SayTeX

Nov 08, 2019

Contents

CHAPTER 1

SayTeX

SayTeX can convert natural language math expressions into valid LaTeX.

The SayTeX project consists of one main product: the `saytex` Python package, which provides a simple interface for converting natural language into LaTeX.

The SayTeX website is at https://saytex.xyz and all code is hosted on GitHub: https://github.com/arvid220u/saytex.

CHAPTER 2

Documentation Index

2.1 Getting Started

2.1.1 Installation

SayTeX exists as a Python 3 package, which allows you to use the SayTeX functionality from any Python module or script.

Install the `saytex` package using PyPI:

```
pip install saytex
```

Note that the `saytex` package only supports Python 3. If the above command gives an error, run `pip -V` and make sure it says Python 3. If it does not, try running `pip3` instead.

2.1.2 Simple Usage

The `saytex` Python package consists mainly of the `Saytex` class. It contains the method `to_latex` which takes a string as its input and outputs a well-formatted LaTeX string.

A simple example of how to use the `Saytex` class is shown below:

```
from saytex import Saytex

saytex_compiler = Saytex()

print(saytex_compiler.to_latex("a squared plus b"))
```

As one would expect, the above code prints `a^2 + b`, which is the valid LaTeX translation of that string.

For notes on how to configure SayTeX to suit specific needs, see the *Advanced Usage*.

2.2 Philosophy

To convert spoken math to LaTeX, we need to deal with two different problems: (1) that LaTeX commands use special symbols and commands that are hard to express in spoken language, and (2) that when math is spoken, there are usually several ways to say the same thing. Trying to solve these two problems with only one method is deemed to at best be cluttered and at worst fail to solve either of the two problems. Instead, SayTeX solves each problem separately, through the introduction of the intermediary SayTeX Syntax.

When SayTeX receives a math string, it first converts it heuristically to SayTeX Syntax, which is then converted unambiguously to LaTeX. Enforcing this split, along with a few other key invariants, enables SayTeX to solve both of the previously outlined problems at once.

2.2.1 SayTeX Syntax

SayTeX Syntax can be thought of as a pronounceable version of LaTeX. The following list outlines some key properties that guided the design of SayTeX Syntax:

1. One-to-one mapping to LaTeX. No exceptions.
2. Easy to pronounce.
3. No special symbols.
4. Only words and numbers.
5. Case sensitive.
6. Linear-time conversion to LaTeX.

The full specifications for SayTeX Syntax can be found in *SayTeX Syntax Specifications*. Some examples are listed below.

- `a plus b superscript 2` = `a + b^2`
- `fraction begin x plus y end begin pi end` = `\frac{x + y}{\pi}`
- `inverse hyperbolic cotangent left parenthesis x right parenthesis` = `\arcoth \left( x \right)`

It is simple to convert from SayTeX Syntax to LaTeX using the `saytex` package:

```
from saytex import SaytexSyntax

saytex_syntax = SaytexSyntax()

print(saytex_syntax.to_latex("a plus b superscript 2"))
```

Note that since SayTeX Syntax should be in one-to-one correspondence with LaTeX, any LaTeX can be converted into SayTeX Syntax, which could potentially be useful for screen readers when encountering LaTeX. The `to_saytex` method is not yet implemented, however.

2.2.2 Natural Language to SayTeX Syntax

SayTeX Syntax addresses the problem of making LaTeX easily pronounceable. As can be seen in the examples above, however, SayTeX Syntax is restrictive and verbose, making it less than ideal for regular use. SayTeX therefore employs a number of heuristics to convert common ways of expressing math into SayTeX Syntax.

SayTeX does not enforce any formal restrictions on its input. Rather, the set of recognizable strings is implementation-specific.

The natural language to SayTeX conversion is implemented using layers; each layer takes as input the output of the layer preceding it. More details on the implementation can be found in *Advanced Usage*. The following is the list of default layers:

- Correct common speech recognition mistakes, such as "eggs" instead of "x".
- Transform the input to lowercase.
- Recognize the word "capital" to capitalize the next word. For example, "capital a" would become "capital A".
- Recognize spoken numbers, such as "three hundred and fifty six" for "356".
- Convert synonyms into the canonical SayTeX Syntax version. For example, "multiplied by" should become "centered dot".
- Recognize expressions on the form "integral from ... to ... of ...".
- Convert "a over b" into "fraction begin a end begin b end".
- Prettify the output, e.g. by inserting a space before the "dx" in an integral.

Each of these layers can be disabled if so desired, and users of the package can easily add their own layers. An important note is that all valid SayTeX Syntax should remain valid after passing through all layers. By enforcing this, we guarantee that virtually any LaTeX expression can be recognized by SayTeX, while at the same time providing shorthand syntax for the most common usecases.

2.3 Advanced Usage

Note: It is recommended to read *Philosophy* before this section.

2.3.1 Interfacing with SayTeX Syntax

A recognizable SayTeX string can be compiled into a SayTeX Syntax string, as shown below.

```
from saytex import Saytex

saytex_compiler = Saytex()

print(saytex_compiler.to_saytex("a plus b to the power of two"))
```

A valid SayTeX Syntax string can then be compiled into LaTeX:

```
from saytex import SaytexSyntax

saytex_syntax = SaytexSyntax()

print(saytex_syntax.to_latex("a plus b superscript 2"))
```

Note that cascading these two methods produces exactly the same result as the `saytex.Saytex.to_latex` method.

A SayTeX Syntax string can be validated using the `saytex.saytexsyntax.SaytexSyntax.is_valid_saytex_syntax` method.

In-depth reference can be found in _saytexsyntaxreference.

2.3.2 Configuring SayTeX Layers

The SayTeX-to-SayTeX-Syntax conversion is done through layers, with each layer's output being the input to the next layer. The `Saytex` class comes with a number of pre-defined layers (as outlined in *Philosophy*), which are good for usage in a general speech-recognition setting, but might not be perfect in all usecases.

To remove an existing layer, use the `Saytex.remove_layer` function:

```
from saytex import Saytex
import saytex.layers.prettification.PrettificationLayer as PrettificationLayer

saytex_compiler = Saytex()

saytex_compiler.remove_layer(PrettificationLayer)
```

One can also create completely new layers by subclassing `saytex.layers.Layer` and implementing the `execute_layer` function:

:: import saytex.layers.Layer

    class ExampleLayer(saytex.layers.Layer):

        def execute_layer(input_string): “”” This example layer adds “implies zero equals zero” to all input strings. “”” return input_string + ” implies zero equals zero”

This layer can then be added to the `saytex_compiler`. When doing that, a priority needs to be specified, which indicate where in the sequence of layers the new layer should be executed. The default priorities can be found here.

```
saytex_compiler.add_layer(ExampleLayer, 4)
```

After this, when running `saytex_compiler.to_latex` or `saytex_compiler.to_saytex`, the compiler will not prettify the output, but it will always add “implies zero equals zero” to all

2.4 SayTeX Syntax Specifications

As outlined in *Philosophy*, SayTeX Syntax is a strictly defined language in one-to-one correspondence with LaTeX. In this section, the exact specifications for SayTeX Syntax will be outlined.

2.4.1 Design Principles

A SayTeX Syntax command consists of at most 7 space-separated words. Each word consists only of the letters a-z, or is a number using the digits 0-9. All strings using any other characters do not conform to SayTeX Syntax.

Commands should generally read as if in a sentence. That is, most relations will take the form of verbs instead of nouns, such as “<” being “is less than” rather than just “less than (sign)”. Also, commands should be as succinct as possible when there are two equally valid ways of saying things; for example, “equals” is preferred over “is equal to.”

2.4.2 Specific Commands

The specific commands of SayTeX Syntax are specified in json format in multiple files in *saytex/saytexsyntax/saytex_dictionary*. The different json files will be concatenated by the compiler, and the reason for having multiple ones is purely organizational.

Each of the json files must be a list of dictionaries, where each dictionary contains the following elements: - *saytex*: a string of the saytex command. Required. - *latex*: a string of the corresponding latex command. Required. -

left_space: an integer indicating space preference. If 0, a space should not be added to its left side, if 1, a space should occasionally be added, and if 2, a space should always be added. Optional; defaults to 1. - *right_space*: an integer indicating space preference. Same format as *left_space*. - *insert_curly_brackets_right*: a boolean indicating whether or not the command (operator) should require the word on its right to be encapsulated in curly braces (used by e.g. ^). Optional; defaults to *false*.

2.5 Development Notes

This section should be of little interest to everyone who is not a maintainer of SayTeX.

2.5.1 Deploying

Update the version number in `setup.py`.

Remember to update the documentation, as per the instructions below.

Commit with the message `version x.x.x`.

Build the package: `python3 setup.py sdist bdist_wheel`.

Upload it to PyPI: `twine upload --skip-existing dist/*`.

Then, make a new release on GitHub, where the binaries from `dist/` are uploaded.

2.5.2 Documentation

Install sphinx: `pip3 install sphinx`

Install theme: `pip3 install sphinx_rtd_theme`

To update the docs, first update the local installation of saytex by running `pip3 install -e .` from the project directory. Then go to the `docs` directory and run `sphinx-apidoc -o . ../saytex -f`, then `rm -rf _build` followed by `make html`.

2.6 saytex package

2.6.1 Subpackages

saytex.layers package

Submodules

saytex.layers.capitalization module

Recognizes the keyword “capital” to capitalize the following word.

class saytex.layers.capitalization.**CapitalizationLayer**

Bases: *saytex.layers.layer.SaytexLayer*

execute_layer(*input_string*)

Transforms “capital a” into “A”, for example.

saytex.layers.case_insensitivity module

Makes the input treated as case insensitive, which is what it should be if coming from spoken language.

class saytex.layers.case_insensitivity.**CaseInsensitivityLayer**
Bases: *saytex.layers.layer.SaytexLayer*

execute_layer(*input_string*)
Transforms all capital letters into lowercase letters.

saytex.layers.divided_by_recognition module

Recognizes the word 'over' and transforms it into a properly formatted fraction (fraction begin ... end begin ... end)

class saytex.layers.divided_by_recognition.**DividedByRecognitionLayer**
Bases: *saytex.layers.layer.SaytexLayer*

execute_layer(*input_string*)
Transforms "a over b" into "fraction begin a end begin b end".

find_associativity_left(*words*, *over_index*)
Finds the index of the start word of the numerator, using some heuristic.

Parameters

- **words** – list of words
- **over_index** – index where the word 'over' is in the words list

Returns index i such that words[i:over_index] are the numerator

find_associativity_right(*words*, *over_index*)
Finds the index of the end word of the denominator, using some heuristic.

Parameters

- **words** – list of words
- **over_index** – index where the word 'over' is in the words list

Returns index i such that words[over_index+1:i+1] are the denominator

saytex.layers.divided_by_recognition.**findmatching**(*words*, *pos*, *w1='left parenthesis'*, *w2='right parenthesis'*, *forward=True*)
Find matching parantheses.

Parameters

- **words** – list, to be searched in
- **pos** – int, position of (we want to find a match for

Returns int, position of matching) bracket; if there is no match, return len(s)

saytex.layers.from_to_recognition module

Recognizes from-to sequences such as "integral from ... to ..." and "sum from ... to ...".

class saytex.layers.from_to_recognition.**FromToRecognitionLayer**
Bases: *saytex.layers.layer.SaytexLayer*

execute_layer(*input_string*)
Replaces "from … to …" with "subscript begin … end superscript begin .. end"

symbolify_fromto(*s*)
Finds a from-to-sequence and its corresponding sum or integral. We assume that everything between the words "from" and "to" is the from part. We assume that the first simple expression after "to" is the to part. Simple expression is defined as at most two variables and one operation. If parantheses, it uses that. In the case where there is no matching "to" word for a "from" word, we assume that the entirety of the rest of the string is the from part. The return string is formatted as sum subscript begin expression1 end superscript begin expression2 end. ('sum' can be substituted for 'integral'.)

exception saytex.layers.from_to_recognition.**NoFromTo**
Bases: Exception

Raised if there is no from-to sequence in a string.

saytex.layers.from_to_recognition.**findmatching**(*s*, *pos*, *w1='left parenthesis'*, *w2='right parenthesis'*)
Find matching parantheses.

Parameters

- **s** – string, to be searched in
- **pos** – int, position of (we want to find a match for

Returns int, position of matching) bracket; if there is no match, return len(s)

saytex.layers.from_to_recognition.**makeword**(*s*)

saytex.layers.handle_of module

Handles the word "of". It should either be interpreted as a function, say f(x) as "f of x", or it should be ignored (e.g., "integral of x")

class saytex.layers.handle_of.**HandleOfLayer**
Bases: *saytex.layers.layer.SaytexLayer*

execute_layer(*input_string*)
Handles the word "of" appropriately.

saytex.layers.layer module

Defines the base layer interface to be used by all layers.

exception saytex.layers.layer.**InvalidLayerAccess**
Bases: Exception

Raised in SaytexLayer.get_layer if someone tries to access an invalid layer.

class saytex.layers.layer.**SaytexLayer**
Bases: object

execute_layer(*input_string*)
To be overridden in a layer subclass.

Parameters **input_string** – str, the input

Returns str, the output of the layer

replace_words(*word_tuples*, *input_string*)
Replaces words in input_string using the word_tuples.

Parameters

- **word_tuples** – a list of tuples of two strings, where the first one is the word to be replaced and the second one is the word to replace it with
- **input_string** – str

Returns str, with words replaced

saytex_syntax_operators()
Returns a set of all recognized operators in SayTeX Syntax.

saytex.layers.prettification module

Does various things to produce prettier LaTeX strings, such as inserting spaces into integrals.

class saytex.layers.prettification.**PrettificationLayer**
Bases: *saytex.layers.layer.SaytexLayer*

execute_layer(*input_string*)
Adds spaces to integrals to produce prettier equations.

saytex.layers.speech_recognition_error_correction module

Corrects errors in the speech recognition software used. Currently specific to the Microsoft speech engine, and might not produce good results if used with other speech recognition software.

class saytex.layers.speech_recognition_error_correction.**SpeechRecognitionErrorCorrectionLay**
Bases: *saytex.layers.layer.SaytexLayer*

aggressive = True

common_mischaracterizations = [('see', 'c'), ('hey', 'a'), ('day', 'a'), ('overbyte',

common_mischaracterizations_aggressive = [('some', 'sum'), ('be', 'b'), ("I'm", 'n'),

execute_layer(*input_string*)
Replaces the mischaracterizations with guesses for what the user actually said. Not doing anything smart really, only using word replacements. Could easily be modified to be smarter.

saytex.layers.spoken_number_recognition module

Recognizes spoken numbers and transforms them into number literals.

class saytex.layers.spoken_number_recognition.**SpokenNumberRecognitionLayer**
Bases: *saytex.layers.layer.SaytexLayer*

execute_layer(*input_string*)
Transforms spoken numbers into number literals. For example, “five hundred and five” is transformed into “505”.

saytex.layers.synonym_standardization module

Maps common ways of expressing math formulas into the specific SayTeX Syntax way of saying things.

exception saytex.layers.synonym_standardization.**InvalidSynonymStandardizationDictionary**
Bases: Exception

class saytex.layers.synonym_standardization.**SynonymStandardizationDictionary**(*syntax_file=None*, *syntax_directory=None*)
Bases: object

Works as an interface for the synonym standardization dictionary.

get_standard_synonym(*word*, *params={}*)
Returns the corresponding latex code for saytex, and None if saytex is an invalid command. The latex code will be space padded according to the value in the dictionary.

Parameters

- **saytex** – str, containing a potential saytex command
- **params** – dict, containing params to be passed to post_process_latex

Returns a post processed str, or None if saytex is not a valid command

get_syntax_entry(*word*)
Returns a syntax entry corresponding to saytex, if exists. Otherwise, raises KeyError.

Parameters **saytex** – str, containing a potential saytex command

Returns a dictionary in self.syntax_list

load_syntax()
Loads the syntax defined in self.syntax_file or self.syntax_directory into the array self.syntax_list and the dictionary self.syntax_dictionary.

class saytex.layers.synonym_standardization.**SynonymStandardizationLayer**
Bases: *saytex.layers.layer.SaytexLayer*

convert_synonyms(*saytex_string*, *word_list=None*, *word_index=0*, *dp_memo=None*)
Converts SayTeX Syntax into LaTeX code.

Parameters

- **saytex_string** – A string containing valid SayTeX Syntax code.
- **(optional)** (*word_list*) – A list of tokenized words derived from saytex_string. If word_list is not None, then the saytex_string will be ignored. Is normally only used for recursive calls.
- **word_index** – The current index in the word_list that we are at. Used in recursive calls. If it is not valid SayTeX Syntax, a SaytexSyntaxError will be raised.
- **dp_memo** – A dictionary mapping indices in the word_list to generated LaTeX strings.
- **next_params** – A dictionary of parameters to be passed to the next get_latex call.

Returns A string containing a valid translation of the input string into LaTeX (if no exception). If word_index > 0, the return value is a tuple (string, value) where value is a measure of how good the string is as a LaTeX translation of saytex_string.

execute_layer(*input_string*)
Converts common synonyms into SayTeX Syntax equivalents.

exception saytex.layers.synonym_standardization.**UnrecognizedSynonym**
Bases: Exception

Module contents

Defines all layers, used for converting natural language into SayTeX Syntax. All layers should subclass the layer interface defined in layers.py.

saytex.saytexsyntax package

Submodules

saytex.saytexsyntax.compiler module

Defines the SaytexSyntax class, containing methods for converting between SayTeX and LaTeX.

exception saytex.saytexsyntax.compiler.**LatexParsingError**
Bases: Exception

Raised in from_latex, if error.

exception saytex.saytexsyntax.compiler.**MultipleSaytexInterpretations**
Bases: Exception

Raised in to_latex, if error.

class saytex.saytexsyntax.compiler.**SaytexSyntax**
Bases: object

Contains methods to_latex and from_latex for converting between LaTeX and SayTeX syntax. Also supports reloading of the syntax dictionary.

compute_latex(*word_list*, *word_index*, *dp_memo*, *next_params*)
Converts the string formed by the words word_list[word_index:] into LaTeX, and returns a score of how good the conversion is. The score is proportional to the number of non-keywords used in the conversion, and a lower score is better. Assumes that the string formed by word_list is valid SayTeX Syntax.

Parameters

- **word_list** – A list of tokenized words derived from saytex_string.
- **word_index** – The current index in the word_list that we are at.
- **dp_memo** – A dictionary mapping indices in the word_list to generated LaTeX strings.
- **next_params** – A dictionary of parameters to be passed to the next get_latex call.

Returns A tuple (string, value) where value is a measure of how good the string is as a LaTeX translation of saytex_string.

is_valid_saytex_syntax(*potential_saytex_string*)
Determines if a string is valid SayTeX Syntax or not.

Parameters **potential_saytex_string** – str, potentially conforming to SayTeX syntax

Returns bool, indicating whether the str actually conforms to SayTeX syntax or not

load_syntax_dictionary(*syntax_directory*)
Load the syntax dictionary as self.syntax_dictionary.

to_latex(*saytex_string*)
Converts SayTeX Syntax into LaTeX code.

Parameters **saytex_string** – A string containing valid SayTeX Syntax code.

Returns A string containing a valid translation of the input string into LaTeX (if no exception).

exception saytex.saytexsyntax.compiler.**SaytexSyntaxError**
Bases: Exception

Raised in to_latex, if error.

saytex.saytexsyntax.config module

saytex.saytexsyntax.config.**ALLOWED_CHARACTERS_IN_SAYTEX_WORD = {'A', 'B', 'C', 'D', 'E', 'I**
This constant should correspond to the most recent definition of the SayTeX Syntax. Note that the specification allows for both words and numbers, the latter of which are not restricted to the characters below.

saytex.saytexsyntax.config.**MAX_WORDS_PER_SAYTEX_COMMAND = 7**
This constant should correspond to the most recent definition of the SayTeX Syntax. Note that the total time of the SayTeX Syntax Compiler will be O(n*m) if m is the variable below, so be careful. (Also, it will be O(n^2).)

saytex.saytexsyntax.config.**SYNTAX_DIRECTORY = 'saytex_dictionary'**
The path to the directory containing all syntax files. The specifications for the files can be found in saytex-syntax-v01.md.

saytex.saytexsyntax.syntax_dictionary module

Defines a SyntaxDictionary class which handles reading from a SayTeX Syntax file.

exception saytex.saytexsyntax.syntax_dictionary.**InvalidSyntaxFile**
Bases: Exception

Raised if the syntax json file is not of valid format.

class saytex.saytexsyntax.syntax_dictionary.**SyntaxDictionary**(*syntax_file=None*, *syntax_directory=None*)
Bases: object

Represents a Syntax Dictionary.

get_latex(*saytex*, *params={}*)
Returns the corresponding latex code for saytex, and None if saytex is an invalid command. The latex code will be space padded according to the value in the dictionary.

Parameters

- **saytex** – str, containing a potential saytex command
- **params** – dict, containing params to be passed to post_process_latex

Returns a post processed str, or None if saytex is not a valid command

get_next_params(*saytex*)
Returns a (possibly empty) dictionary containing the params that should be passed to the next call of post_process_latex

Parameters **saytex** – str, containing a potential saytex command

get_syntax_entry(*saytex*)
Returns a syntax entry corresponding to saytex, if exists. Otherwise, raises KeyError.

Parameters **saytex** – str, containing a potential saytex command

Returns a dictionary in self.syntax_list

load_syntax()
Loads the syntax defined in self.syntax_file or self.syntax_directory into the array self.syntax_list and the dictionary self.syntax_dictionary.

make_syntax_entry_default(*syntax_item*)
Modifies the supplied syntax entry d, to add default values for parameters.

Parameters **syntax_item** – A dictionary representing a syntax entry to be modified in place.

post_process_latex(*syntax_entry*, *insert_curly_brackets=False*)
Returns the post processed version of the syntax_entry, by adding spacing or curly brackets.

Parameters

- **syntax_entry** – a dictionary that could be a member of syntax_list
- **insert_curly_brackets** – bool, indicating whether the latex command should be curly bracket padded instead of space padded

Returns str, the post processed version

exception saytex.saytexsyntax.syntax_dictionary.**UnrecognizedSaytexCommand**
Bases: Exception

Module contents

The saytexsyntax package provides the SaytexSyntax class, which can be used for converting between SayTeX Syntax and LaTeX.

2.6.2 Submodules

2.6.3 saytex.compiler module

Defines the Saytex class, containing methods for converting between natural language and LaTeX. The conversion is done in a two-step process by first translating the input into SayTeX Syntax, after which the saytexsyntax module is used for the final LaTeX conversion.

class saytex.compiler.**Saytex**
Bases: object

Contains the method to_latex to convert from natural language to LaTeX. It will do this by invoking the layers defined in config.py in the specified order, followed by a final call to SaytexSyntax. The method to_saytex will do the same thing, without the call to SaytexSyntax.

add_layer(*layer_class*, *layer_priority*)
Adds a layer to the natural language -> Saytex Syntax conversion process.

Parameters

- **layer_class** – The class of the layer, which must be a subclass of saytex.layers.layer.SaytexLayer. Note that this parameter is not a string, but rather the class itself.

- **layer_priority** – The priority of the layer, represented as a number. The priority affects in which order the layers are executed, and a lower number means it is executed sooner. The priorities only make sense in relation to other priorities, and the default priorities can be found in saytex.config.

Returns None

get_layer_priorities()
Returns a dictionary mapping the currently used layers to their priorities.

Returns Dictionary of layers to numbers.

get_layers()
Returns the set of currently used layers.

Returns Set of layers.

remove_layer(*layer_class*)
Removes a layer from the conversion process. Can be used to remove default layers.

Parameters **layer_class** – The class of the layer, which must be a subclass of saytex.layers.layer.SaytexLayer. Note that this parameter is not a string, but rather the class itself.

Returns None

to_latex(*math_string*)
Converts natural language into LaTeX code.

Parameters **math_string** – A string containing a spoken math expression. The string must be recognizable to SayTeX, of which the specifics depend on the particular layers that are used. The set of all recognizable SayTeX strings is a superset of SayTeX Syntax.

Returns A string containing a valid translation of the input string to LaTeX code. If the string is not recognizable (that is, it cannot be converted into SayTeX), the UnrecognizableSaytexInput exception is thrown.

to_saytex(*math_string*)
Converts natural language into SayTeX Syntax.

Parameters **math_string** – A string containing a spoken math expression. The string should use the SayTeX+ format, which is a superset of SayTeX Syntax.

Returns A string containing a valid translation of the input string to SayTeX Syntax. If the string is not recognizable (that is, it cannot be converted into SayTeX), the UnrecognizableSaytexInput exception is thrown.

exception saytex.compiler.**UnrecognizableSaytexInput**
Bases: Exception

Raised in to_latex and to_saytex, if the input cannot be transformed into valid SayTeX Syntax.

2.6.4 saytex.config module

This configuration file contains the default layers as well as the default layer priorities. Changing it will change the default for all Saytex instances. To change the layers for a specific Saytex instance, look into the documentation on add_layer and remove_layer of Saytex.

saytex.config.**default_layer_priorities = {<class 'saytex.layers.speech_recognition_error_co**
The layer_priorities is a dictionary mapping layer classes to a number, reflecting the in which order the layers should be executed. Layers are executed from low to high priority. Layers with the same priority can be executed

in any order; the idea is that such layers are independent of each other. In designing new layers, one should strive for not adding a new priority level.

saytex.config.**default_layers = {<class 'saytex.layers.from_to_recognition.FromToRecognitio**

The used_layers is a set containing the default layers to be used by Saytex when converting natural language into SayTeX Syntax. Changing the layers that are in use will affect what spoken math expressions that SayTeX can recognize. Each layer must be a subclass of saytex.layers.layer.SaytexLayer. Creating new layers is as simple as creating a new subclass of saytex.layers.layer.SaytexLayer, and then adding it to the default_layers and default_layer_priorities, or just adding it on a case-by-case situation to a Saytex instance.

2.6.5 Module contents

The saytex package mainly consists of the Saytex class, used for converting from natural language to LaTeX, as well as the SaytexSyntax class, used for converting from the intermediary SayTeX Syntax into LaTeX.

Python Module Index

S

Index

H

I

L

M

N

P

R

S

T

U