Grascii

chanicpanic

Apr 20, 2024

CONTENTS:

CHAPTER

ONE

GRASCII

1.1 About the Project

Grascii is a language used to represent Gregg Shorthand forms using the ASCII character set (characters found on a standard keyboard). The Grascii Project, also referred to as Grascii, encompasses the set of tools and resources accompanying the language that facilitate the reading, writing, and study of Gregg Shorthand at all levels.

1.2 Useful Links

- Full Documentation (readthedocs)
- Additional Dictionaries
- grascii-gui (graphical interface for Grascii Search)

1.3 Made With

- Python 3

1.4 Getting Started

1.4.1 Prerequisites

- Python 3.7+

1.4.2 Installation

Install the package:

```
$ python -m pip install grascii[interactive]
```

Note: We recommend the interactive extra for the majority of users. You may omit the interactive extra when using the package as a library to reduce dependencies. Also see grascii-gui for a graphical interface for Grascii Search.

Verify the installation:

```
$ grascii --help
```

If the command fails, your PATH may not contain the location of Python scripts.

You can also try:

```
$ python -m grascii --help
```

1.5 Grascii Language

The Grascii Language aims to be straightforward for those who are familiar with Gregg Shorthand. That is, Grascii represents most strokes with the letters that match their sounds. For example, the word `Cross` is written as `KROS`.

For a more detailed overview of the language, see language.

1.6 Grascii Search

Grascii Search is the headline tool of the Grascii Project. It provides many useful options for searching Grascii Dictionaries (reverse Gregg Shorthand dictionaries).

1.6.1 Motivation

The existence of shorthand dictionaries have aided the conversion of longhand to shorthand. However, the reverse has remained a challenge since the inception of Gregg Shorthand. Grascii Search solves this problem by allowing users to identify the longhand corresponding to a shorthand form by performing a search based on its Grascii representation.

1.6.2 Basic Usage

Ex.:

```
$ grascii search -g AB
AB About
A|B Agreeable
Results: 2
```

1.6.3 Uncertainty

Occassionally, a stroke is mistaken for one of similar form. Thus, Grascii Search provides levels of uncertainty.

Ex.:

```
$ grascii search -g FND -u1
FND Found
FND Fund
FTH Forth
FTH Further
SND Sound
Results: 5
```

The ND stroke could also be an under TH or an MT/MD. The search accounts for these possibilities with Forth and Further. F is also close to S or V, resulting in Sound.

1.6.4 Interactive Mode

For repeated usage, we recommend running Grascii Search in interactive mode. For more complex queries, interactive mode removes the need of using escape sequences on the command line.

```
$ grascii search -i
```

Note: Requires the interactive extra

1.6.5 More Options

For more options, see search.

1.7 Grascii Dictionary

Grascii comes with a dictionary based on the 1916 Gregg Shorthand Dictionary.

More dictionaries for other versions of Gregg and dictionaries including phrases are available for installation at the Grascii Dictionaries repository.

You can also write, build, and install your own custom dictionaries.

For more information, see dictionary.

1.8 Grascii Dephrase (Experimental)

Grascii includes an experimental phrase parsing module.

It attempts to give the phrase for the most common phrase constructions in Gregg Shorthand and provide suggestions for never before seen phrases:

```
$ python -m grascii.dephrase AVNBA
I HAVE NOT BEEN ABLE
```

1.9 Documentation

Documentation is available on Read the Docs.

1.10 Contributing

Contributions of any kind are welcome and appreciated. You can contribute by:

- Reporting bugs or unexpected behavior
- Fixing bugs and solving issues
- Helping implement new features
- Editing documentation for correctness, completeness, and clarity
- Sharing thoughts and suggestions to improve the Grascii Language

1.10.1 Dictionary

If you find an error in any of the dictionaries, please open an issue or pull request at the dictionaries repository.

Contributions to the dictionaries repository are also welcome to correct errors and create more dictionaries.

1.11 License

This project is under the MIT License.

1.12 Acknowledgements

Many thanks to the developers of Lark, Questionary, appdirs and Qwertigraphy.

1.13 Maintainer's Note

Grascii is not completely stable, but I hope others find the project useful. I try to open draft pull requests with task lists to keep the community informed of upcoming features and the direction of the project. If you notice that there has not been any activity for a couple of weeks, feel free to leave a comment requesting a status update.

– chanicpanic

CHAPTER

TWO

LANGUAGE

2.1 What is Grascii?

Grascii is a language designed to represent Gregg Shorthand forms using the ASCII character set.

It is designed to be intuitive to those already familiar with the system.

Grascii is a context free grammar, and it's implementation can be viewed in grascii.lark.

Grascii is moderately ambiguous. However, as the shorthand system is also ambiguous, it is reasonable that Grascii inherits this attribute.

The current definition of Grascii is based on the Pre-anniversary (1916) version of Gregg Shorthand.

It aims to describe the shorthand forms accurately and succinctly. It also has many additional symbols enabling it to describe some of the lesser used features of the system.

For a summary of what the language does not currently support, see the list below.

Shorthand Form	Grascii Representation(s)	Annotation(s)
	K	
	G	
	R	
	L	
	N	
	M	
	T	
	D	
	TH	( ) ,
	P	
	B	
	F	
	V	
	CH	
	J	
	S, Z	( ) ,
	X	
	SH	,
	‘	
	NG	
	NK	
	LD	

continues on next page

Table 1 – continued from previous page

Shorthand Form	Grascii Representation(s)	Annotation(s)
	A	~ | . ,
	E	~ | . ,
	O	(. ,
	U	) . ,
	EU	
	AU	
	OE	
	I	~ |
	A&E	~ |
	A&’	~ |
	NT, ND	
	MT, MD	
	TN, DN	
	TM, DM	
	MN, MM	
	DT, TD, DD	
	DF, DV, TV	
	SS	
	XS	
	JNT, JND, PNT, PND	

2.2 Annotations

Annotation	Acceptable Tokens	Description
.	A, E, O, U	Denotes the medium sound of the four standard vowel groups.
,	A, E, O, U	Denotes the long sound of the four standard vowel groups.
,	S, Z, TH, SH	Denotes the more obscure sound of the preceeding consonant. Ex. *gas* vs. *gaze*, *breath* vs. *breathe*, *assure* vs. *azure*.
~	A, E, I, A&’, A&E	Denotes that the preceeding circle vowel is reversed.
|	A, E, I, A&’, A&E	Denotes that the preceeding circle vowel is looped.
)	S, Z, TH	When following an S/Z, denotes a right S/Z. When following an TH, denotes an under TH.
(	S, Z, TH	When following an S/Z, denotes a left S/Z. When following an TH, denotes an over TH.
(	O	Denotes an O on its side.
)	U	Denotes an U on its side.
_	A, E, O, U, I, EU, OU, OE, A&’, A&E	Signifies a W sound to be applied before the preceeding vowel.

2.3 Other Symbols

Symbol	Description
^	When placed between tokens, denotes that the two forms are disjoined. When placed at the end of a form, denotes that the preceeding form lies above the line of writing.
-	When placed between grascii forms, denotes that the two characters should not be interpreted as a blended form. Ex. N-T prevents interpretation on NT.

2.4 Examples

Shorthand Form	English	Grascii

2.5 Unsupported Language Features

- Grascii does provide a way of distinguishing between smooth and sharp joinings. There is no plan to make it possible to make this distinction in the future.
- Intersection is currently not implemented. Proposed symbol to denote two intersected characters: \.
- RD is currently not implemented as it does not appear in Gregg 1916, although, it is a form in subsequent versions.
- There is no way of distinguishing the capitalization of a form.
- The under joining/short vowel sound is not included.

CHAPTER

THREE

SEARCH

The core feature of the Grascii suite is search.

Fundamentally, it allows one to enter a Grascii string as a query and search the Grascii dictionary for potential translations.

3.1 Usage

`grascii search ...`

`-h``,` **`--help`**

Print a help message and exit.

`-g` `<grascii>`, **`--grascii`** `<grascii>`

Set a Grascii String to use as a query.

`-e` `<regex>`, **`--regex`** `<regex>`

Set a regular expression to use as a query.

`-r` `<word>`, **`--reverse`** `<word>`

Search by word instead of Grascii.

`-i``,` **`--interactive`**

Run searches in interactive mode. This is the recommended mode for general use, as :option`–grascii` and *`--regex`* may require using shell escape sequences.

`-u` `{0, 1, 2}`, **`--uncertainty`** `{0, 1, 2}`

Set the uncertainty level of a Grascii string. 2 represents the greatest uncertainty.

`-s` `{match,` **`start`**`,` **`contain`**`}`, **`--search-mode`** `{match,` **`start`**`,` **`contain`**`}`

Set the type of search to perform.

`match`: Search for words that closely match the input.

`start`; Search for words that start with the input.

`contain`; Search for words that contain the input.

-a {discard, **retain**, **strict**}, **--annotation-mode** {discard, **retain**, **strict**}

Set how to handle Grascii annotations.

`discard`: Annotations are discarded. Search results may contain annotations in any location.

`retain`: Annotations in the input must appear in search results. Other annotations may appear in the results.

`strict`: Annotations in the input must appear in search results. Other annotations may not appear in the results.

-p {discard, **retain**, **strict**}, **--aspirate-mode** {discard, **retain**, **strict**}

Set how to handle Grascii aspirates.

`discard`: Aspirates are discarded. Search results may contain aspirates in any location.

`retain`: Aspirates in the input must appear in search results. Other aspirates may appear in the results.

`strict`: Aspirates in the input must appear in search results. Other aspirates may not appear in the results.

-j {discard, **retain**, **strict**}, **--disjoiner-mode** {discard, **retain**, **strict**}

Set how to handle Grascii disjoiners.

`discard`: Disjoiners are discarded. Search results may contain disjoiners in any location.

`retain`: Disjoiners in the input must appear in search results. Other disjoiners may appear in the results.

`strict`: Disjoiners in the input must appear in search results. Other disjoiners may not appear in the results.

-n {best, **all**}, **--interpretation** {best, **all**}

How to handle ambiguous Grascii strings.

`best`: Only search with the best interpretation.

`all`: Search with all interpretations.

-f, **--fix-first**

Apply an uncertainty of 0 to the first stroke.

-d <dictionary>, **--dictionary** <dictionary>

Specify which dictionary to search. This option may be used more than once to search multiple dictionaries at the same time.

<`dictionary`> is either a path to the output directory of a built dictionary, or a colon followed by the name of an installed dictionary. Ex: `:preanniversary`.

3.1.1 Suggestions

- use interactive mode
- *`--regex`* is intended for advanced users and advanced searches. Regexes can be difficult to deal with manually, and most users should use *`--grascii`* instead as it handles many of these complications. Using *`--regex`* is effectively equivalent to `$ grep [regex] dict/*`

3.2 Implementation

The search procedure when given a Grascii query is as follows:

1. Convert the Grascii string to uppercase. Parse the Grascii string into tokens and sets of annotations on those tokens.
2. As the Grascii language is ambiguous, all possible parsings are generated.
3. Choose an interpretation (parse). For each interpretation a regular expression is constructed.
4. Each token is replaced with a string of regex alternatives among its equivalent forms and similar forms based on the uncertainty level. To learn how uncertainty is resolved, see similarity.md.
5. In standard mode, modifiers are preserved. Or all possible modifiers for each token are built into the regex which may or may not occur.
6. A set of starting letters is tracked which are the first alphabetic characters required to be accepted by any regex.
7. The dictionary files corresponding to these letters are opened and each line is searched with each regex.
8. Any lines that have a matching regex are returned.

CHAPTER

FOUR

DICTIONARY

Grascii comes with the Grascii forms of all words in the 1916 Gregg Shorthand Dictionary.

These mappings of Grascii strings to their corresponding words are contained in a series of text files in the `dictionaries/builtins` subdirectories.

These dictionary source files are compiled into the dictionary format that Grascii Search expects using `grascii dictionary build`.

4.1 Dictionary Source File Layout

4.1.1 Basic Entry

Each entry in a dictionary source file is contained on its own line in the following scheme:

`[Grascii String] [Translation]`

There can be any amount of whitespace surrounding the `Grascii String` and its `Translation`.

Both `Grascii String` and `Translation` are case-insensitive.

4.1.2 Blank Lines

Blank Lines are ignored

4.1.3 Comments

Lines whose first non-whitespace character is a # are ignored.

```
# This is a comment
```

4.1.4 Uncertainties

An entry preceded by a *?* will produce a warning during the build phase.

```
# I am not sure if that is an A or an E
? ken keen
```

4.2 Source File Conventions

While there is a reasonable amount of freedom in the dictionary source file format, a number of conventions were followed in writing the source files for the dictionary. It is recommended for new files to also follow these conventions.

- Within source files, entries are placed alphabetically by translation.
- When adding entries from a Gregg Shorthand dictionary, a comment denotes the corresponding page and column number in the dictionary. Entries in different pages/columns are separated by a blank line.
- Comments should have *#* as the first character of the line, and there should be a single space following the *#* before the first word of the comment.
- If applicable, *?* should be the first character of the line, and there should be a single space following the *?* before the Grascii string.
- There should be no excess whitespace before or after the Grascii string and its translation. There should be a single space between the Grascii string and its translation.
- Grascii Strings and translations are written in lower case. The case will be adjusted during a build.
- Entries taken from a dictionary are written in Grascii as presented. That is, annotations are not applied unless explicitly displayed. By extension, entries should be written in the simplest form possible. Use annotations only if necessary to distinguish the word from another. This helps generalize the dictionary for better search results.
- The direction annotations on *S* and *TH* are only included if the character is in the direction contrary to its standard joining based on the characters around it.
- Words which include two strokes next to each other that make up a blend, but are not blended, are written with a barrier between them -. While these are stripped in the standard build mode, this information is useful for other build types that may be valuable in the future.
- When writing a stroke that has more than one sound, Use the version that matches the sound it makes in the word.

4.3 The Build Process

4.3.1 Input and Output

The `build` routine takes a set of dictionary source files and outputs a set of text files in the format expected by Grascii Search.

It outputs files of the form: *A*, *B*, *C*, *D*, etc. where each file contains entries whose first alphabetic character in its Grascii form matches the name of the file in which it is contained.

This light indexing reduces the number of entries that Grascii Search must check.

4.3.2 Output File Format

Entries

Each entry in an output file is contained on its own line in the following scheme:

`[GRASCII STRING] [Translation]``

Where `GRASCII STRING` is in all uppercase and `Translation`'s first letter is uppercase, and the rest of the string is lowercase.

There is no whitespace preceding `GRASCII STRING` or following `Translation` . There is exactly one space between them.

Blank Lines

Output files contain no blank lines.

4.4 Building

4.4.1 Usage

`grascii dictionary build [-h] [-o OUTPUT] [-c] [-p] [-s] infiles [infiles ...]`

`<infiles>`
The dictionary source files to compile.

`-h, --help`
Print a help message and exit.

`-o, --output`
Set the directory in which compiled files will be output.

`-c, --clean`
Remove all files in the output directory before compiling.

`-p, --parse`
During the build, all Grascii Strings will be attempted to be parsed to verify that it is a valid Grascii string. If the parse fails, an error will be reported, and the corresponding entry will not be included in the output.

`-w, --words`
Provide a path to a line-separated words file. If provided, all translations will be looked up in the words file to check the spelling/existence of the word. If the word is not found, a warning will be reported, but the corresponding entry will still be included in the output.

`-n, --count`
During the build, all lines are checked to have a single Grascii String followed by a translation of an expected number of words (default 1). If the expected number of words in the translation is less than the actual number of words, a warning will be reported, but the corresponding entry will still be included in the output.

`-k, --check-only`
Only check the input. No output is generated.

`-v, --verbose`
Increase the output verbosity. May be specified up to two times.

4.4.2 Warnings and Errors

During a build, you may encounter warnings and errors.

Warnings indicate that something unusual has been found with an entry. Entries that receive a warning may warrant special attention/review. However, these entries will still be included in the final output.

Errors indicate that there was a failure when processing an entry. Entries that receive an error will not be included in the final output.

Possible Warnings

Uncertainty

Reports that an entry beginning with *?* has been found.

Too many tokens

When the `--count` flag is set, denotes that too many tokens have been found in a source entry. The first word on a line is interpreted as a Grascii string and the rest are interpreted as its translation. By default, the translation is expected to be one word in length. For longer translations, this warning may be silenced by including **[#]* at the beginning of the line (but after *?* if present) where # is the number of words in the translation. Example entry: **2 uer we are*.

Spelling

When a words file is provided with `--words`, denotes that one or more parts of an entry's translation has not been found in the words file.

Possible Errors

Too few tokens

Denotes that there are too few words on a line. A translation may be missing or incomplete.

Invalid Grascii

When the `--parse` flag is set, denotes that the first word is not a valid Grascii string.

Suggestions

Most of the time, it is acceptable to run the build without the `--parse` flag for a quick build. However, it is recommended to run a build with this option and resolve the issues before releasing the dictionary publicly.

The `--count` flag is recommended for standard dictionaries, but may be omitted for phrase dictionaries in which the majority of translations are more than one word in length.

On Unix systems, words files for the `--words` option may be found in */usr/share/dict* or */usr/dict*.

4.5 Working with Custom Dictionaries

It is possible to write your own dictionaries to use with the Grascii tool suite.

1. Make a directory to store your dictionary source files.

```
$ mkdir mysrc
```

2. Add source files to this directory that follow the dictionary source file format.
3. Build your dictionary.

```
$ grascii dictionary build mysrc/*.txt -o mydict
```

Note: At this point, your dictionary is usable.

```
$ grascii search --dictionary ./mydict/ -g AB
```

If you would like to install the dictionary so you do not have to keep track of the path, continue with step 4.

---

4. Install the dictionary.

```
$ grascii dictionary install --name custom ./mydict/
```

5. Verify the installation.

```
$ grascii dictionary list
Built-in Dictionaries:
preanniversary

Installed Dictionaries:
custom
```

6. Enjoy.

```
$ grascii search --dictionary :custom -g AB
```

4.5.1 Uninstalling

Simply run:

```
$ grascii dictionary uninstall custom
```

CHAPTER FIVE

CONFIGURATION

Grascii provides a user-level configuration file to set the defaults for several of its tools.

5.1 Getting Started

Create a configuration file with the following command:

```
$ grascii config --init
```

Locate the file with:

```
$ grascii config --where
```

5.2 Editing the Configuration

To change the defaults, open the generated configuration file and make your desired changes. The available options are described in the default file.

CHAPTER

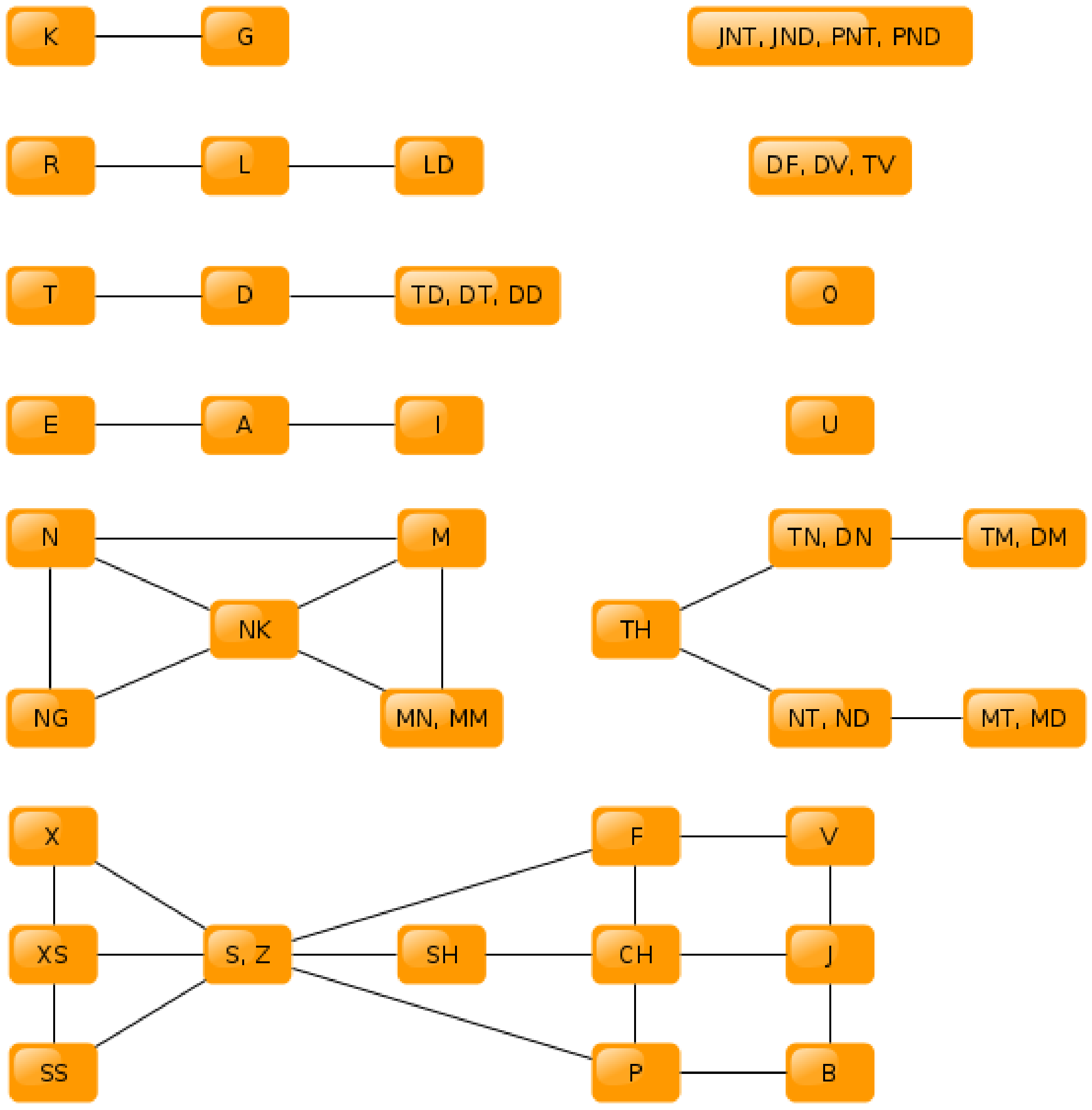
SIX

SIMILARITY RESOLUTION

When running a search, regular expressions are generated with alternatives based on the given tokens. At a basic level, alternatives include equivalent forms of the same token. When uncertainty is greater than 0, similar tokens are also added as alternatives.

The similar tokens are defined by a similarity graph. The set of tokens returned as being similar are all those within a distance equal to the uncertainty from the target node when performing a breadth-first-search.

The similarity graph is shown below.


K
G
JNT, JND, PNT, PND
R
L
LD
DF, DV, TV
T
D
TD, DT, DD
O
E
A
I
U
N
M
TN, DN
TM, DM
NK
TH
NG
MN, MM
NT, ND
MT, MD
X
F
V
XS
S, Z
SH
CH
J
SS
P
B

CHAPTER

SEVEN

CHANGELOG

7.1 Unreleased

7.1.1 Added

- Dictionary build `--no-output` option
- `DictionaryOutputOptions` class for `DictionaryBuilder.build`
- `BuildSummary` class for results of `DictionaryBuilder.build`
- Experimental pipelines for dictionary builds
- `ignore_case` option to `GrasciiValidator`

7.1.2 Changed

- `Searcher.__init__` does not handle `DictionaryNotFound` exceptions
- `grascii search` prints an error if a dictionary cannot be found
- Many `DictionaryBuilder.__init__` options moved to `DictionaryBuilder.build` or were removed
- `DictionaryBuilder.build` takes `infiles` and `output` arguments and returns a `BuildSummary`

7.1.3 Removed

- `grascii.grammars.get_grammar`: Use `Lark.open_from_package("grascii.grammars", grammar_name)` instead.
- `BuildDirectory` configuration file option
- Dictionary build `--check-only` option: Use `--no-output` instead
- `grascii.dictionary.build.build` function: Use `DictionaryBuilder.build` instead

7.2 0.5.0 - 2022-08-12

7.2.1 Added

- `SearchResult` class to group together relevant data from matches.
- `Searcher.sorted_search` to obtain a list of sorted `SearchResult`s.
- `grascii.dictionary.common` module to contain `DictionaryException`s and utility functions.
- `Dictionary` class to work with grascii dictionaries.
- `config.get_default_config` to get the text of the default configuration file.
- `-V` and `--version` command line options.
- `InvalidGrascii` exception which is produced by a parser.
- `--no-sort` option for `grascii search`.
- `grascii.parser.get_grascii_regex_str()` to get a string that can be compiled into a regular expression that recognizes grascii strings.

7.2.2 Changed

- `Searcher.search` no longer sorts results.
- `grascii.dictionary.list`: `get_installed` and `get_built_ins` return a collection of installed dictionary names (prefixed with `:`).
- `grascii.dictionary.install.install_dict` renamed to `install_dictionary` and accepts more options.
- `grascii.dictionary.uninstall.uninstall_dict` renamed to `uninstall_dictionary` and accepts more options.
- `DICTIONARY_PATH` renamed to `INSTALLATION_DIR`.
- Using builtin `sorted` function speeds up general grascii searches.
- `GrasciiParser.interpret` returns an iterator instead of a list.
- Updated preanniversary dictionary to 2022.07.26.

7.2.3 Removed

- Dropped Python 3.6 support.
- `grascii.dictionary.get_dict_file`: Use `grascii.dictionary.Dictionary.open` instead.
- `GrasciiValidator.__init__` `use_cache` option

7.2.4 Fixed

- Typing issues with searchers and metrics.
- Errors when passing a grascii string with boundaries or disjoiners to the aggressive dephraser.

7.3 0.4.1 - 2022-06-29

7.3.1 Added

- Some classes and functions that are considered to be part of the public API are importable from the top-level `grascii`.

7.3.2 Fixed

- Included `TV` in `grammar.STROKES`.

7.4 0.4.0 - 2022-06-27

7.4.1 Added

- New `parser` module abstracts away Grascii parsing details.
- `grammar.CONSONANTS` and `grammar.VOWELS` constants.
- Experimental `outline` module with `Outline` class to infer directions of shorthand strokes.
- `GrasciiValidator` class to quickly validate, but not interpret, Grascii strings.
- `dictionaries` submodule to include dictionary source files.
- Dictionary build `--words` option to specify words file for spell checking.
- Dictionary build `--verbose` option to increase build output.
- New `docs` extra to specify doc building requirements.

7.4.2 Changed

- Switched from `lark-parser` to new `lark` package.
- Boundaries can be retained during Grascii Interpretation creation.
- `grascii.lark` is compatible with LALR.
- An apostrophe is accepted to represent “a” or “an”.
- Multiple semantic `grascii.lark` grammar changes (see @7ebfd07).
- Dictionary build `--parse` option is now much faster.
- `ReverseSearcher` provides a more useful sorting of results.
- Updated preanniversary dictionary to r00004.

7.4.3 Removed

- The `types` module has been removed. `Interpretation` is now defined in `grascii.parser`.
- The `utils` module has been removed.
- Dictionary source files are no longer stored in `dsrc`.
- The dictionary build `--spell` option has been removed. (Succeeded by `--words`)

7.4.4 Fixed

- Removed `Y` from `grammar.HARD_CHARACTERS` and `grammar.ALPHABET`.
- Included `DV` in `grammar.STROKES`.
- Grascii contain searches do not match translations.
- Grascii searches match -ing(s) at the end of words.
- Grascii searches match a disjoiner at the end of words.
- Grascii searches do not match double aspirates (except at the end of a word) or double disjoiners.
- Fixed crash on interrupt during interactive interpretation selection.

7.5 0.3.0 - 2021-12-14

7.5.1 Added

- New interactive search mode setting to select the dictionaries to search.

7.5.2 Changed

- The search `-d/--dictionary` option can be specified multiple times to search more than one dictionary at a time.
- The config file `[Search] Dictionary` option now accepts a list of dictionaries.

7.6 0.2.2 - 2021-07-08

7.6.1 Added

- Added the `-n/--count` option to `dictionary build` to enable the validation of expected word counts.

7.6.2 Changed

- Word count validation for dictionary builds is no longer performed by default, but enabled with the `--count` option–helpful for phrase dictionaries.
- When the dictionary builder cannot determine an appropriate output file for an entry, it now prints an error and continues instead of crashing the build process.

7.6.3 Fixed

- In dictionary builds, the incorrect number of words warning now properly behaves like a warning. The entry with the warning is now included in the build instead of being skipped.

7.7 0.2.1 - 2021-07-02

7.7.1 Added

- `grascii.grammar.ALPHABET`: The set of valid characters in the Grascii language.

7.7.2 Changed

- Grascii Search produces a better error message when given an invalid Grascii string.
- Grascii Dephrase produces a better error message when there are no results.

7.8 0.2.0 - 2021-06-25

First Release

7.8.1 Added

- Grascii Search with Grascii, Interactive, Reverse, and Regex modes
- Grascii Dictionary build and installation tools
- Grascii Configuration file and management
- Built-in pre-anniversary dictionary [Status: Review]
- Experimental Grascii Dephrase tool

CHAPTER

EIGHT

INDICES AND TABLES

- genindex
- modindex

INDEX