17.8. `readline` — GNU readline interface¶

Platforms: Unix

The `readline` module defines a number of functions to facilitate completion and reading/writing of history files from the Python interpreter. This module can be used directly or via the `rlcompleter` module. Settings made using this module affect the behaviour of both the interpreter's interactive prompt and the prompts offered by the `raw_input()` and `input()` built-in functions.

..note:

```
On MacOS X the :mod:`readline` module can be implemented using
the ``libedit`` library instead of GNU readline.

The configuration file for ``libedit`` is different from that
of GNU readline. If you programmaticly load configuration strings
you can check for the text "libedit" in :const:`readline.__doc__`
to differentiate between GNU readline and libedit.
```

The `readline` module defines the following functions:

`readline.parse_and_bind`(*string*)¶
Parse and execute single line of a readline init file.

`readline.get_line_buffer`()¶
Return the current contents of the line buffer.

`readline.insert_text`(*string*)¶
Insert text into the command line.

`readline.read_init_file`([*filename*])¶
Parse a readline initialization file. The default filename is the last filename used.

`readline.read_history_file`([*filename*])¶
Load a readline history file. The default filename is `~/.history`.

`readline.write_history_file`([*filename*])¶
Save a readline history file. The default filename is `~/.history`.

`readline.clear_history`()¶

Clear the current history. (Note: this function is not available if the installed version of GNU readline doesn't support it.)

New in version 2.4.

`readline.get_history_length`()¶
Return the desired length of the history file. Negative values imply unlimited history file size.

`readline.set_history_length`(*length*)¶
Set the number of lines to save in the history file. `write_history_file()` uses this value to truncate the history file when saving. Negative values imply unlimited history file size.

`readline.get_current_history_length`()¶

Return the number of lines currently in the history. (This is different from `get_history_length()`, which returns the maximum number of lines that will be written to a history file.)

New in version 2.3.

`readline.get_history_item`(*index*)¶

Return the current contents of history item at *index*.

New in version 2.3.

`readline.remove_history_item`(*pos*)¶

Remove history item specified by its position from the history.

New in version 2.4.

`readline.replace_history_item`(*pos*, *line*)¶

Replace history item specified by its position with the given line.

New in version 2.4.

`readline.redisplay`()¶

Change what's displayed on the screen to reflect the current contents of the line buffer.

New in version 2.3.

`readline.set_startup_hook`([*function*])¶
Set or remove the startup_hook function. If *function* is specified, it will be used as the new startup_hook function; if omitted or `None`, any hook function already installed is removed. The startup_hook function is called with no arguments just before readline prints the first prompt.

`readline.set_pre_input_hook`([*function*])¶
Set or remove the pre_input_hook function. If *function* is specified, it will be used as the new pre_input_hook function; if omitted or `None`, any hook function already installed is removed. The pre_input_hook function is called with no arguments after the first prompt has been printed and just before readline starts reading input characters.

`readline.set_completer`([*function*])¶
Set or remove the completer function. If *function* is specified, it will be used as the new completer function; if omitted or `None`, any completer function already installed is removed. The completer function is called as `function(text, state)`, for *state* in `0`, `1`, `2`, ..., until it returns a non-string value. It should return the next possible completion starting with *text*.

`readline.get_completer`()¶

Get the completer function, or `None` if no completer function has been set.

New in version 2.3.

`readline.get_completion_type`()¶

Get the type of completion being attempted.

New in version 2.6.

`readline.get_begidx`()¶
Get the beginning index of the readline tab-completion scope.

`readline.get_endidx`()¶
Get the ending index of the readline tab-completion scope.

`readline.set_completer_delims`(*string*)¶
Set the readline word delimiters for tab-completion.

`readline.get_completer_delims`()¶
Get the readline word delimiters for tab-completion.

`readline.set_completion_display_matches_hook`([*function*])¶

Set or remove the completion display function. If *function* is specified, it will be used as the new completion display function; if omitted or `None`, any completion display function already installed is removed. The completion display function is called as `function(substitution, [matches], longest_match_length)` once each time matches need to be displayed.

New in version 2.6.

`readline.add_history`(*line*)¶
Append a line to the history buffer, as if it was the last line typed.

See also

Module `rlcompleter`
Completion of Python identifiers at the interactive prompt.

17.8.1. Example¶

The following example demonstrates how to use the `readline` module's history reading and writing functions to automatically load and save a history file named `.pyhist` from the user's home directory. The code below would normally be executed automatically during interactive sessions from the user's **PYTHONSTARTUP** file.

```
import os
histfile = os.path.join(os.environ["HOME"], ".pyhist")
try:
   readline.read_history_file(histfile)
except IOError:
   pass
import atexit
atexit.register(readline.write_history_file, histfile)
del os, histfile
```

The following example extends the `code.InteractiveConsole` class to support history save/restore.

```
import code
import readline
import atexit
import os

class HistoryConsole(code.InteractiveConsole):
   def __init__(self, locals=None, filename="<console>",
                histfile=os.path.expanduser("~/.console-history")):
       code.InteractiveConsole.__init__(self, locals, filename)
       self.init_history(histfile)

   def init_history(self, histfile):
       readline.parse_and_bind("tab: complete")
       if hasattr(readline, "read_history_file"):
           try:
               readline.read_history_file(histfile)
           except IOError:
               pass
           atexit.register(self.save_history, histfile)

   def save_history(self, histfile):
       readline.write_history_file(histfile)
```


Previous topic

17.7. `mmap` — Memory-mapped file support

Next topic

17.9. `rlcompleter` — Completion function for GNU readline

This Page

- Show Source

Navigation

- index
- modules |
- next |
- previous |
- Python v2.6.4 documentation »
- The Python Standard Library »
- 17. Optional Operating System Services »