)
Writes the given document to the output.

If pdf_output is set to True then the document will be compiled into a pdf and the pdf content will be written to output.

Parameters

- **latex_document** – contains the entire LaTeX document
- **pdf_output** – if True, output will be written in pdf format

write_set_font_size_to(*arg*)

mathmaker.lib.machine.Structure module

class mathmaker.lib.machine.Structure.**Structure**(*language*)
Bases: object

clone(*language*)

insert_dashed_hline(***options*)

insert_nonbreaking_space(***options*)

insert_picture(*drawable_arg*, ***options*)

insert_vspace(***options*)

redirect_output_to_str()

reset_exercises_counter()

set_font_size_offset(*arg*)

set_redirect_output_to_str(*arg*)

translate_font_size(*arg*)

type_string(*objct*, ***options*)

write(*given_string*, ***options*)

write_document_begins()

write_document_ends()

write_document_header()

write_exercise_number()

write_jump_to_next_page()

write_layout(*size*, *col_widths*, *content*, ***options*)

write_math_style1(*given_string*)

write_math_style2(*given_string*)

write_new_line(***options*)

write_new_line_twice(***options*)

write_out(*given_string*, ***options*)

write_set_font_size_to(*arg*)

write_table(*size*, *col_widths*, *content*, ***options*)

Module contents

mathmaker.lib.sheet package

Subpackages

mathmaker.lib.sheet.exercise package

Subpackages

mathmaker.lib.sheet.exercise.question package

Subpackages

mathmaker.lib.sheet.exercise.question.mc_modules package

Submodules

mathmaker.lib.sheet.exercise.question.mc_modules.addi_direct module

class mathmaker.lib.sheet.exercise.question.mc_modules.addi_direct.**sub_object**(*numbers_to_use*, ***options*)

Bases: mathmaker.lib.sheet.exercise.question.submodule.structure

a(***options*)

q(***options*)

mathmaker.lib.sheet.exercise.question.mc_modules.area_rectangle module

class mathmaker.lib.sheet.exercise.question.mc_modules.area_rectangle.**sub_object**(*numbers_to_use*, ***options*)

Bases: mathmaker.lib.sheet.exercise.question.submodule.structure

a(***options*)

q(***options*)

mathmaker.lib.sheet.exercise.question.mc_modules.area_square module

class mathmaker.lib.sheet.exercise.question.mc_modules.area_square.**sub_object**(*numbers_to_use*, ***options*)

Bases: mathmaker.lib.sheet.exercise.question.submodule.structure

a(***options*)

q(***options*)

mathmaker.lib.sheet.exercise.question.mc_modules.divi_direct module

class mathmaker.lib.sheet.exercise.question.mc_modules.divi_direct.**sub_object**(*numbers_to_use*, ***options*)

Bases: mathmaker.lib.sheet.exercise.question.submodule.structure

a(***options*)

q(***options*)

mathmaker.lib.sheet.exercise.question.mc_modules.mc_module module

mathmaker.lib.sheet.exercise.question.mc_modules.multi_direct module

class mathmaker.lib.sheet.exercise.question.mc_modules.multi_direct.**sub_object**(*numbers_to_use*, ***options*)

Bases: mathmaker.lib.sheet.exercise.question.submodule.structure

a(***options*)

q(***options*)

mathmaker.lib.sheet.exercise.question.mc_modules.multi_hole module

class mathmaker.lib.sheet.exercise.question.mc_modules.multi_hole.**sub_object**(*numbers_to_use*, ***options*)

Bases: object

a(***options*)

q(***options*)

mathmaker.lib.sheet.exercise.question.mc_modules.multi_reversed module

class mathmaker.lib.sheet.exercise.question.mc_modules.multi_reversed.**sub_object**(*numbers_to_use*, ***options*)

Bases: object

a(***options*)

q(***options*)

mathmaker.lib.sheet.exercise.question.mc_modules.perimeter_irregular_quadrilateral module

class mathmaker.lib.sheet.exercise.question.mc_modules.perimeter_irregular_quadrilateral.**sub_**

Bases: mathmaker.lib.sheet.exercise.question.submodule.structure

a(***options*)

q(***options*)

mathmaker.lib.sheet.exercise.question.mc_modules.perimeter_rectangle module

class mathmaker.lib.sheet.exercise.question.mc_modules.perimeter_rectangle.**sub_object**(*numbers_*, ***options*)

Bases: mathmaker.lib.sheet.exercise.question.submodule.structure

a(***options*)

q(***options*)

mathmaker.lib.sheet.exercise.question.mc_modules.perimeter_square module

class mathmaker.lib.sheet.exercise.question.mc_modules.perimeter_square.**sub_object**(*numbers_to_u*, ***options*)

Bases: mathmaker.lib.sheet.exercise.question.submodule.structure

a(***options*)

q(***options*)

mathmaker.lib.sheet.exercise.question.mc_modules.rank_direct module

class mathmaker.lib.sheet.exercise.question.mc_modules.rank_direct.**sub_object**(*rank_to_use*, ***options*)

Bases: object

a(***options*)

q(***options*)

mathmaker.lib.sheet.exercise.question.mc_modules.rank_numberof module

class mathmaker.lib.sheet.exercise.question.mc_modules.rank_numberof.**sub_object**(*rank_to_use*, ***options*)

Bases: object

a(***options*)

q(***options*)

mathmaker.lib.sheet.exercise.question.mc_modules.rank_reversed module

class mathmaker.lib.sheet.exercise.question.mc_modules.rank_reversed.**sub_object**(*rank_to_use*, ***options*)

Bases: object

a(***options*)

q(***options*)

mathmaker.lib.sheet.exercise.question.mc_modules.rectangle_length_or_width module

class mathmaker.lib.sheet.exercise.question.mc_modules.rectangle_length_or_width.**sub_object**(*n*
***
t

Bases: mathmaker.lib.sheet.exercise.question.submodule.structure

a(***options*)

q(***options*)

mathmaker.lib.sheet.exercise.question.mc_modules.subtr_direct module

class mathmaker.lib.sheet.exercise.question.mc_modules.subtr_direct.**sub_object**(*numbers_to_use*, ***options*)

Bases: mathmaker.lib.sheet.exercise.question.submodule.structure

a(***options*)

q(***options*)

mathmaker.lib.sheet.exercise.question.mc_modules.vocabulary_addi module

class mathmaker.lib.sheet.exercise.question.mc_modules.vocabulary_addi.**sub_object**(*numbers_to_use*
***options*)

Bases: *mathmaker.lib.sheet.exercise.question.mc_modules.vocabulary_questions.structure*

mathmaker.lib.sheet.exercise.question.mc_modules.vocabulary_divi module

class mathmaker.lib.sheet.exercise.question.mc_modules.vocabulary_divi.**sub_object**(*numbers_to_use*
***options*)

Bases: *mathmaker.lib.sheet.exercise.question.mc_modules.vocabulary_questions.structure*

mathmaker.lib.sheet.exercise.question.mc_modules.vocabulary_multi module

class mathmaker.lib.sheet.exercise.question.mc_modules.vocabulary_multi.**sub_object**(*numbers_to_u*
***options*)

Bases: *mathmaker.lib.sheet.exercise.question.mc_modules.vocabulary_questions.structure*

mathmaker.lib.sheet.exercise.question.mc_modules.vocabulary_questions module

class mathmaker.lib.sheet.exercise.question.mc_modules.vocabulary_questions.**structure**(*nbs_to_us*
***kwargs*

Bases: mathmaker.lib.sheet.exercise.question.submodule.structure

a(***kwargs*)

q(***kwargs*)

mathmaker.lib.sheet.exercise.question.mc_modules.vocabulary_simple_multiple_of_a_number module

class mathmaker.lib.sheet.exercise.question.mc_modules.vocabulary_simple_multiple_of_a_number

Bases: *mathmaker.lib.sheet.exercise.question.mc_modules.vocabulary_questions.structure*

mathmaker.lib.sheet.exercise.question.mc_modules.vocabulary_simple_part_of_a_number module

class mathmaker.lib.sheet.exercise.question.mc_modules.vocabulary_simple_part_of_a_number.**sub**

Bases: *mathmaker.lib.sheet.exercise.question.mc_modules.vocabulary_questions.structure*

mathmaker.lib.sheet.exercise.question.mc_modules.vocabulary_subtr module

class mathmaker.lib.sheet.exercise.question.mc_modules.vocabulary_subtr.**sub_object**(*numbers_to_u*
***options*)

Bases: *mathmaker.lib.sheet.exercise.question.mc_modules.vocabulary_questions.structure*

Module contents

Submodules

mathmaker.lib.sheet.exercise.question.Q_AlgebraExpressionExpansion module

class mathmaker.lib.sheet.exercise.question.Q_AlgebraExpressionExpansion.**Q_AlgebraExpressionE**

Bases: *mathmaker.lib.sheet.exercise.question.Q_Structure.Q_Structure*

answer_to_str()

text_to_str()

mathmaker.lib.sheet.exercise.question.Q_AlgebraExpressionReduction module

class mathmaker.lib.sheet.exercise.question.Q_AlgebraExpressionReduction.**Q_AlgebraExpressionR**

Bases: *mathmaker.lib.sheet.exercise.question.Q_Structure.Q_Structure*

answer_to_str()

text_to_str()

mathmaker.lib.sheet.exercise.question.Q_Calculation module

class mathmaker.lib.sheet.exercise.question.Q_Calculation.**Q_Calculation**(*q_kind='default_nothing'*, ***options*)

Bases: *mathmaker.lib.sheet.exercise.question.Q_Structure.Q_Structure*

answer_to_str()

text_to_str()

mathmaker.lib.sheet.exercise.question.Q_Equation module

class mathmaker.lib.sheet.exercise.question.Q_Equation.**Q_Equation**(*q_kind='default_nothing'*, ***options*)

Bases: *mathmaker.lib.sheet.exercise.question.Q_Structure.Q_Structure*

answer_to_str()

text_to_str()

mathmaker.lib.sheet.exercise.question.Q_Factorization module

class mathmaker.lib.sheet.exercise.question.Q_Factorization.**Q_Factorization**(*q_kind='default_nothing*
***options*)

Bases: *mathmaker.lib.sheet.exercise.question.Q_Structure.Q_Structure*

answer_to_str()

text_to_str()

mathmaker.lib.sheet.exercise.question.Q_Factorization.**level_01**(*q_subkind*, ***options*)

mathmaker.lib.sheet.exercise.question.Q_Factorization.**level_02**(*q_subkind*, ***options*)

mathmaker.lib.sheet.exercise.question.Q_Factorization.**level_03**(*q_subkind*, ***options*)

mathmaker.lib.sheet.exercise.question.Q_MentalCalculation module

mathmaker.lib.sheet.exercise.question.Q_Model module

class mathmaker.lib.sheet.exercise.question.Q_Model.**Q_Model**(*q_kind='default_nothing'*, ***options*)

Bases: *mathmaker.lib.sheet.exercise.question.Q_Structure.Q_Structure*

answer_to_str()

text_to_str()

mathmaker.lib.sheet.exercise.question.Q_RightTriangle module

class mathmaker.lib.sheet.exercise.question.Q_RightTriangle.**Q_RightTriangle**(*q_kind='default_nothing'*, ***options*)

Bases: *mathmaker.lib.sheet.exercise.question.Q_Structure.Q_Structure*

answer_to_str()

text_to_str()

mathmaker.lib.sheet.exercise.question.Q_Structure module

class mathmaker.lib.sheet.exercise.question.Q_Structure.**Q_Structure**(*q_kind*, *AVAILABLE_Q_KIND_VALUES*, ***options*)

Bases: object

answer_to_str(***options*)

hint_to_str(***options*)

text_to_str(***options*)

to_str(*ex_or_answers*)

Module contents

Submodules

mathmaker.lib.sheet.exercise.X_AlgebraExpressionExpansion module

class mathmaker.lib.sheet.exercise.X_AlgebraExpressionExpansion.**X_AlgebraExpressionExpansion**(

Bases: *mathmaker.lib.sheet.exercise.X_Structure.X_Structure*

mathmaker.lib.sheet.exercise.X_AlgebraExpressionReduction module

class mathmaker.lib.sheet.exercise.X_AlgebraExpressionReduction.**X_AlgebraExpressionReduction**(

Bases: *mathmaker.lib.sheet.exercise.X_Structure.X_Structure*

mathmaker.lib.sheet.exercise.X_Calculation module

class mathmaker.lib.sheet.exercise.X_Calculation.**X_Calculation**(*x_kind='default_nothing'*, ***options*)

Bases: *mathmaker.lib.sheet.exercise.X_Structure.X_Structure*

mathmaker.lib.sheet.exercise.X_Equation module

class mathmaker.lib.sheet.exercise.X_Equation.**X_Equation**(*x_kind='default_nothing'*, ***options*)

Bases: *mathmaker.lib.sheet.exercise.X_Structure.X_Structure*

mathmaker.lib.sheet.exercise.X_Factorization module

class mathmaker.lib.sheet.exercise.X_Factorization.**X_Factorization**(*x_kind='default_nothing'*, ***options*)

Bases: *mathmaker.lib.sheet.exercise.X_Structure.X_Structure*

mathmaker.lib.sheet.exercise.X_MentalCalculation module

mathmaker.lib.sheet.exercise.X_Model module

class mathmaker.lib.sheet.exercise.X_Model.**X_Model**(*x_kind='default_nothing'*, ***options*)

Bases: *mathmaker.lib.sheet.exercise.X_Structure.X_Structure*

mathmaker.lib.sheet.exercise.X_RightTriangle module

class mathmaker.lib.sheet.exercise.X_RightTriangle.**X_RightTriangle**(*x_kind='default_nothing'*, ***options*)

Bases: *mathmaker.lib.sheet.exercise.X_Structure.X_Structure*

mathmaker.lib.sheet.exercise.X_Structure module

class mathmaker.lib.sheet.exercise.X_Structure.**X_Structure**(*x_kind*, *AVAILABLE_X_KIND_VALUES*, *X_LAYOUTS*, *X_LAYOUT_UNIT*, *number_of_questions=6*, ***options*)

Bases: object

to_str(*ex_or_answers*)

Module contents

Submodules

mathmaker.lib.sheet.AlgebraBalance_01 module

class mathmaker.lib.sheet.AlgebraBalance_01.**AlgebraBalance_01**(***options*)
Bases: *mathmaker.lib.sheet.S_Structure.S_Structure*

mathmaker.lib.sheet.AlgebraBinomialIdentityExpansion module

class mathmaker.lib.sheet.AlgebraBinomialIdentityExpansion.**AlgebraBinomialIdentityExpansion**(*
Bases: *mathmaker.lib.sheet.S_Structure.S_Structure*

write_answers()

mathmaker.lib.sheet.AlgebraExpressionExpansion module

class mathmaker.lib.sheet.AlgebraExpressionExpansion.**AlgebraExpressionExpansion**(***options*)
Bases: *mathmaker.lib.sheet.S_Structure.S_Structure*

mathmaker.lib.sheet.AlgebraExpressionReduction module

class mathmaker.lib.sheet.AlgebraExpressionReduction.**AlgebraExpressionReduction**(***options*)
Bases: *mathmaker.lib.sheet.S_Structure.S_Structure*

mathmaker.lib.sheet.AlgebraFactorization_01 module

class mathmaker.lib.sheet.AlgebraFactorization_01.**AlgebraFactorization_01**(***options*)
Bases: *mathmaker.lib.sheet.S_Structure.S_Structure*

mathmaker.lib.sheet.AlgebraFactorization_02 module

class mathmaker.lib.sheet.AlgebraFactorization_02.**AlgebraFactorization_02**(***options*)
Bases: *mathmaker.lib.sheet.S_Structure.S_Structure*

mathmaker.lib.sheet.AlgebraFactorization_03 module

class mathmaker.lib.sheet.AlgebraFactorization_03.**AlgebraFactorization_03**(***options*)
Bases: *mathmaker.lib.sheet.S_Structure.S_Structure*

mathmaker.lib.sheet.AlgebraMiniTest0 module

class mathmaker.lib.sheet.AlgebraMiniTest0.**AlgebraMiniTest0**(***options*)
Bases: *mathmaker.lib.sheet.S_Structure.S_Structure*

mathmaker.lib.sheet.AlgebraMiniTest1 module

class mathmaker.lib.sheet.AlgebraMiniTest1.**AlgebraMiniTest1**(**options*)
Bases: *mathmaker.lib.sheet.S_Structure.S_Structure*

mathmaker.lib.sheet.AlgebraShortTest module

class mathmaker.lib.sheet.AlgebraShortTest.**AlgebraShortTest**(**options*)
Bases: *mathmaker.lib.sheet.S_Structure.S_Structure*

mathmaker.lib.sheet.AlgebraTest module

class mathmaker.lib.sheet.AlgebraTest.**AlgebraTest**(**options*)
Bases: *mathmaker.lib.sheet.S_Structure.S_Structure*

mathmaker.lib.sheet.AlgebraTest2 module

class mathmaker.lib.sheet.AlgebraTest2.**AlgebraTest2**(**options*)
Bases: *mathmaker.lib.sheet.S_Structure.S_Structure*

mathmaker.lib.sheet.ConverseAndContrapositiveOfPythagoreanTheoremShortTest module

class mathmaker.lib.sheet.ConverseAndContrapositiveOfPythagoreanTheoremShortTest.**ConverseAndC**
Bases: *mathmaker.lib.sheet.S_Structure.S_Structure*

mathmaker.lib.sheet.EquationsBasic module

class mathmaker.lib.sheet.EquationsBasic.**EquationsBasic**(**options*)
Bases: *mathmaker.lib.sheet.S_Structure.S_Structure*

mathmaker.lib.sheet.EquationsClassic module

class mathmaker.lib.sheet.EquationsClassic.**EquationsClassic**(**options*)
Bases: *mathmaker.lib.sheet.S_Structure.S_Structure*

mathmaker.lib.sheet.EquationsHarder module

class mathmaker.lib.sheet.EquationsHarder.**EquationsHarder**(**options*)
Bases: *mathmaker.lib.sheet.S_Structure.S_Structure*

mathmaker.lib.sheet.EquationsShortTest module

class mathmaker.lib.sheet.EquationsShortTest.**EquationsShortTest**(**options*)
Bases: *mathmaker.lib.sheet.S_Structure.S_Structure*

mathmaker.lib.sheet.EquationsTest module

class mathmaker.lib.sheet.EquationsTest.**EquationsTest**(***options*)
Bases: *mathmaker.lib.sheet.S_Structure.S_Structure*

mathmaker.lib.sheet.FractionSimplification module

class mathmaker.lib.sheet.FractionSimplification.**FractionSimplification**(***options*)
Bases: *mathmaker.lib.sheet.S_Structure.S_Structure*

mathmaker.lib.sheet.FractionsProductAndQuotient module

class mathmaker.lib.sheet.FractionsProductAndQuotient.**FractionsProductAndQuotient**(***options*)
Bases: *mathmaker.lib.sheet.S_Structure.S_Structure*

mathmaker.lib.sheet.FractionsSum module

class mathmaker.lib.sheet.FractionsSum.**FractionsSum**(***options*)
Bases: *mathmaker.lib.sheet.S_Structure.S_Structure*

mathmaker.lib.sheet.PythagoreanTheoremShortTest module

class mathmaker.lib.sheet.PythagoreanTheoremShortTest.**PythagoreanTheoremShortTest**(***options*)
Bases: *mathmaker.lib.sheet.S_Structure.S_Structure*

mathmaker.lib.sheet.S_Generic module

class mathmaker.lib.sheet.S_Generic.**S_Generic**(*filename*, ***options*)
Bases: *mathmaker.lib.sheet.S_Structure.S_Structure*

mathmaker.lib.sheet.S_Model module

class mathmaker.lib.sheet.S_Model.**S_Model**(***options*)
Bases: *mathmaker.lib.sheet.S_Structure.S_Structure*

mathmaker.lib.sheet.S_Structure module

class mathmaker.lib.sheet.S_Structure.**S_Structure**(*font_size_offset*, *sheet_layout_unit*, *sheet_layout*, *layout_type*, ***options*)
Bases: object

answers_title_to_str()

sheet_header_to_str()

sheet_text_to_str()

sheet_title_to_str()

texts_to_str(*ex_or_answers*, *n_of_first_ex*)

Module contents

mathmaker.lib.tools package

Submodules

mathmaker.lib.tools.config module

Read configuration files.

mathmaker.lib.tools.config.**load_config**(*file_tag*, *settingsdir*)
Will load the values from the yaml config file, named file_tag.yaml.

The default configuration values are loaded from mathmaker/settings/default/*.yaml, then load_config will update with values found successively in /etc/mathmaker/*.yaml, then in ~/.config/mathmaker/*.yaml, finally in mathmaker/settings/dev/*.yaml.

mathmaker.lib.tools.db module

class mathmaker.lib.tools.db.**source**(*table_name*, *cols*, ***kwargs*)
Bases: object

next(***kwargs*)

mathmaker.lib.tools.ext_dict module

Extend dict.

class mathmaker.lib.tools.ext_dict.**ext_dict**
Bases: dict

A dict with more methods.

flat(*sep='.'*)
Return a recursively flattened dict.

If the dictionary contains nested dictionaries, this function will return a one-level ("flat") dictionary.

Example

```
>>> d = ext_dict({'a': {'a1': 3, 'a2': {'z': 5}}, 'b': 'data'})
>>> d.flat() == {'a.a1': 3, 'a.a2.z': 5, 'b': 'data'}
True
```

recursive_update(*d2*)
Update self with d2 key/values, recursively update nested dicts.

Example

```
>>> d = ext_dict({'a': 1, 'b': {'a': 7, 'c': 10}})
>>> d.recursive_update({'a': 24, 'd': 13, 'b': {'c': 100}})
>>> print(d == {'a': 24, 'd': 13, 'b': {'a': 7, 'c': 100}})
True
```

mathmaker.lib.tools.header_comment module

Provide the function that builds the header comment from software infos.

mathmaker.lib.tools.header_comment.**generate**(*document_format*, *comment_symbol='% '*)
Return the header comment for output text files.

mathmaker.lib.tools.po_file module

mathmaker.lib.tools.po_file.**get_list_of**(*what*, *language*, *arg*)

mathmaker.lib.tools.po_file.**retrieve**(*language*, *po_filename*)

mathmaker.lib.tools.tag module

mathmaker.lib.tools.tag.**classify_tag**(*tag*)

mathmaker.lib.tools.tag.**translate_int_pairs_tag**(*tag*)

mathmaker.lib.tools.tag.**translate_single_nb_tag**(*tag*)
From single..._mintomax, get and return min and max in a dictionary.

mathmaker.lib.tools.wording module

Use these functions to process sentences or objects containing a wording.

mathmaker.lib.tools.wording.**cut_off_hint_from**(*sentence: str*) → tuple
Return the sentence and the possible hint separated.

Only one hint will be taken into account.

Parameters **sentence** (*str*) – the sentence to inspect

Return type tuple

Examples

```
>>> cut_off_hint_from("This sentence has no hint.")
('This sentence has no hint.', '')
>>> cut_off_hint_from("This sentence has a hint: |hint:length_unit|")
('This sentence has a hint:', 'length_unit')
>>> cut_off_hint_from("Malformed hint:|hint:length_unit|")
('Malformed hint:|hint:length_unit|', '')
>>> cut_off_hint_from("Malformed hint: |hint0:length_unit|")
('Malformed hint: |hint0:length_unit|', '')
>>> cut_off_hint_from("Two hints: |hint:unit| |hint:something_else|")
('Two hints: |hint:unit|', 'something_else')
```

mathmaker.lib.tools.wording.**extract_formatting_tags_from**(*s: str*)
Return all tags found wrapped in {}. Punctuation has no effect.

Parameters **s** – the sentence where to look for {tags}.

mathmaker.lib.tools.wording.**handle_valueless_names_tags**(*arg: object*, *sentence: str*)
Each {name} tag found triggers an arg.name attribute to be randomly set.

All concerned tags are: {name}, {nameN}, {masculine_name}, {masculine_nameN}, {feminine_name}, {feminine_nameN}.

If the tag embbeds a value, like in {name=John}, then it's ignored by this function. If arg already has an attribute matching the tag, then it's also ignored by this function.

Now, say arg has no attributes like name, name1, etc. then, if sentence contains:

- "{name}" then arg.name will receive a random name.
- "{name1}", then arg.name1 will receive a random name.
- "{name=Michael}", then this function ignores it.
- "{feminine_name}", then arg.feminine_name will get a random feminine name.

Parameters

- **arg** – the object that attributes must be checked and possibly set
- **sentence** – the sentence where to look for "name" tags.

mathmaker.lib.tools.wording.**handle_valueless_unit_tags**(*arg: object*, *sentence: str*)

Each {*_unit} tag triggers an arg.*_unit attribute to be randomly set.

For instance, if {length_unit} is found, then arg.length_unit will get a random length unit. Moreover, if {area_unit} or {volume_unit} are found, arg.length_unit is set accordingly too. If arg.length_unit does already exist, then arg.area_unit will be set accordingly (and not randomly any more).

{*_unitN}, <*_unit> and <*_unitN> tags will be handled the same way by this function.

If the tag embbeds a value, like in {capacity_unit=dL}, then it's ignored by this function. If arg already has an attribute matching the tag, then it's also ignored by this function.

Parameters

- **arg** (*object*) – the object that attributes must be checked and possibly set
- **sentence** (*str*) – the sentence where to look for "unit" tags.

Return type None

mathmaker.lib.tools.wording.**insert_nonbreaking_spaces**(*sentence: str*)

Replace spaces by nonbreaking ones between a number and the next word.

Parameters **sentence** – the sentence to process

mathmaker.lib.tools.wording.**is_unit**(*word: str*) → bool

Return True if word is a "unit" tag (e.g. ends with _unit}).

Punctuation has no effect.

Parameters **word** – the word to inspect

mathmaker.lib.tools.wording.**is_unitN**(*word*)

Return True if word is a "unitN" tag (e.g. ends with _unitN}).

Punctuation has no effect.

Parameters **word** – the word to inspect

mathmaker.lib.tools.wording.**is_wrapped**(*word: str*, *braces='{}'*, *extra_braces=''*) → bool

Return True if word is wrapped between braces.

Parameters

- **word** – the word to inspect
- **braces** – to change the default {} braces to something else,

like [] or <> :param extra_braces: to add extra braces around the usual ones. Like in ({tag}) or [{tag}] :Examples:

```
>>> is_wrapped('{word}')
True
>>> is_wrapped('{word},')
False
>>> is_wrapped('<word>')
False
>>> is_wrapped('<word>', braces='<>')
True
>>> is_wrapped('({word})')
False
>>> is_wrapped('({word})', extra_braces='()')
True
>>> is_wrapped('[{word}]', extra_braces='()')
False
```

mathmaker.lib.tools.wording.**is_wrapped_P**(*word: str*, *braces='{}'*, *extra_braces=''*) → bool

Return True if word is wrapped between braces & followed by a punctuation.

Parameters

- **word** – the word to inspect
- **braces** – to change the default {} braces to something else,

like [] or <> :param extra_braces: to add extra braces around the usual ones. Like in ({tag}) or [{tag}]

Examples

```
>>> is_wrapped_P('{word}')
False
>>> is_wrapped_P('{word},')
True
>>> is_wrapped_P('<word>')
False
>>> is_wrapped_P('<word>', braces='<>')
False
>>> is_wrapped_P('<word>:', braces='<>')
True
>>> is_wrapped_P('({word})', extra_braces='()')
False
>>> is_wrapped_P('({word}).', extra_braces='()')
True
>>> is_wrapped_P('[{word}]?', extra_braces='[]')
True
```

mathmaker.lib.tools.wording.**is_wrapped_p**(*word: str*, *braces='{}'*, *extra_braces=''*) → bool

Return True if word is wrapped between braces. Punctuation has no effect.

Parameters

- **word** – the word to inspect
- **braces** – to change the default {} braces to something else,

like [] or <> :param extra_braces: to add extra braces around the usual ones. Like in ({tag}) or [{tag}]

Examples

```
>>> is_wrapped_p('{word}')
True
>>> is_wrapped_p('{word},')
True
>>> is_wrapped_p('<word>')
False
>>> is_wrapped_p('<word>', braces='<>')
True
>>> is_wrapped_p('<word>:', braces='<>')
True
>>> is_wrapped_p('({word}).')
False
>>> is_wrapped_p('({word}).', extra_braces='()')
True
>>> is_wrapped_p('[{word}]?', extra_braces='[]')
True
```

mathmaker.lib.tools.wording.**merge_nb_unit_pairs**(*arg: object*, *w_prefix=''*)

Merge all occurences of {nbN} {*_unit} in arg.wording into {nbN_*_unit}.

In the same time, the matching attribute arg.nbN_*_unit is set with Value(nbN, unit=Unit(arg.*_unit)).into_str(display_SI_unit=True) (the possible exponent is taken into account too).

Parameters **arg** (*object*) – the object whose attribute wording will be processed. It must have a wording attribute as well as nbN and *_unit attributes.

Return type None

Example

```
>>> class Object(object): pass
...
>>> arg = Object()
>>> arg.wording = 'I have {nb1} {capacity_unit} of water.'
>>> arg.nb1 = 2
>>> arg.capacity_unit = 'L'
>>> merge_nb_unit_pairs(arg)
>>> arg.wording
'I have {nb1_capacity_unit} of water.'
>>> arg.nb1_capacity_unit
'\\SI{2}{L}'
```

mathmaker.lib.tools.wording.**post_process**(*sentence: str*)

Apply all desired post processes to a sentence without {tags}.

So far, this is only the replacement of spaces following a number and preceding a word by nonbreaking spaces.

Parameters **sentence** – the sentence to post process

mathmaker.lib.tools.wording.**process_attr_values**(*sentence: str*) → tuple

Build a dict with all {key=val} occurrences in sentence. Update such tags.

All {key=val} occurrences will be replaced by {key}.

Parameters **sentence** – the sentence to process

Returns this couple: (transformed_sentence, {key:val, ...})

mathmaker.lib.tools.wording.**setup_wording_format_of**(*w_object: object*, *w_prefix=''*)

Set w_object's attributes according to the tags found in w_object.wording.

This is the complete process of the wording. w_object.wording will also be modified in the process.

For instance, if w_object.wording is: “Here are one {name}, {nb1} {length_unit1} of roads, and a cube of {nb2} {volume_unit=cm}. What is the side’s length of the cube? |hint:length_unit|”

Then w_object.wording becomes: “Here are one {name}, {nb1_length_unit1} of roads, and a cube of {nb2_volume_unit}. What is the side’s length of the cube?”

w_object.name will be set with a random name,

w_object.nb1_length_unit1 will be set with: ‘\SI{<value of nb1>}{<random length unit>}’

w_object.length_unit will be set to centimeters

w_object.nb2_volume_unit will be set with: ‘\SI{<value of nb2>}{cm^{3}}’

w_object.hint will be set with: ‘cm’

If w_prefix is set, the “wording” processed attributes will be w_object.<prefix>wording and w_object.<prefix>wording_format. This allows to process several different wordings.

Parameters

- **w_object** – The object having a ‘wording’ attribute to process.
- **w_prefix** – The possible prefix of the “wording” attributes to

process.

mathmaker.lib.tools.wording.**unwrapped**(*word: str*) → str

Remove first and last char plus possible punctuation of word.

Examples

```
>>> unwrapped('{word}')
'word'
>>> unwrapped('{word},')
'word'
>>> unwrapped('{word}:')
'word'
```

mathmaker.lib.tools.wording.**wrap**(*word: str*, *braces='{}'*, *o_str=None*, *e_str=None*) → str

Return the word wrapped between the two given strings.

Using o_str and e_str (e.g. opening str and ending str) will override braces content. It’s interesting when one want to wrap the word with something longer than a char.

Parameters

- **word** (*str*) – the chunk of text to be wrapped
- **braces** (*str*) – the pair of braces that will wrap the word
- **o_str** (*str*) – prefix the word.
- **e_str** (*str*) – suffix the word

Return type str

Examples

```
>>> wrap('wonderful')
'{wonderful}'
>>> wrap('wonderful', braces='<>')
'<wonderful>'
>>> wrap('wonderful', o_str='<<', e_str='>>')
'<<wonderful>>'
```

```
>>> wrap('wonderful', e_str='}*')
'{wonderful}*'
```

mathmaker.lib.tools.wording.**wrap_latex_keywords**(*s: str*) → str
Replace some {kw} by {{kw}}, to prevent format() from using them as keys.

mathmaker.lib.tools.xml_sheet module

mathmaker.lib.tools.xml_sheet.**check_q_consistency**(*q_attrib*, *sources*)
(Unfinished) Check the consistency of question's kind, subkind and source.

mathmaker.lib.tools.xml_sheet.**get_attributes**(*filename*, *tag*)
Gathers the attributes of all *filename*'s '*node*'s matching *tag*.

Parameters

- **filename** (*str*) – The XML file name.
- **tag** (*str*) – The tag we're looking for.

Return type list

mathmaker.lib.tools.xml_sheet.**get_exercises_list**(*file_name*)
Retrieves the exercises' list from *file_name*.

Parameters **file_name** (*str*) – The XML file name.

Return type list

mathmaker.lib.tools.xml_sheet.**get_q_kinds_from**(*exercise_node*)
Retrieves the exercise kind and the questions from one exercise section.

Parameters **exercise_node** – The XML node of the exercise.

Return type tuple

mathmaker.lib.tools.xml_sheet.**get_sheet_config**(*file_name*)
Retrieves the sheet configuration values from *file_name*.

Parameters **file_name** (*str*) – The XML file name.

Return type tuple

mathmaker.lib.tools.xml_sheet.**get_xml_schema_path**()

mathmaker.lib.tools.xml_sheet.**get_xml_sheets_paths**()
Returns all paths to default xml frameworks.

They are returned as a dictionary like: {id: path_to_matching_file.xml, ...} the id being the filename without its extension.

Return type dict

Module contents

Submodules

mathmaker.lib.error module

exception mathmaker.lib.error.**ArgumentNeeded**(*data*)
Bases: Exception

exception mathmaker.lib.error.**ImpossibleAction**(*data*)
Bases: Exception

exception mathmaker.lib.error.**MethodShouldBeRedefined**(*objct*, *method*)
Bases: Exception

exception mathmaker.lib.error.**NotImplementedYet**(*method*)
Bases: Exception

exception mathmaker.lib.error.**NotInstanciableObject**(*objct*)
Bases: Exception

exception mathmaker.lib.error.**OutOfRangeArgument**(*objct*, *expected_range*)
Bases: Exception

exception mathmaker.lib.error.**UncompatibleObjects**(*objct1*, *objct2*, *given_action*, *expected_result*)
Bases: Exception

exception mathmaker.lib.error.**UncompatibleOptions**(*opt1*, *opt2*)
Bases: Exception

exception mathmaker.lib.error.**UncompatibleType**(*objct*, *possible_types*)
Bases: Exception

exception mathmaker.lib.error.**UnknownOutputFormat**(*given_format*)
Bases: Exception

exception mathmaker.lib.error.**UnknownXMLTag**(*given_tag*)
Bases: Exception

exception mathmaker.lib.error.**WrongArgument**(*given_arg*, *expected_arg*)
Bases: Exception

exception mathmaker.lib.error.**WrongObject**(*data*)
Bases: Exception

exception mathmaker.lib.error.**XMLFileFormatError**(*msg*)
Bases: Exception

mathmaker.lib.is_ module

mathmaker.lib.is_.**a_natural_int**(*objct*)

mathmaker.lib.is_.**a_number**(*objct*)

mathmaker.lib.is_.**a_numerical_string**(*objct*)

mathmaker.lib.is_.**a_sign**(*objct*)

mathmaker.lib.is_.**a_string**(*objct*)

mathmaker.lib.is_.**a_string_list**(*objct*)

mathmaker.lib.is_.**an_int**(*objct*)

mathmaker.lib.is_.**an_integer**(*objct*)

mathmaker.lib.is_.**an_ordered_calculable_objects_list**(*provided_list*)

mathmaker.lib.list_sheets module

mathmaker.lib.list_sheets.**list_all_sheets**()

Creates the list of all available mathmaker's sheets.

The list is displayed as a tabular.

Returns The list as str

mathmaker.lib.maths_lib module

mathmaker.lib.maths_lib.**abs**(*nb*)

mathmaker.lib.maths_lib.**barycenter**(*points_list*, *barycenter_name*, *weights=None*)

mathmaker.lib.maths_lib.**coprime_generator**(*n*)

mathmaker.lib.maths_lib.**correct_normalize_results**(*d*)

mathmaker.lib.maths_lib.**deg_to_rad**(*arg*)

mathmaker.lib.maths_lib.**gcd**(*a*, *b*)

mathmaker.lib.maths_lib.**generate_decimal**(*width*, *ranks_scale*, *start_rank*)

mathmaker.lib.maths_lib.**is_even**(*objct*)

mathmaker.lib.maths_lib.**is_uneven**(*objct*)

mathmaker.lib.maths_lib.**lcm**(*a*, *b*)

mathmaker.lib.maths_lib.**lcm_of_the_list**(*l*)

mathmaker.lib.maths_lib.**mean**(*numberList*, *weights=None*)

mathmaker.lib.maths_lib.**pupil_gcd**(*a*, *b*)

mathmaker.lib.maths_lib.**round**(*d*, *precision*, ***options*)

mathmaker.lib.maths_lib.**sign_of_product**(*signed_objctlist*)

mathmaker.lib.maths_lib.**ten_power_gcd**(*a*, *b*)

mathmaker.lib.randomly module

mathmaker.lib.randomly.**coprime_to**(*n*, *range*)

mathmaker.lib.randomly.**coprime_to_the_first**(*n*, *p*, *range*)

mathmaker.lib.randomly.**decimal_0_1**()

mathmaker.lib.randomly.**heads_or_tails**()

mathmaker.lib.randomly.**integer**(*min_value*, *max_value*, ***options*)

mathmaker.lib.randomly.**mix**(*objects_list*)

mathmaker.lib.randomly.**not_coprime_to**(*n*, *range*, ***options*)

mathmaker.lib.randomly.**pop**(*provided_list*, ***options*)

mathmaker.lib.randomly.**sign**(**options*)

mathmaker.lib.shared module

mathmaker.lib.shared.**init**()

mathmaker.lib.sources module

mathmaker.lib.sources.**generate_values**(*source_id*)

class mathmaker.lib.sources.**mc_source**
Bases: object

next(*source_id*, ***kwargs*)

class mathmaker.lib.sources.**sub_source**(*source_id*)
Bases: object

next(***kwargs*)

mathmaker.lib.startup_actions module

This module gathers functions that should be run at startup.

These functions check dependencies, settings consistency and setup the language for gettext translations.

mathmaker.lib.startup_actions.**check_dependencies**(*euktoeps='euktoeps'*, *xmllint='xmllint'*, *lualatex='lualatex'*, *luaotfload_tool='luaotfload-tool'*) → bool
Will check all mathmaker's dependencies.

mathmaker.lib.startup_actions.**check_dependency**(*name: str*, *goal: str*, *path_to: str*, *required_version_nb: str*) → bool
Will check if a dependency is installed plus its version number.

The version number is supposed to be displayed at the end of the line containing 'version' when calling *executable –version* (or the equivalent).

Parameters

- **name** (*str*) – the dependency's name.
- **goal** (*str*) – tells shortly why mathmaker needs it for
- **path_to** (*str*) – the path to the executable to test
- **required_version_nb** (*str*) – well, the required version number

Return type bool

mathmaker.lib.startup_actions.**check_font**() → bool
Will check if settings.font belongs to data/fonts_list.txt.

It will first check if the exact name is in the list, then if one line of the list starts with the exact name.

mathmaker.lib.startup_actions.**check_settings_consistency**(*language=None*, *od=None*)
Will check the consistency of several settings values.

The checked values are: whether the language is supported as a LaTeX package that mathmaker uses, the output directory (is it an existing directory?) and whether the chosen font is usable by lualatex.

mathmaker.lib.startup_actions.**install_gettext_translations**(***kwargs*)
Will install output's language (gettext functions).

mathmaker.lib.startup_actions.**warning_msg**(*name: str*, *path_to: str*, *c_out: str*, *c_err: str*, *gkw: str*, *g_out: str*, *g_err: str*)
Return the formatted warning message.

Parameters

- **name** – name of the software
- **path_to** – the path to the software
- **c_out** – output of the call to *software –version*
- **c_err** – error output of the call to *software –version*
- **gkw** – keyword used to grep the version from output
- **g_out** – output of the call to *grep...*
- **g_err** – error output of the call to *grep...*

Module contents

mathmaker.settings package

Module contents

class mathmaker.settings.**ContextFilter**(*name=''*)
Bases: logging.Filter

Removes the 'dbg.' at the beginning of logged messages.

filter(*record*)

mathmaker.settings.**config_dbglogger**(*sd*)
Configures dbg_logger, using to the configuration file values.

class mathmaker.settings.**default_object**
Bases: object

mathmaker.settings.**init**()

class mathmaker.settings.**path_object**(***dirs*)
Bases: object

Submodules

mathmaker.cli module

mathmaker.cli.**entry_point**()

Module contents

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

m

Index

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y