# PyCDSL Documentation

***Release 0.9.0***

**Hrishikesh Terdalkar**

**Apr 26, 2022**

# CONTENTS:

PyCDSL is a python interface to Cologne Digital Sanskrit Lexicon (CDSL).

- Free software: GNU General Public License v3
- Documentation: https://pycdsl.readthedocs.io.

CHAPTER

ONE

# FEATURES

- CDSL Corpus Management (Download, Update, Access)
- Unified Programmable Interface to access all dictionaries available at CDSL
- Command Line Interfaces for quick and easy search
  - Console Command: `cdsl`
  - REPL Interface (*powered by* `cmd2`)
- Extensive support for transliteration using `indic-transliteration` module
- Search by key, value or both

## 1.1 PyCDSL

PyCDSL is a python interface to Cologne Digital Sanskrit Lexicon (CDSL).

- Free software: GNU General Public License v3
- Documentation: https://pycdsl.readthedocs.io.

### 1.1.1 Features


- CDSL Corpus Management (Download, Update, Access)
- Unified Programmable Interface to access all dictionaries available at CDSL
- Command Line Interfaces for quick and easy search
  - Console Command: `cdsl`
  - REPL Interface (*powered by* `cmd2`)
- Extensive support for transliteration using `indic-transliteration` module
- Search by key, value or both

### 1.1.2 Install

To install PyCDSL, run this command in your terminal:

```
$ pip install PyCDSL
```

### 1.1.3 Usage

PyCDSL can be used in a python project, as a console command and as an interactive REPL interface.

#### Using PyCDSL in a Project

Import PyCDSL in a project:

```
import pycdsl
```

Create a CDSLCorpus Instance:

```
# Default installation at ~/cdsl_data
CDSL = pycdsl.CDSLCorpus()

# Custom installation path can be specified with argument `data_dir`
# e.g. CDSL = pycdsl.CDSLCorpus(data_dir="custom-installation-path")

# Custom transliteration schemes for input and output can be specified
# with arguments `input_scheme` and `output_scheme`.
# Values should be valid names of the schemes from `indic-transliteration`
# If unspecified, `DEFAULT_SCHEME` (`devanagari`) would be used.
# e.g. CDSL = pycdsl.CDSLCorpus(input_scheme="hk", output_scheme="iast")

# Search mode can be specified to search values by key or value or both.
# Valid options for `search_mode` are "key", "value", "both".
# These are also stored in convenience variables, and it is recommended
# to use these instead of string literals.
# The variables are, SEARCH_MODE_KEY, SEARCH_MODE_VALUE, SEARCH_MODE_BOTH.
# The variable SEARCH_MODES will always hold the list of all valid modes.
# The variable DEFAULT_SEARCH_MODE will alway point to the default mode.
# e.g. CDSL = pycdsl.CDSLCorpus(search_mode=pycdsl.SEARCH_MODE_VALUE)
```

Setup default dictionaries (`["MW", "MWE", "AP90", "AE"]`):

```
# Note: Any additional dictionaries that are installed will also be loaded.
CDSL.setup()

# For loading specific dictionaries only,
# a list of dictionary IDs can be passed to the setup function
# e.g. CDSL.setup(["VCP"])

# If `update` flag is True, update check is performed for every dictionary
# in `dict_ids` and if available, the updated version is installed
# e.g. CDSL.setup(["MW"], update=True)
```

Search in a dictionary:

```
# Any loaded dictionary is accessible using `[]` operator and dictionary ID
# e.g. CDSL["MW"]
results = CDSL["MW"].search("")

# Alternatively, they are also accessible like an attribute
# e.g. CDSL.MW, CDSL.MWE etc.
results = CDSL.MW.search("")

# Note: Attribute access and Item access both use the `dicts` property
# under the hood to access the dictionaries.
# >>> CDSL.MW is CDSL.dicts["MW"]
# True
# >>> CDSL["MW"] is CDSL.dicts["MW"]
# True

# `input_scheme` and `output_scheme` can be specified to the search function.
CDSL.MW.search("ka", input_scheme="iast", output_scheme="itrans")[0]
# <MWEntry: 55142: kRRiShNa = 1. kRRiShNa/ mf(A/)n. black, dark, dark-blue (opposed to␣
↪shveta/, shukla/, ro/hita, and aruNa/), RV.; AV. &c.>

# Search using wildcard (i.e. `*`)
# e.g. To search all etnries starting with kRRi (i.e. )
CDSL.MW.search("kRRi*", input_scheme="itrans")

# Limit and/or Offset the number of search results, e.g.
# Show the first 10 results
CDSL.MW.search("k*", input_scheme="iast", limit=10)
# Show the next 10 results
CDSL.MW.search("k*", input_scheme="iast", limit=10, offset=10)

# Search using a different search mode
CDSL.MW.search("", mode=pycdsl.SEARCH_MODE_VALUE)
```

Access an entry by ID:

```
# Access entry by `entry_id` using `[]` operator
entry = CDSL.MW["263938"]

# Alternatively, use `CDSLDict.entry` function
entry = CDSL.MW.entry("263938")

# Note: Access using `[]` operator calls the `CDSLDict.entry` function.
# The difference is that, in case an `entry_id` is absent,
# `[]` based access will raise a `KeyError`
# `CDSLDict.entry` will return None and log a `logging.ERROR` level message

# >>> entry
# <MWEntry: 263938:  = lord of the senses (said of Manas), BhP.>

# Output transliteration scheme can also be provided

CDSL.MW.entry("263938", output_scheme="iast")
# <MWEntry: 263938: hīkeśa = lord of the senses (said of Manas), BhP.>
```

`Entry` class also supports transliteration after creation. Thus, any entry fetched either through `search()` function or through `entry()` function can be transliterated.

Transliterate a single entry:

```
CDSL.MW.entry("263938").transliterate("slp1")
# <MWEntry: 263938: hfzIkeSa = lord of the senses (said of Manas), BhP.>
```

Change transliteration scheme for a dictionary:

```
CDSL.MW.set_scheme(input_scheme="itrans")
CDSL.MW.search("rAma")
```

Change search mode for a dictionary:

```
CDSL.MW.set_search_mode(mode="value")
CDSL.MW.search("hRRiShIkesha")
```

Classes `CDSLCorpus` and `CDSLDict` are iterable.

- Iterating over `CDSLCorpus` yields loaded dictionary instances.
- Iterating over `CDSLDict` yields entries in that dictionary.

```
# Iteration over a `CDSLCorpus` instance

for cdsl_dict in CDSL:
    print(type(cdsl_dict))
    print(cdsl_dict)
    break

# <class 'pycdsl.lexicon.CDSLDict'>
# CDSLDict(id='MW', date='1899', name='Monier-Williams Sanskrit-English Dictionary')

# Iteration over a `CDSLDict` isntance
for entry in CDSL.MW:
    print(type(entry))
    print(entry)
    break

# <class 'pycdsl.models.MWEntry'>
# <MWEntry: 1:  = 1.    the first letter of the alphabet>
```

**Note**: Please check the documentation of modules in the PyCDSL Package for more detailed information on available classes and functions.

https://pycdsl.readthedocs.io/en/latest/pycdsl.html

### Using Console Interface of PyCDSL

Help to the Console Interface:

```
usage: cdsl [-h] [-i] [-s SEARCH] [-p PATH] [-d DICTS [DICTS ...]]
            [-sm SEARCH_MODE] [-is INPUT_SCHEME] [-os OUTPUT_SCHEME]
            [-hf HISTORY_FILE] [-sc STARTUP_SCRIPT]
            [-u] [-dbg] [-v]

Access dictionaries from Cologne Digital Sanskrit Lexicon (CDSL)

optional arguments:
  -h, --help            show this help message and exit
  -i, --interactive     start in an interactive REPL mode
  -s SEARCH, --search SEARCH
                        search pattern (ignored if `--interactive` mode is set)
  -p PATH, --path PATH  path to installation
  -d DICTS [DICTS ...], --dicts DICTS [DICTS ...]
                        dictionary id(s)
  -sm SEARCH_MODE, --search-mode SEARCH_MODE
                        search mode
  -is INPUT_SCHEME, --input-scheme INPUT_SCHEME
                        input transliteration scheme
  -os OUTPUT_SCHEME, --output-scheme OUTPUT_SCHEME
                        output transliteration scheme
  -hf HISTORY_FILE, --history-file HISTORY_FILE
                        path to the history file
  -sc STARTUP_SCRIPT, --startup-script STARTUP_SCRIPT
                        path to the startup script
  -u, --update          update specified dictionaries
  -dbg, --debug         turn debug mode on
  -v, --version         show version and exit
```

Common Usage:

```
$ cdsl -d MW AP90 -s
```

**Note**: Arguments for specifying installation path, dictionary IDs, input and output transliteration schemes are valid for both interactive REPL shell and non-interactive console command.

### Using REPL Interface of PyCDSL

REPL Interface is powered by `cmd2`, and thus supports persistent history, start-up script, and several other rich features.

To use REPL Interface to Cologne Digital Sanskrit Lexicon (CDSL):

```
$ cdsl -i
```

### cmd2 Inherited REPL Features

- **Persistent History** across sessions is maintained at `~/.cdsl_history`.
- If **Start-up Script** is present (`~/.cdslrc`), the commands (one per line) are run at the start-up.
- Customized **shortcuts** for several useful commands, such as `!` for `shell`, `/` for `search` and `$` for `show`.
- **Aliases** can be created on runtime.
- **Output Redirection** works like the standard console, e.g. `command args > output.txt` will write the output of `command` to `output.txt`. Similarly, >> can be used to append the output.
- **Clipboard Integration** is supported through `Pyperclip`. If the output file name is omitted, the output is copied to the clipboard, e.g., `command args >`. The output can even be appended to clipboard by `command args >>`.

**References**

- `cmd2`: https://cmd2.readthedocs.io/en/latest/index.html
- `pyperclip`: https://pypi.org/project/pyperclip/

**Note**: The locations of history file and start-up script can be customized through CLI options.

### REPL Session Example

```
Cologne Sanskrit Digital Lexicon (CDSL)
---------------------------------------
Install or load dictionaries by typing `use [DICT_IDS..]` e.g. `use MW`.
Type any keyword to search in the selected dictionaries. (help or ? for list of options)
Loaded 4 dictionaries.

(CDSL::None) help -v

Documented commands (use 'help -v' for verbose/'help <topic>' for details):

Core
================================================================================================================
available             Display a list of dictionaries available in CDSL
dicts                 Display a list of dictionaries available locally
info                  Display information about active dictionaries
search                Search in the active dictionaries
show                  Show a specific entry by ID
stats                 Display statistics about active dictionaries
update                Update loaded dictionaries
use                   Load the specified dictionaries from CDSL.
                      If not available locally, they will be installed first.

Utility
================================================================================================================
alias                 Manage aliases
help                  List available commands or provide detailed help for a specific␣
↪command
history               View, run, edit, save, or clear previously entered commands
macro                 Manage macros
quit                  Exit this application
run_script            Run commands in script file that is encoded as either ASCII or UTF-
↪8 text
```

(continues on next page)

(continued from previous page)

```
set                   Set a settable parameter or show current settings of parameters
shell                 Execute a command as if at the OS prompt
shortcuts             List available shortcuts
version               Show the current version of PyCDSL

(CDSL::None) help available
Display a list of dictionaries available in CDSL

(CDSL::None) help search

Usage: search [-h] [--limit LIMIT] [--offset OFFSET] pattern

    Search in the active dictionaries

    Note
    ----
    * Searching in the active dictionaries is also the default action.
    * In general, we do not need to use this command explicitly unless we
      want to search the command keywords, such as, `available` `search`,
      `version`, `help` etc. in the active dictionaries.


positional arguments:
pattern          search pattern

optional arguments:
  -h, --help       show this help message and exit
  --limit LIMIT    limit results
  --offset OFFSET  skip results

(CDSL::None) help dicts
Display a list of dictionaries available locally

(CDSL::None) dicts
CDSLDict(id='AP90', date='1890', name='Apte Practical Sanskrit-English Dictionary')
CDSLDict(id='MW', date='1899', name='Monier-Williams Sanskrit-English Dictionary')
CDSLDict(id='MWE', date='1851', name='Monier-Williams English-Sanskrit Dictionary')
CDSLDict(id='AE', date='1920', name="Apte Student's English-Sanskrit Dictionary")

(CDSL::None) update
Data for dictionary 'AP90' is up-to-date.
Data for dictionary 'MW' is up-to-date.
Data for dictionary 'MWE' is up-to-date.
Data for dictionary 'AE' is up-to-date.

(CDSL::None) use MW
Using 1 dictionaries: ['MW']

(CDSL::MW)

Found 6 results in MW.
```

(continues on next page)

(continued from previous page)

```
<MWEntry: 263922:  = - a   See below under .>
<MWEntry: 263934:  =  b m. (perhaps = - cf. - above) id. (- n.), MBh.; Hariv. &c.>
<MWEntry: 263935:  = N. of the tenth month, VarBS.>
<MWEntry: 263936:  = of a Tīrtha, Cat.>
<MWEntry: 263937:  = of a poet, ib.>
<MWEntry: 263938:  = lord of the senses (said of Manas), BhP.>

(CDSL::MW) show 263938

<MWEntry: 263938:  = lord of the senses (said of Manas), BhP.>

(CDSL::MW) show 263938 --show-data

<MWEntry: 263938:  = lord of the senses (said of Manas), BhP.>

Data:
<H3A><h><key1>hfzIkeSa<\/key1><key2>hfzIkeSa<\/key2><\/h>
<body>  lord of the senses (said of <s1 slp1="manas">Manas<\/s1>), <ls>BhP.<\/ls><info␣
↪lex="inh"\/><\/body>
<tail><L>263938<\/L><pc>1303,2<\/pc><\/tail><\/H3A>

(CDSL::MW) $263938

<MWEntry: 263938:  = lord of the senses (said of Manas), BhP.>

(CDSL::MW) $263938 > output.txt
(CDSL::MW) !cat output.txt

<MWEntry: 263938:  = lord of the senses (said of Manas), BhP.>

(CDSL::MW) set input_scheme itrans
input_scheme - was: 'devanagari'
now: 'itrans'

(CDSL::MW) hRRiSIkesha

Found 6 results in MW.

<MWEntry: 263922:  = - a   See below under .>
<MWEntry: 263934:  =  b m. (perhaps = - cf. - above) id. (- n.), MBh.; Hariv. &c.>
<MWEntry: 263935:  = N. of the tenth month, VarBS.>
<MWEntry: 263936:  = of a Tīrtha, Cat.>
<MWEntry: 263937:  = of a poet, ib.>
<MWEntry: 263938:  = lord of the senses (said of Manas), BhP.>

(CDSL::MW) set output_scheme iast
output_scheme - was: 'devanagari'
now: 'iast'

(CDSL::MW) hRRiSIkesha

Found 6 results in MW.
```

(continues on next page)

(continued from previous page)

```
<MWEntry: 263922: hīkeśa = hī-keśa a   See below under hīka.>
<MWEntry: 263934: hīkeśa = hīkeśa b m. (perhaps = hī-keśa cf. hī-vat above) id. (-tva n.
↪), MBh.; Hariv. &c.>
<MWEntry: 263935: hīkeśa = N. of the tenth month, VarBS.>
<MWEntry: 263936: hīkeśa = of a Tīrtha, Cat.>
<MWEntry: 263937: hīkeśa = of a poet, ib.>
<MWEntry: 263938: hīkeśa = lord of the senses (said of Manas), BhP.>

(CDSL::MW) set limit 2
limit - was: 50
now: 2

(CDSL::MW) hRRiSIkesha

Found 2 results in MW.

<MWEntry: 263922: hīkeśa = hī-keśa a   See below under hīka.>
<MWEntry: 263934: hīkeśa = hīkeśa b m. (perhaps = hī-keśa cf. hī-vat above) id. (-tva n.
↪), MBh.; Hariv. &c.>

(CDSL::MW) set limit -1
limit - was: 2
now: None

(CDSL::MW) set search_mode value
search_mode - was: 'key'
now: 'value'

(CDSL::MW) hRRiSIkesha

Found 1 results in MW.

<MWEntry: 263938.1: hīkeśatva = hīkeśa-tva n.>

(CDSL::MW) set search_mode both
search_mode - was: 'value'
now: 'both'

(CDSL::MW) hRRiSIkesha

Found 7 results in MW.

<MWEntry: 263922: hīkeśa = hī-keśa a   See below under hīka.>
<MWEntry: 263934: hīkeśa = hīkeśa b m. (perhaps = hī-keśa cf. hī-vat above) id. (-tva n.
↪), MBh.; Hariv. &c.>
<MWEntry: 263935: hīkeśa = N. of the tenth month, VarBS.>
<MWEntry: 263936: hīkeśa = of a Tīrtha, Cat.>
<MWEntry: 263937: hīkeśa = of a poet, ib.>
<MWEntry: 263938: hīkeśa = lord of the senses (said of Manas), BhP.>
<MWEntry: 263938.1: hīkeśatva = hīkeśa-tva n.>
```

(continues on next page)

(continued from previous page)

```
(CDSL::MW) info
Total 1 dictionaries are active.
CDSLDict(id='MW', date='1899', name='Monier-Williams Sanskrit-English Dictionary')

(CDSL::MW) stats
Total 1 dictionaries are active.
---
CDSLDict(id='MW', date='1899', name='Monier-Williams Sanskrit-English Dictionary')
{'total': 287627, 'distinct': 194044, 'top': [('', 50), ('', 46), ('', 46), ('', 45), ('
↪', 39), ('', 39), ('', 39), ('', 38), ('', 36), ('', 36)]}

(CDSL::MW) use WIL

Downloading 'WIL.web.zip' ... (8394727 bytes)
100%|| 8.39M/8.39M [00:21<00:00, 386kB/s]
Successfully downloaded 'WIL.web.zip' from 'https://www.sanskrit-lexicon.uni-koeln.de/
↪scans/WILScan/2020/downloads/wilweb1.zip'.
Using 1 dictionaries: ['WIL']

(CDSL::WIL)

(CDSL::WIL) use WIL MW
Using 2 dictionaries: ['WIL', 'MW']

(CDSL::WIL,MW) hRRiSIkesha

Found 1 results in WIL.

<WILEntry: 44411: hīkeśa = hīkeśa  m. (-śa) KA or VIU. E. hīka an organ of sense, and␣
↪īśa lord.>

Found 6 results in MW.

<MWEntry: 263922: hīkeśa = hī-keśa a   See below under hīka.>
<MWEntry: 263934: hīkeśa = hīkeśa b m. (perhaps = hī-keśa cf. hī-vat above) id. (-tva n.
↪), MBh.; Hariv. &c.>
<MWEntry: 263935: hīkeśa = N. of the tenth month, VarBS.>
<MWEntry: 263936: hīkeśa = of a Tīrtha, Cat.>
<MWEntry: 263937: hīkeśa = of a poet, ib.>
<MWEntry: 263938: hīkeśa = lord of the senses (said of Manas), BhP.>

(CDSL::WIL,MW) use MW AP90 MWE AE
Using 4 dictionaries: ['MW', 'AP90', 'MWE', 'AE']

(CDSL::MW+3) use --all
Using 5 dictionaries: ['AP90', 'MW', 'MWE', 'AE', 'WIL']

(CDSL::AP90+3) use --none
Using 0 dictionaries: []

(CDSL::None) quit
```

### 1.1.4 Credits

This application uses data from Cologne Digital Sanskrit Dictionaries, Cologne University.

## 1.2 Installation

### 1.2.1 Stable release

To install PyCDSL, run this command in your terminal:

```
$ pip install PyCDSL
```

This is the preferred method to install PyCDSL, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

### 1.2.2 From sources

The sources for PyCDSL can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/hrishikeshrt/PyCDSL
```

Or download the tarball:

```
$ curl -OJL https://github.com/hrishikeshrt/PyCDSL/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

## 1.3 Usage

PyCDSL can be used in a python project, as a console command and as an interactive REPL interface.

### 1.3.1 Using PyCDSL in a Project

Import PyCDSL in a project:

```
import pycdsl
```

Create a CDSLCorpus Instance:

```
# Default installation at ~/cdsl_data
CDSL = pycdsl.CDSLCorpus()

# Custom installation path can be specified with argument `data_dir`
# e.g. CDSL = pycdsl.CDSLCorpus(data_dir="custom-installation-path")
```

(continues on next page)

(continued from previous page)

```
# Custom transliteration schemes for input and output can be specified
# with arguments `input_scheme` and `output_scheme`.
# Values should be valid names of the schemes from `indic-transliteration`
# If unspecified, `DEFAULT_SCHEME` (`devanagari`) would be used.
# e.g. CDSL = pycdsl.CDSLCorpus(input_scheme="hk", output_scheme="iast")

# Search mode can be specified to search values by key or value or both.
# Valid options for `search_mode` are "key", "value", "both".
# These are also stored in convenience variables, and it is recommended
# to use these instead of string literals.
# The variables are, SEARCH_MODE_KEY, SEARCH_MODE_VALUE, SEARCH_MODE_BOTH.
# The variable SEARCH_MODES will always hold the list of all valid modes.
# The variable DEFAULT_SEARCH_MODE will alway point to the default mode.
# e.g. CDSL = pycdsl.CDSLCorpus(search_mode=pycdsl.SEARCH_MODE_VALUE)
```

Setup default dictionaries (`["MW", "MWE", "AP90", "AE"]`):

```
# Note: Any additional dictionaries that are installed will also be loaded.
CDSL.setup()

# For loading specific dictionaries only,
# a list of dictionary IDs can be passed to the setup function
# e.g. CDSL.setup(["VCP"])

# If `update` flag is True, update check is performed for every dictionary
# in `dict_ids` and if available, the updated version is installed
# e.g. CDSL.setup(["MW"], update=True)
```

Search in a dictionary:

```
# Any loaded dictionary is accessible using `[]` operator and dictionary ID
# e.g. CDSL["MW"]
results = CDSL["MW"].search("")

# Alternatively, they are also accessible like an attribute
# e.g. CDSL.MW, CDSL.MWE etc.
results = CDSL.MW.search("")

# Note: Attribute access and Item access both use the `dicts` property
# under the hood to access the dictionaries.
# >>> CDSL.MW is CDSL.dicts["MW"]
# True
# >>> CDSL["MW"] is CDSL.dicts["MW"]
# True

# `input_scheme` and `output_scheme` can be specified to the search function.
CDSL.MW.search("ka", input_scheme="iast", output_scheme="itrans")[0]
# <MWEntry: 55142: kRRiShNa = 1. kRRiShNa/ mf(A/)n. black, dark, dark-blue (opposed to␣
↪shveta/, shukla/, ro/hita, and aruNa/), RV.; AV. &c.>

# Search using wildcard (i.e. `*`)
```

(continues on next page)

(continued from previous page)

```
# e.g. To search all etnries starting with kRRi (i.e. )
CDSL.MW.search("kRRi*", input_scheme="itrans")

# Limit and/or Offset the number of search results, e.g.
# Show the first 10 results
CDSL.MW.search("k*", input_scheme="iast", limit=10)
# Show the next 10 results
CDSL.MW.search("k*", input_scheme="iast", limit=10, offset=10)

# Search using a different search mode
CDSL.MW.search("", mode=pycdsl.SEARCH_MODE_VALUE)
```

Access an entry by ID:

```
# Access entry by `entry_id` using `[]` operator
entry = CDSL.MW["263938"]

# Alternatively, use `CDSLDict.entry` function
entry = CDSL.MW.entry("263938")

# Note: Access using `[]` operator calls the `CDSLDict.entry` function.
# The difference is that, in case an `entry_id` is absent,
# `[]` based access will raise a `KeyError`
# `CDSLDict.entry` will return None and log a `logging.ERROR` level message

# >>> entry
# <MWEntry: 263938:  = lord of the senses (said of Manas), BhP.>

# Output transliteration scheme can also be provided

CDSL.MW.entry("263938", output_scheme="iast")
# <MWEntry: 263938: hīkeśa = lord of the senses (said of Manas), BhP.>
```

`Entry` class also supports transliteration after creation. Thus, any entry fetched either through `search()` function or through `entry()` function can be transliterated.

Transliterate a single entry:

```
CDSL.MW.entry("263938").transliterate("slp1")
# <MWEntry: 263938: hfzIkeSa = lord of the senses (said of Manas), BhP.>
```

Change transliteration scheme for a dictionary:

```
CDSL.MW.set_scheme(input_scheme="itrans")
CDSL.MW.search("rAma")
```

Change search mode for a dictionary:

```
CDSL.MW.set_search_mode(mode="value")
CDSL.MW.search("hRRiShIkesha")
```

Classes `CDSLCorpus` and `CDSLDict` are iterable.

- Iterating over `CDSLCorpus` yields loaded dictionary instances.

- Iterating over `CDSLDict` yields entries in that dictionary.

```
# Iteration over a `CDSLCorpus` instance

for cdsl_dict in CDSL:
    print(type(cdsl_dict))
    print(cdsl_dict)
    break

# <class 'pycdsl.lexicon.CDSLDict'>
# CDSLDict(id='MW', date='1899', name='Monier-Williams Sanskrit-English Dictionary')

# Iteration over a `CDSLDict` isntance
for entry in CDSL.MW:
    print(type(entry))
    print(entry)
    break

# <class 'pycdsl.models.MWEntry'>
# <MWEntry: 1:  = 1.    the first letter of the alphabet>
```

**Note**: Please check the documentation of modules in the PyCDSL Package for more detailed information on available classes and functions.

https://pycdsl.readthedocs.io/en/latest/pycdsl.html

### 1.3.2 Using Console Interface of PyCDSL

Help to the Console Interface:

```
usage: cdsl [-h] [-i] [-s SEARCH] [-p PATH] [-d DICTS [DICTS ...]]
            [-sm SEARCH_MODE] [-is INPUT_SCHEME] [-os OUTPUT_SCHEME]
            [-hf HISTORY_FILE] [-sc STARTUP_SCRIPT]
            [-u] [-dbg] [-v]

Access dictionaries from Cologne Digital Sanskrit Lexicon (CDSL)

optional arguments:
  -h, --help            show this help message and exit
  -i, --interactive     start in an interactive REPL mode
  -s SEARCH, --search SEARCH
                        search pattern (ignored if `--interactive` mode is set)
  -p PATH, --path PATH  path to installation
  -d DICTS [DICTS ...], --dicts DICTS [DICTS ...]
                        dictionary id(s)
  -sm SEARCH_MODE, --search-mode SEARCH_MODE
                        search mode
  -is INPUT_SCHEME, --input-scheme INPUT_SCHEME
                        input transliteration scheme
  -os OUTPUT_SCHEME, --output-scheme OUTPUT_SCHEME
                        output transliteration scheme
  -hf HISTORY_FILE, --history-file HISTORY_FILE
                        path to the history file
```

(continues on next page)

(continued from previous page)

```
  -sc STARTUP_SCRIPT, --startup-script STARTUP_SCRIPT
                        path to the startup script
  -u, --update          update specified dictionaries
  -dbg, --debug         turn debug mode on
  -v, --version         show version and exit
```

Common Usage:

```
$ cdsl -d MW AP90 -s
```

**Note**: Arguments for specifying installation path, dictionary IDs, input and output transliteration schemes are valid for both interactive REPL shell and non-interactive console command.

### 1.3.3 Using REPL Interface of PyCDSL

REPL Interface is powered by `cmd2`, and thus supports persistent history, start-up script, and several other rich features.

To use REPL Interface to Cologne Digital Sanskrit Lexicon (CDSL):

```
$ cdsl -i
```

### 1.3.4 cmd2 Inherited REPL Features

- **Persistent History** across sessions is maintained at `~/.cdsl_history`.
- If **Start-up Script** is present (`~/.cdslrc`), the commands (one per line) are run at the start-up.
- Customized **shortcuts** for several useful commands, such as `!` for `shell`, `/` for `search` and `$` for `show`.
- **Aliases** can be created on runtime.
- **Output Redirection** works like the standard console, e.g. `command args > output.txt` will write the output of `command` to `output.txt`. Similarly, >> can be used to append the output.
- **Clipboard Integration** is supported through `Pyperclip`. If the output file name is omitted, the output is copied to the clipboard, e.g., `command args >`. The output can even be appended to clipboard by `command args >>`.

**References**

- `cmd2`: https://cmd2.readthedocs.io/en/latest/index.html
- `pyperclip`: https://pypi.org/project/pyperclip/

**Note**: The locations of history file and start-up script can be customized through CLI options.

### 1.3.5 REPL Session Example

```
Cologne Sanskrit Digital Lexicon (CDSL)
---------------------------------------
Install or load dictionaries by typing `use [DICT_IDS..]` e.g. `use MW`.
Type any keyword to search in the selected dictionaries. (help or ? for list of options)
Loaded 4 dictionaries.

(CDSL::None) help -v
```

(continues on next page)

(continued from previous page)

```
Documented commands (use 'help -v' for verbose/'help <topic>' for details):

Core
================================================================================================================
available             Display a list of dictionaries available in CDSL
dicts                 Display a list of dictionaries available locally
info                  Display information about active dictionaries
search                Search in the active dictionaries
show                  Show a specific entry by ID
stats                 Display statistics about active dictionaries
update                Update loaded dictionaries
use                   Load the specified dictionaries from CDSL.
                      If not available locally, they will be installed first.

Utility
================================================================================================================
alias                 Manage aliases
help                  List available commands or provide detailed help for a specific␣
↪command
history               View, run, edit, save, or clear previously entered commands
macro                 Manage macros
quit                  Exit this application
run_script            Run commands in script file that is encoded as either ASCII or UTF-
↪8 text
set                   Set a settable parameter or show current settings of parameters
shell                 Execute a command as if at the OS prompt
shortcuts             List available shortcuts
version               Show the current version of PyCDSL

(CDSL::None) help available
Display a list of dictionaries available in CDSL

(CDSL::None) help search

Usage: search [-h] [--limit LIMIT] [--offset OFFSET] pattern

    Search in the active dictionaries

    Note
    ----
    * Searching in the active dictionaries is also the default action.
    * In general, we do not need to use this command explicitly unless we
      want to search the command keywords, such as, `available` `search`,
      `version`, `help` etc. in the active dictionaries.


positional arguments:
pattern          search pattern

optional arguments:
  -h, --help       show this help message and exit
```

(continues on next page)

(continued from previous page)

```
  --limit LIMIT    limit results
  --offset OFFSET  skip results

(CDSL::None) help dicts
Display a list of dictionaries available locally

(CDSL::None) dicts
CDSLDict(id='AP90', date='1890', name='Apte Practical Sanskrit-English Dictionary')
CDSLDict(id='MW', date='1899', name='Monier-Williams Sanskrit-English Dictionary')
CDSLDict(id='MWE', date='1851', name='Monier-Williams English-Sanskrit Dictionary')
CDSLDict(id='AE', date='1920', name="Apte Student's English-Sanskrit Dictionary")

(CDSL::None) update
Data for dictionary 'AP90' is up-to-date.
Data for dictionary 'MW' is up-to-date.
Data for dictionary 'MWE' is up-to-date.
Data for dictionary 'AE' is up-to-date.

(CDSL::None) use MW
Using 1 dictionaries: ['MW']

(CDSL::MW)

Found 6 results in MW.

<MWEntry: 263922:  = - a   See below under .>
<MWEntry: 263934:  =  b m. (perhaps = - cf. - above) id. (- n.), MBh.; Hariv. &c.>
<MWEntry: 263935:  = N. of the tenth month, VarBS.>
<MWEntry: 263936:  = of a Tīrtha, Cat.>
<MWEntry: 263937:  = of a poet, ib.>
<MWEntry: 263938:  = lord of the senses (said of Manas), BhP.>

(CDSL::MW) show 263938

<MWEntry: 263938:  = lord of the senses (said of Manas), BhP.>

(CDSL::MW) show 263938 --show-data

<MWEntry: 263938:  = lord of the senses (said of Manas), BhP.>

Data:
<H3A><h><key1>hfzIkeSa<\/key1><key2>hfzIkeSa<\/key2><\/h>
<body>  lord of the senses (said of <s1 slp1="manas">Manas<\/s1>), <ls>BhP.<\/ls><info␣
↪lex="inh"\/><\/body>
<tail><L>263938<\/L><pc>1303,2<\/pc><\/tail><\/H3A>

(CDSL::MW) $263938

<MWEntry: 263938:  = lord of the senses (said of Manas), BhP.>

(CDSL::MW) $263938 > output.txt
(CDSL::MW) !cat output.txt
```

(continues on next page)

(continued from previous page)

```
<MWEntry: 263938:  = lord of the senses (said of Manas), BhP.>

(CDSL::MW) set input_scheme itrans
input_scheme - was: 'devanagari'
now: 'itrans'

(CDSL::MW) hRRiSIkesha

Found 6 results in MW.

<MWEntry: 263922:  = - a   See below under .>
<MWEntry: 263934:  =  b m. (perhaps = - cf. - above) id. (- n.), MBh.; Hariv. &c.>
<MWEntry: 263935:  = N. of the tenth month, VarBS.>
<MWEntry: 263936:  = of a Tīrtha, Cat.>
<MWEntry: 263937:  = of a poet, ib.>
<MWEntry: 263938:  = lord of the senses (said of Manas), BhP.>

(CDSL::MW) set output_scheme iast
output_scheme - was: 'devanagari'
now: 'iast'

(CDSL::MW) hRRiSIkesha

Found 6 results in MW.

<MWEntry: 263922: hīkeśa = hī-keśa a   See below under hīka.>
<MWEntry: 263934: hīkeśa = hīkeśa b m. (perhaps = hī-keśa cf. hī-vat above) id. (-tva n.
↪), MBh.; Hariv. &c.>
<MWEntry: 263935: hīkeśa = N. of the tenth month, VarBS.>
<MWEntry: 263936: hīkeśa = of a Tīrtha, Cat.>
<MWEntry: 263937: hīkeśa = of a poet, ib.>
<MWEntry: 263938: hīkeśa = lord of the senses (said of Manas), BhP.>

(CDSL::MW) set limit 2
limit - was: 50
now: 2

(CDSL::MW) hRRiSIkesha

Found 2 results in MW.

<MWEntry: 263922: hīkeśa = hī-keśa a   See below under hīka.>
<MWEntry: 263934: hīkeśa = hīkeśa b m. (perhaps = hī-keśa cf. hī-vat above) id. (-tva n.
↪), MBh.; Hariv. &c.>

(CDSL::MW) set limit -1
limit - was: 2
now: None

(CDSL::MW) set search_mode value
search_mode - was: 'key'
```

(continues on next page)

(continued from previous page)

```
now: 'value'

(CDSL::MW) hRRiSIkesha

Found 1 results in MW.

<MWEntry: 263938.1: hīkeśatva = hīkeśa-tva n.>

(CDSL::MW) set search_mode both
search_mode - was: 'value'
now: 'both'

(CDSL::MW) hRRiSIkesha

Found 7 results in MW.

<MWEntry: 263922: hīkeśa = hī-keśa a   See below under hīka.>
<MWEntry: 263934: hīkeśa = hīkeśa b m. (perhaps = hī-keśa cf. hī-vat above) id. (-tva n.
↪), MBh.; Hariv. &c.>
<MWEntry: 263935: hīkeśa = N. of the tenth month, VarBS.>
<MWEntry: 263936: hīkeśa = of a Tīrtha, Cat.>
<MWEntry: 263937: hīkeśa = of a poet, ib.>
<MWEntry: 263938: hīkeśa = lord of the senses (said of Manas), BhP.>
<MWEntry: 263938.1: hīkeśatva = hīkeśa-tva n.>

(CDSL::MW) info
Total 1 dictionaries are active.
CDSLDict(id='MW', date='1899', name='Monier-Williams Sanskrit-English Dictionary')

(CDSL::MW) stats
Total 1 dictionaries are active.
---
CDSLDict(id='MW', date='1899', name='Monier-Williams Sanskrit-English Dictionary')
{'total': 287627, 'distinct': 194044, 'top': [('', 50), ('', 46), ('', 46), ('', 45), ('
↪', 39), ('', 39), ('', 39), ('', 38), ('', 36), ('', 36)]}

(CDSL::MW) use WIL

Downloading 'WIL.web.zip' ... (8394727 bytes)
100%|| 8.39M/8.39M [00:21<00:00, 386kB/s]
Successfully downloaded 'WIL.web.zip' from 'https://www.sanskrit-lexicon.uni-koeln.de/
↪scans/WILScan/2020/downloads/wilweb1.zip'.
Using 1 dictionaries: ['WIL']

(CDSL::WIL)

(CDSL::WIL) use WIL MW
Using 2 dictionaries: ['WIL', 'MW']

(CDSL::WIL,MW) hRRiSIkesha

Found 1 results in WIL.
```

(continues on next page)

(continued from previous page)

```
<WILEntry: 44411: hīkeśa = hīkeśa  m. (-śa) KA or VIU. E. hīka an organ of sense, and␣
↪īśa lord.>

Found 6 results in MW.

<MWEntry: 263922: hīkeśa = hī-keśa a   See below under hīka.>
<MWEntry: 263934: hīkeśa = hīkeśa b m. (perhaps = hī-keśa cf. hī-vat above) id. (-tva n.
↪), MBh.; Hariv. &c.>
<MWEntry: 263935: hīkeśa = N. of the tenth month, VarBS.>
<MWEntry: 263936: hīkeśa = of a Tīrtha, Cat.>
<MWEntry: 263937: hīkeśa = of a poet, ib.>
<MWEntry: 263938: hīkeśa = lord of the senses (said of Manas), BhP.>

(CDSL::WIL,MW) use MW AP90 MWE AE
Using 4 dictionaries: ['MW', 'AP90', 'MWE', 'AE']

(CDSL::MW+3) use --all
Using 5 dictionaries: ['AP90', 'MW', 'MWE', 'AE', 'WIL']

(CDSL::AP90+3) use --none
Using 0 dictionaries: []

(CDSL::None) quit
```

## 1.4 pycdsl

### 1.4.1 pycdsl package

#### Submodules

#### pycdsl.cli module

Console script for PyCDSL

pycdsl.cli.**main**()

Command Line Interface for PyCDSL

#### pycdsl.constants module

PyCDSL Constants

### pycdsl.corpus module

CDSL Corpus Management

**class** pycdsl.corpus.**CDSLCorpus**(*data_dir: Optional[str] = None*, *search_mode: str = 'key'*, *input_scheme: str = 'devanagari'*, *output_scheme: str = 'devanagari'*, *transliterate_keys: bool = True*)

Bases: `object`

CDSL Corpus Class

Refers to a CDSL installation instance at the location *data_dir*.

**data_dir: str = None**

**search_mode: str = 'key'**

**input_scheme: str = 'devanagari'**

**output_scheme: str = 'devanagari'**

**transliterate_keys: bool = True**

**setup**(*dict_ids: Optional[list] = None*, *update: bool = False*, *model_map: Optional[Dict[str, Tuple[*pycdsl.models.Lexicon, pycdsl.models.Entry*]]] = None*) → bool

Setup CDSL dictionaries in bulk

Calls *CDSLDict.setup()* on every *CDSLDict*, and if successful, also calls *CDSLDict.connect()* to establish a connection to the database

**Parameters**

- **dict_ids** (*list or None, optional*) – List of dictionary IDs to setup. If None, the dictionaries from *DEFAULT_DICTIONARIES* as well as locally installed dictionaries will be setup. The default is None.
- **update** (*bool, optional*) – If True, and update check is performed for every dictionary in *dict_ids*, and if available, the updated version is installed The default is False.
- **lexicon_model** (*object, optional*) – Lexicon model argument passed to *CDSLDict.connect()* The default is None.
- **entry_model** (*object, optional*) – Entry model argument passed to *CDSLDict.connect()* The default is None.
- **model_map** (*dict, optional*) – Map of dictionary ID to a tuple of lexicon model and entry model. The argument is used to specify *lexicon_model* and *entry_model* arguments passed to *CDSLDict.connect()*. If None, the default map *DEFAULT_MODEL_MAP* will be used. The default is None.

**Returns** True, if the setup of all the dictionaries from *dict_ids* is successful. i.e. If every *CDSLDict.setup()* call returns True.

**Return type** bool

**Raises** **ValueError** – If *dict_ids* is not a *list* or *None*.

**search**(*pattern: str*, *dict_ids: Optional[List[str]] = None*, *mode: Optional[str] = None*, *input_scheme: Optional[str] = None*, *output_scheme: Optional[str] = None*, *ignore_case: bool = False*, *limit: Optional[int] = None*, *offset: Optional[int] = None*, *omit_empty: bool = True*) → Dict[str, List[*pycdsl.models.Entry*]]

Search in multiple dictionaries from the corpus

**Parameters**

- **pattern** (*str*) – Search pattern, may contain wildcards (*).
- **dict_ids** (*list or None*) – List of dictionary IDs to search in. Only the *dict_ids* that exist in *self.dicts* will be used. If None, all the dictionaries that have been setup, i.e., the dictionaries from *self.dicts* will be used. The default is None.
- **mode** (*str or None, optional*) – Search mode to query by *key*, *value* or *both*. The default is None.
- **input_scheme** (*str or None, optional*) – Input transliteration scheme If None, *self.input_scheme* will be used. The default is None.
- **output_scheme** (*str or None, optional*) – Output transliteration scheme If None, *self.output_scheme* will be used. The default is None.
- **ignore_case** (*bool, optional*) – Ignore case while performing lookup. The default is False.
- **limit** (*int or None, optional*) – Limit the number of search results to *limit*. The default is None.
- **offset** (*int or None, optional*) – Offset the search results by *offset*. The default is None
- **omit_empty** (*bool, optional*) – If True, only the non-empty search results will be included. The default is False.

**Returns** Dictionary of (dict_id, list of matching entries)

**Return type** dict

**get_available_dicts**() → Dict[str, *pycdsl.lexicon.CDSLDict*]

Fetch a list of dictionaries available for download from CDSL

Homepage of CDSL Project (*SERVER_URL*) is fetched and parsed to obtain this list.

**get_installed_dicts**() → Dict[str, *pycdsl.lexicon.CDSLDict*]

Fetch a list of dictionaries installed locally

## pycdsl.lexicon module

CDSL Lexicon Management

**class** pycdsl.lexicon.**CDSLDict**(*id: str*, *date: str*, *name: str*, *url: str*, *db: Optional[str] = None*, *search_mode: Optional[str] = None*, *input_scheme: Optional[str] = None*, *output_scheme: Optional[str] = None*, *transliterate_keys: Optional[bool] = None*)

Bases: object

Dictionary from CDSL

**id: str**

**date: str**

**name: str**

**url: str**

**db: str = None**

**search_mode: str = None**

**input_scheme: str = None**

**output_scheme: str = None**

**transliterate_keys: bool = None**

**download**(*download_dir: str*) → bool

Download and extract dictionary data

**Parameters** **download_dir** (*str or Path*) – Full path of directory where the dictionary data should be downloaded and extracted

**Returns** True if successfully downloaded or already up-to-date

**Return type** bool

**setup**(*data_dir: str*, *symlink_dir: Optional[str] = None*, *update: bool = False*) → bool

Setup the dictionary database path

**Parameters**

- **data_dir** (*str or Path*) – Full path of directory where the dictionary data is stored
- **symlink_dir** (*str or Path, optional*) – Full path of the directory where the symbolink links to the SQLite database of dictionary will be created If None, symbolic links aren't created. The default is None.
- **update** (*bool, optional*) – If True, an attempt to update dictionary data will be made. The default is False.

**Returns** True if the setup was successful

**Return type** bool

**set_scheme**(*input_scheme: Optional[str] = None*, *output_scheme: Optional[str] = None*, *transliterate_keys: Optional[bool] = None*)

Set transliteration scheme for the dictionary instance

**Parameters**

- **input_scheme** (*str, optional*) – Input transliteration scheme. If None, *INTERNAL_SCHEME* is used. The default is None.
- **output_scheme** (*str, optional*) – Output transliteration scheme. If None, *INTERNAL_SCHEME* is used. The default is None.
- **transliterate_keys** (*bool, optional*) – Determines whether the keys in lexicon should be transliterated to *scheme* or not. If None, the value will be inferred based on dictionary type. The default is None.

**set_search_mode**(*mode: str*)

Set search mode

**Parameters** **mode** (*str*) – Valid values are 'key', 'value', 'both' Recommended to use the convenience variables SEARCH_MODE_KEY, SEARCH_MODE_VALUE or SEARCH_MODE_BOTH.

**connect**(*lexicon_model: Optional[pycdsl.models.Lexicon] = None*, *entry_model: Optional[pycdsl.models.Entry] = None*)

Connect to the SQLite database

If both *lexicon_model* and *entry_model* are specified, they are used as the ORM layer, and take preference over *model_map*.

If any of *lexicon_model* or *entry_model* is None, then the models are resolved in the following way.

First, if the current dictionary ID is present in *model_map* the models specified by the *model_map* are used. Otherwise, *models.lexicon_constructor* and *models.entry_constructor* functions are used, which subclass the *models.Lexicon* and *models.Entry* models.

**Parameters**

- **lexicon_model** (*object, optional*) – Lexicon model. The default is None.
- **entry_model** (*object, optional*) – Entry model. The default is None.

**stats**(*top: int = 10*, *output_scheme: str = None*) → Dict

Display statistics about the lexicon

**Parameters**

- **top** (*int, optional*) – Display top *top* entries having most different meanings. The default is 10.
- **output_scheme** (*str, optional*) – Output transliteration scheme If None, *self.output_scheme* will be used. The default is None.

**Returns** Statistics about the dictionary

**Return type** dict

**search**(*pattern: str*, *mode: str = None*, *input_scheme: str = None*, *output_scheme: str = None*, *ignore_case: str = False*, *limit: int = None*, *offset: int = None*) → List[*pycdsl.models.Entry*]

Search in the dictionary

**Parameters**

- **pattern** (*str*) – Search pattern, may contain wildcards (*).
- **mode** (*str or None, optional*) – Search mode to query by *key*, *value* or *both*. If None, *self.search_mode* will be used. The default is None.
- **input_scheme** (*str or None, optional*) – Input transliteration scheme If None, *self.input_scheme* will be used. The default is None.
- **output_scheme** (*str or None, optional*) – Output transliteration scheme If None, *self.output_scheme* will be used. The default is None.
- **ignore_case** (*bool, optional*) – Ignore case while performing lookup. The default is False.
- **limit** (*int or None, optional*) – Limit the number of search results to *limit*. The default is None.
- **offset** (*int or None, optional*) – Offset the search results by *offset*. The default is None

**Returns** List of matching entries

**Return type** list

**entry**(*entry_id: str*, *output_scheme: Optional[str] = None*) → *pycdsl.models.Entry*

Get an entry by ID

**Parameters**

- **entry_id** (*str*) – Entry ID to lookup
- **output_scheme** (*str or None, optional*) – Output transliteration scheme If None, *self.output_scheme* will be used. The default is None.

**Returns** If the *entry_id* is valid, *Entry* with the matching ID otherwise, None.

**Return type** object

**dump**(*output_path: Optional[str] = None*, *output_scheme: Optional[str] = None*) → List[Dict[str, str]]

Dump data as JSON

**Parameters**

- **output_path** (*str or Path, optional*) – Path to the output JSON file. If None, the data isn't written to the disk, only returned. The default is None.
- **output_scheme** (*str or None, optional*) – Output transliteration scheme If None, *self.output_scheme* will be used. The default is None

**Returns** List of all the entries in the dictionary. Every entry is a *dict*. If *output_path* is provided, the same list is written as JSON.

**Return type** list

## pycdsl.models module

Models for Lexicon Access

**class** pycdsl.models.**Lexicon**(**args*, ***kwargs*)

Bases: peewee.Model

Lexicon Model

**id = <DecimalField: Lexicon.id>**

**key = <CharField: Lexicon.key>**

**data = <TextField: Lexicon.data>**

**DoesNotExist**

alias of pycdsl.models.LexiconDoesNotExist

**class** pycdsl.models.**Entry**(*lexicon_entry:* pycdsl.models.Lexicon, *lexicon_id: Optional[str] = None*, *scheme: Optional[str] = None*, *transliterate_keys: bool = True*)

Bases: object

Lexicon Entry

Wraps instances of Lexicon model which respresent query results

Lexicon Entry

**Parameters**

- **lexicon_entry** (Lexicon) – Instance of Lexicon model
- **lexicon_id** (*str, optional*) – ID of the Lexicon to which the entry belongs

- **scheme** (*str, optional*) – Output transliteration scheme. If valid, parts of the *data* in lexicon which are enclosed in <*s*> tags will be transliterated to *scheme*. If invalid or None, no transliteration will take place. The default is None.
- **transliterate_keys** (*bool, optional*) – If True, the keys in lexicon will be transliterated to *scheme*. The default is True.

**__init__**(*lexicon_entry: pycdsl.models.Lexicon*, *lexicon_id: Optional[str] = None*, *scheme: Optional[str] = None*, *transliterate_keys: bool = True*)

Lexicon Entry

**Parameters**

- **lexicon_entry** (Lexicon) – Instance of Lexicon model
- **lexicon_id** (*str, optional*) – ID of the Lexicon to which the entry belongs
- **scheme** (*str, optional*) – Output transliteration scheme. If valid, parts of the *data* in lexicon which are enclosed in <*s*> tags will be transliterated to *scheme*. If invalid or None, no transliteration will take place. The default is None.
- **transliterate_keys** (*bool, optional*) – If True, the keys in lexicon will be transliterated to *scheme*. The default is True.

**__post_init__**()

Placeholder to implement a custom post-init hook

**transliterate**(*scheme: str = 'devanagari'*, *transliterate_keys: bool = True*)

Transliterate Data

Part of the *data* in lexicon that is enclosed in <*s*> tags will be transliterated to *scheme*.

**Parameters**

- **scheme** (*str, optional*) – Output transliteration scheme. If invalid or None, no transliteration will take place. The default is *DEFAULT_SCHEME*.
- **transliterate_keys** (*bool, optional*) – If True, the keys in lexicon will be transliterated to *scheme*. The default is True.

**Returns** Returns a new transliterated instance

**Return type** object

**meaning**() → str

Extract meaning of the entry

**to_dict**() → Dict[str, str]

Get a python *dict* representation of the entry

**parse**()

pycdsl.models.**lexicon_constructor**(*dict_id: str*, *table_name: Optional[str] = None*) → *pycdsl.models.Lexicon*

Construct a Lexicon Model

**Parameters**

- **dict_id** (*str*) – Dictionary ID
- **table_name** (*str, optional*) – Name of the table in SQLite database. If None, it will be inferred as dict_id.lower() The default is None.

**Returns** Constructed class (a subclass of *Lexicon*) for a dictionary

**Return type** object

pycdsl.models.**entry_constructor**(*dict_id: str*) → *pycdsl.models.Entry*

Construct an Entry Model

**Parameters** **dict_id** (*str*) – Dictionary ID

**Returns** Constructed class (a subclass of *Entry*) for a dictionary entry

**Return type** object

**class** pycdsl.models.**AP90Lexicon**(**args*, ***kwargs*)

Bases: *pycdsl.models.Lexicon*

**DoesNotExist**

alias of pycdsl.models.AP90LexiconDoesNotExist

**data = <TextField: AP90Lexicon.data>**

**id = <DecimalField: AP90Lexicon.id>**

**key = <CharField: AP90Lexicon.key>**

**class** pycdsl.models.**AP90Entry**(*lexicon_entry: pycdsl.models.Lexicon*, *lexicon_id: Optional[str] = None*, *scheme: Optional[str] = None*, *transliterate_keys: bool = True*)

Bases: *pycdsl.models.Entry*

Lexicon Entry

**Parameters**

- **lexicon_entry** (Lexicon) – Instance of Lexicon model
- **lexicon_id** (*str, optional*) – ID of the Lexicon to which the entry belongs
- **scheme** (*str, optional*) – Output transliteration scheme. If valid, parts of the *data* in lexicon which are enclosed in <*s*> tags will be transliterated to *scheme*. If invalid or None, no transliteration will take place. The default is None.
- **transliterate_keys** (*bool, optional*) – If True, the keys in lexicon will be transliterated to *scheme*. The default is True.

**class** pycdsl.models.**MWLexicon**(**args*, ***kwargs*)

Bases: *pycdsl.models.Lexicon*

**DoesNotExist**

alias of pycdsl.models.MWLexiconDoesNotExist

**data = <TextField: MWLexicon.data>**

**id = <DecimalField: MWLexicon.id>**

**key = <CharField: MWLexicon.key>**

**class** pycdsl.models.**MWEntry**(*lexicon_entry: pycdsl.models.Lexicon*, *lexicon_id: Optional[str] = None*, *scheme: Optional[str] = None*, *transliterate_keys: bool = True*)

Bases: *pycdsl.models.Entry*

Lexicon Entry

**Parameters**

- **lexicon_entry** (Lexicon) – Instance of Lexicon model

- **lexicon_id** (*str, optional*) – ID of the Lexicon to which the entry belongs
- **scheme** (*str, optional*) – Output transliteration scheme. If valid, parts of the *data* in lexicon which are enclosed in <*s*> tags will be transliterated to *scheme*. If invalid or None, no transliteration will take place. The default is None.
- **transliterate_keys** (*bool, optional*) – If True, the keys in lexicon will be transliterated to *scheme*. The default is True.

### pycdsl.shell module

REPL Shell for PyCDSL

**class** pycdsl.shell.**BasicShell**(*completekey: str = 'tab', stdin: Optional[TextIO] = None, stdout: Optional[TextIO] = None, *, persistent_history_file: str = '', persistent_history_length: int = 1000, startup_script: str = '', silence_startup_script: bool = False, include_py: bool = False, include_ipy: bool = False, allow_cli_args: bool = True, transcript_files: Optional[List[str]] = None, allow_redirection: bool = True, multiline_commands: Optional[List[str]] = None, terminators: Optional[List[str]] = None, shortcuts: Optional[Dict[str, str]] = None, command_sets: Optional[Iterable[cmd2.command_definition.CommandSet]] = None, auto_load_commands: bool = True*)

Bases: cmd2.cmd2.Cmd

An easy but powerful framework for writing line-oriented command interpreters. Extends Python's cmd package.

**Parameters**

- **completekey** – readline name of a completion key, default to Tab
- **stdin** – alternate input file object, if not specified, sys.stdin is used
- **stdout** – alternate output file object, if not specified, sys.stdout is used
- **persistent_history_file** – file path to load a persistent cmd2 command history from
- **persistent_history_length** – max number of history items to write to the persistent history file
- **startup_script** – file path to a script to execute at startup
- **silence_startup_script** – if True, then the startup script's output will be suppressed. Anything written to stderr will still display.
- **include_py** – should the "py" command be included for an embedded Python shell
- **include_ipy** – should the "ipy" command be included for an embedded IPython shell
- **allow_cli_args** – if True, then cmd2.Cmd.__init__() will process command line arguments as either commands to be run or, if -t or --test are given, transcript files to run. This should be set to False if your application parses its own command line arguments.
- **transcript_files** – pass a list of transcript files to be run on initialization. This allows running transcript tests when allow_cli_args is False. If allow_cli_args is True this parameter is ignored.
- **allow_redirection** – If False, prevent output redirection and piping to shell commands. This parameter prevents redirection and piping, but does not alter parsing behavior. A user can still type redirection and piping tokens, and they will be parsed as such but they won't do anything.

- **multiline_commands** – list of commands allowed to accept multi-line input
- **terminators** – list of characters that terminate a command. These are mainly intended for terminating multiline commands, but will also terminate single-line commands. If not supplied, the default is a semicolon. If your app only contains single-line commands and you want terminators to be treated as literals by the parser, then set this to an empty list.
- **shortcuts** – dictionary containing shortcuts for commands. If not supplied, then defaults to constants.DEFAULT_SHORTCUTS. If you do not want any shortcuts, pass an empty dictionary.
- **command_sets** – Provide CommandSet instances to load during cmd2 initialization. This allows CommandSets with custom constructor parameters to be loaded. This also allows the a set of CommandSets to be provided when *auto_load_commands* is set to False
- **auto_load_commands** – If True, cmd2 will check for all subclasses of *CommandSet* that are currently loaded by Python and automatically instantiate and register all commands. If False, CommandSets must be manually installed with *register_command_set*.

class pycdsl.shell.**CDSLShell**(*data_dir: Optional[str] = None*, *dict_ids: Optional[List[str]] = None*, *search_mode: Optional[str] = None*, *input_scheme: Optional[str] = None*, *output_scheme: Optional[str] = None*, *history_file: Optional[str] = None*, *startup_script: Optional[str] = None*)

Bases: *pycdsl.shell.BasicShell*

REPL Interface to CDSL

REPL Interface to CDSL

Create an instance of CDSLCorpus as per the providd parameters. CDSLCorpus.setup() is called after the command-loop starts.

**Parameters**

- **data_dir** (*str or None, optional*) – Load a CDSL installation instance at the location *data_dir*. Passed to CDSLCorpus instance as a keyword argument *data_dir*.
- **dict_ids** (*list or None, optional*) – List of dictionary IDs to setup. Passed to a CDSLCorpus.setup() as a keyword argument *dict_ids*.
- **search_mode** (*str or None, optional*) – Search mode to query by *key*, *value* or *both*. The default is None.
- **input_scheme** (*str or None, optional*) – Transliteration scheme for input. If None, *DEFAULT_SCHEME* is used. The default is None.
- **output_scheme** (*str or None, optional*) – Transliteration scheme for output. If None, *DEFAULT_SCHEME* is used. The default is None.
- **history_file** (*str or None, optional*) – Path to the history file to keep a persistant history. If None, the history does not persist across sessions. The default is None.
- **startup_script** (*str or None, optional*) – Path to the startup script with a list of startup commands to be executed after initialization. If None, no startup commands are run. The default is None.

```
intro = 'Cologne Sanskrit Digital Lexicon
(CDSL)\n---------------------------------------'

desc = 'Install or load dictionaries by typing `use [DICT_IDS..]` e.g. `use
MW`.\nType any keyword to search in the selected dictionaries. (help or ? for list
of options)'
```

**prompt = '(CDSL::None) '**

**schemes = ['devanagari', 'iast', 'itrans', 'velthuis', 'hk', 'slp1', 'wx']**

**search_modes = ['key', 'value', 'both']**

**__init__**(*data_dir: Optional[str] = None*, *dict_ids: Optional[List[str]] = None*, *search_mode: Optional[str] = None*, *input_scheme: Optional[str] = None*, *output_scheme: Optional[str] = None*, *history_file: Optional[str] = None*, *startup_script: Optional[str] = None*)

REPL Interface to CDSL

Create an instance of CDSLCorpus as per the providd parameters. CDSLCorpus.setup() is called after the command-loop starts.

**Parameters**

- **data_dir** (*str or None, optional*) – Load a CDSL installation instance at the location *data_dir*. Passed to CDSLCorpus instance as a keyword argument *data_dir*.
- **dict_ids** (*list or None, optional*) – List of dictionary IDs to setup. Passed to a CDSLCorpus.setup() as a keyword argument *dict_ids*.
- **search_mode** (*str or None, optional*) – Search mode to query by *key*, *value* or *both*. The default is None.
- **input_scheme** (*str or None, optional*) – Transliteration scheme for input. If None, *DEFAULT_SCHEME* is used. The default is None.
- **output_scheme** (*str or None, optional*) – Transliteration scheme for output. If None, *DEFAULT_SCHEME* is used. The default is None.
- **history_file** (*str or None, optional*) – Path to the history file to keep a persistant history. If None, the history does not persist across sessions. The default is None.
- **startup_script** (*str or None, optional*) – Path to the startup script with a list of startup commands to be executed after initialization. If None, no startup commands are run. The default is None.

**do_info**(*_: cmd2.parsing.Statement*)

Display information about active dictionaries

**do_stats**(*_: cmd2.parsing.Statement*)

Display statistics about active dictionaries

**do_dicts**(*_: cmd2.parsing.Statement*)

Display a list of dictionaries available locally

**do_available**(*_: cmd2.parsing.Statement*)

Display a list of dictionaries available in CDSL

**do_update**(*_: cmd2.parsing.Statement*)

Update loaded dictionaries

**use_parser = Cmd2ArgumentParser(prog='use', usage=None, description='\n Load the specified dictionaries from CDSL.\n If not available locally, they will be installed first.\n ', formatter_class=<class 'cmd2.argparse_custom.Cmd2HelpFormatter'>, conflict_handler='error', add_help=True)**

**do_use**(*namespace: argparse.Namespace*)

Load the specified dictionaries from CDSL. If not available locally, they will be installed first.

**show_parser = Cmd2ArgumentParser(prog='show', usage=None, description='Show a specific entry by ID', formatter_class=<class 'cmd2.argparse_custom.Cmd2HelpFormatter'>, conflict_handler='error', add_help=True)**

**do_show**(*namespace: argparse.Namespace*)

Show a specific entry by ID

**search_parser = Cmd2ArgumentParser(prog='search', usage=None, description='\n Search in the active dictionaries\n\n Note\n ----\n * Searching in the active dictionaries is also the default action.\n * In general, we do not need to use this command explicitly unless we\n want to search the command keywords, such as, `available` `search`,\n `version`, `help` etc. in the active dictionaries.\n ', formatter_class=<class 'cmd2.argparse_custom.Cmd2HelpFormatter'>, conflict_handler='error', add_help=True)**

**do_search**(*namespace: argparse.Namespace*)

Search in the active dictionaries

---

**Note:**

- Searching in the active dictionaries is also the default action.
- In general, we do not need to use this command explicitly unless we want to search the command keywords, such as, *available search*, *version*, *help* etc. in the active dictionaries.

---

**default**(*statement: cmd2.parsing.Statement*)

Executed when the command given isn't a recognized command implemented by a do_* method.

**Parameters** **statement** – Statement object with parsed input

**cmdloop**(*intro: Optional[cmd2.parsing.Statement] = None*)

This is an outer wrapper around _cmdloop() which deals with extra features provided by cmd2.

_cmdloop() provides the main loop equivalent to cmd.cmdloop(). This is a wrapper around that which deals with the following extra features provided by cmd2: - transcript testing - intro banner - exit code

**Parameters** **intro** – if provided this overrides self.intro and serves as the intro banner printed once at start

**do_version**(*_: cmd2.parsing.Statement*)

Show the current version of PyCDSL

### pycdsl.utils module

Utility Functions

pycdsl.utils.**validate_search_mode**(*mode: str*) → str

Validate the search mode

**Parameters** **mode** (*str*) – Search mode

**Returns** If mode is valid, mode.lower() otherwise, None.

**Return type** str or None

pycdsl.utils.**validate_scheme**(*scheme: str*) → str

Validate the name of transliteration scheme

**Parameters** **scheme** (*str*) – Name of the transltieration scheme

**Returns** If scheme is valid, scheme.lower() otherwise, None.

**Return type** str or None

pycdsl.utils.**transliterate_between**(*text: str*, *from_scheme: str*, *to_scheme: str*, *start_pattern: str*, *end_pattern: str*) → str

Transliterate the text appearing between two patterns

Only the text appearing between patterns *start_pattern* and *end_pattern* it transliterated. *start_pattern* and *end_pattern* can appear multiple times in the full text, and for every occurrence, the text between them is transliterated.

*from_scheme* and *to_scheme* should be compatible with scheme names from *indic-transliteration*

**Parameters**

- **text** (*str*) – Full text
- **from_scheme** (*str*) – Input transliteration scheme
- **to_scheme** (*str*) – Output transliteration scheme
- **start_pattern** (*regexp*) – Pattern describing the start tag
- **end_pattern** (*regexp*) – Pattern describing the end tag

#### Module contents

PyCDSL

Python Interface to Cologne Digital Sanskrit Lexicon (CDSL).

## 1.5 Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

### 1.5.1 Types of Contributions

#### Report Bugs

Report bugs at https://github.com/hrishikeshrt/PyCDSL/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

#### Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

#### Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

#### Write Documentation

Python Interface to Cologne Digital Sanskrit Lexicon (CDSL) could always use more documentation, whether as part of the official Python Interface to Cologne Digital Sanskrit Lexicon (CDSL) docs, in docstrings, or even on the web in blog posts, articles, and such.

#### Submit Feedback

The best way to send feedback is to file an issue at https://github.com/hrishikeshrt/PyCDSL/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

### 1.5.2 Get Started!

Ready to contribute? Here's how to set up *pycdsl* for local development.

1. Fork the *pycdsl* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/pycdsl.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv pycdsl
$ cd pycdsl/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 pycdsl tests
$ python setup.py test or pytest
$ tox
```

To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

### 1.5.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 3.6, 3.7, and 3.8 and for PyPy. Check https://travis-ci.com/hrishikeshrt/PyCDSL/pull_requests and make sure that the tests pass for all supported Python versions.

### 1.5.4 Tips

To run a subset of tests:

```
$ pytest tests.test_pycdsl
```

### 1.5.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bump2version patch # possible: major / minor / patch
$ git push
$ git push --tags
```

Travis will then deploy to PyPI if tests pass.

## 1.6 Credits

### 1.6.1 Development Lead

- Hrishikesh Terdalkar <hrishikeshrt@linuxmail.org>

### 1.6.2 Contributors

None yet. Why not be the first?

## 1.7 History

### 1.7.1 0.8.0 (2022-04-17)

- Add search mode support to all interfaces (Issue #24)
- Uniformly follow lower case convention for CLI help mesages
- Update documentation
- Fix bugs

### 1.7.2 0.7.0 (2022-03-17)

- Add the explicit REPL command `search`
- Add a REPL command `stats`
- Interpret arguments `all` and `none` to the REPL command `use`
- Add `lexicon_id` to `Entry` class
- Add a placeholder for post-init hook in `Entry`. If implemented, this will be run after `__init__()` of `Entry`
- Remove `model_map` from `CDSLDict` and add to `CDSLCorpus`
- Add tests for lexicon initalization, download, setup, transliteration, iteration, getitem, stats, entry, dump
- Add credits to CDSL website
- Update documentation
- Fix bugs

### 1.7.3 0.6.0 (2022-02-14)

- Add `__getitem__` method to `CDSLCorpus` to access loaded dictionaries using *[]* operator with `dict_id`
- Add `__getitem__` method to `CDSLDict` to access dictionary entries using *[]* operator with `entry_id`
- Add unit tests and integration tests for `pycdsl.utils`
- Add unit tests and integration tests for `pycdsl.corpus`
- Update documentation
- Fix bugs

### 1.7.4 0.5.0 (2022-02-13)

- Add `model_map` argument to `CDSLDict.connect` for better customization
- Make `CDSLCorpus` iterable (iterate over loaded dictionaries)
- Make `CDSLDict` iterable (iterate over dictionary entries)
- Update documentation

### 1.7.5 0.4.0 (2022-02-12)

- Add ability to limit and offset the number of search results
- Add `.to_dict()` method to `Entry` class
- Add multi-dictionary `.search()` from `CDSLCorpus`
- Add support for multiple active dictionaries in REPL
- Improve code structure (more modular)
- Improve documentation formatting
- Update documentation
- Fix bugs

### 1.7.6 0.3.0 (2022-02-07)

- Functional CLI (console command) for dictionary search
- Integration of existing REPL into the CLI. (`--interactive`)
- Extend transliteration support on Corpus, Dictionary, Search and Entry level
- Make the package Python 3.6 compatibile

### 1.7.7 0.2.0 (2022-02-05)

- Improve dictionary setup
- Add a function to dump data
- Add logging support
- Add transliteration support using `indic-transliteration`

### 1.7.8 0.1.0 (2022-01-28)

- First release on PyPI.

CHAPTER

TWO

# INDICES AND TABLES

- genindex
- modindex
- search

# PYTHON MODULE INDEX

## p

# INDEX

## Symbols



## A



## B



## C



## D



## E



## G



## I



## K



## L



## M

## N



## O



## P



## S



## T



## U



## V