cli$_demoDocumentation$

Release 0.0.1

Han Keong

May 06, 2018

Contents

Welcome to the documentation page of cli_demo!

CHAPTER 1

Contents

1.1 About

`cli_demo` provides a framework for interactive demonstrations in a command line interface.

1.1.1 Features

Registering an option

There are various ways to `register()` an option:

- Registering with an expected user response

```
@options.register("r", "Restart."):
def restart(self):
    ...  # Restart demo
```

- Registering with an input function key

```
@options.register("setup"):
def setup_callback(self, response):
    ...  # Process response.
```

- Setting newline to True

```
@options.register("h", "Help." newline=True):
def print_help(self):
    print("This is the help text.")
    ...  # Print the help text
```

```
>>> Enter an input: h

This is the help text.  # A gap is inserted beforehand.
...
```

- Setting retry to True

```
@options.register("echo", retry=True):
def echo_response(self, response):
    print("Got:", response)
```

```
>>> Enter an input: hello
Got: hello
>>> Enter an input:  # The input function is called again.
```

- Setting lock to True

```
@options.register("o", lock=True):
def print_options(self, key):
    if key == "setup":
        ...  # Print setup options
    elif key == "echo":
        ...  # Print echo options
```

CodeDemo

CodeDemo information here.

SandboxDemo

SandboxDemo information here.

1.1.2 Credits

cli_demo was written by Han Keong <hk997@live.com>.

1.2 Documentation

This module contains a framework for interactive command line demonstrations.

Examples

Making a simple CodeDemo subclass:

```
# spam.py
from cli_demo import CodeDemo

def scramble(num):
    return "SCRAMBLE " * num

class SpamDemo(CodeDemo):
    help_text = "An eggs and bacon bonanza."

    setup_code = '''\
eggs = 6
spam = 42'''

    commands = [
        "eggs + spam  # yum",
        "bacon = spam % eggs",
        "eggs // bacon",
        "scramble(eggs)",
        "response + ' was your response!'"
    ]
```

Running a Demo:

```
>>> from spam import SpamDemo
>>> demo = SpamDemo()
>>> demo.run()
Welcome to SpamDemo!

Options:
 *: Any response.
 h: Help.
 o: Options.
 r: Restart.
 q: Quit.

Select an option, or type something random: h

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
====
Help
====

SpamDemo
--------

An eggs and bacon bonanza.

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Select an option, or type something random: noodles

Setup:
>>> eggs = 6
>>> spam = 42

Options:
 0: "eggs + spam  # yum"
```

(continues on next page)

(continued from previous page)

```
 1: "bacon = spam % eggs"
 2: "eggs // bacon"
 3: "scramble(eggs)"
 4: "response + ' was your response!'"
 a: Execute all of the above.
 c: Setup code.
 o: Options.
 r: Restart.
 q: Quit.

Choose a command: a
>>> eggs + spam  # yum
48

>>> bacon = spam % eggs
>>> bacon
0

>>> eggs // bacon
ZeroDivisionError: integer division or modulo by zero

>>> scramble(eggs)
'SCRAMBLE SCRAMBLE SCRAMBLE SCRAMBLE SCRAMBLE SCRAMBLE '

>>> response + ' was your response!'
'noodles was your response!'

Choose a command: c

Setup:
>>> spam = 6
>>> eggs = 42

Choose a command: q
Goodbye!
```

Making a Demo script:

```
# spam.py
...
...

if __name__ == "__main__":
    demo = SpamDemo()
    demo.run()
```

```
>>> python spam.py
Welcome to SpamDemo!
...
...
```

```
>>> python3 spam.py
Welcome to SpamDemo!
...
...
```

1.2.1 Demo

class cli_demo.demo.**Demo**

A basic framework for interactive demonstrations in a command line interface.

help_text

str – The help text used in *print_help()*.

setup_prompt

str – The input prompt for *run_setup()*.

options

A *DemoOptions* instance for *registering* option callbacks and *designating* options to input functions.

Warning: When inheriting *options* from a *Demo* superclass, either a new *DemoOptions* instance should be created:

```
class NewDemo(Demo):
    options = DemoOptions()
    ...
```

Or a copy should be made by calling *copy()*:

```
class DemoSubclass(Demo):
    options = Demo.options.copy()
    ...
```

This is to avoid mangling options between superclass and subclasses.

Program logic of a Demo instance

Demo.**run**(**args*, ***kwargs*)

The main logic of a *Demo* program.

First, call *print_intro()*, then print the options for *run_setup()* using *print_options()* before calling *run_setup()*.

Note: *run()* is decorated with:

```
@catch_exc
def run(self):
    ...
```

For more information, refer to *catch_exc()*.

setup process of a Demo instance

Demo.**run_setup**(**args*, ***kwargs*)

Prompt the user for input for the setup process.

Note: *run_setup()* is decorated with:

```
@options("h", "o", "r", "q", key="setup")
def run_setup(self):
    ...
```

For more information, refer to `options`.

`Demo.`**`setup_callback`**`(`*response*`)`

Handle user input to `run_setup()`.

Parameters **`response`** (`str`) – The user input to `run_setup()`.

Note: `setup_callback()` is decorated with:

```
@options.register("setup", retry=True)
def setup_callback(self, response):
    ...
```

For more information, refer to `options.register`.

`Demo.`**`setup_options`**`()`

Provide options for `run_setup()`.

Note: The default option is `"*"` with description `"Any response."`.

Print functions of a Demo instance

`Demo.`**`print_intro`**`()`

Print the welcome text once.

Note: After `print_intro()` is called for the first time, calling it again will no longer have any effect.

`Demo.`**`print_options`**`(`**opts*, ***key*`)`

Print what responses are allowed for an input function.

Parameters

- **`*opts`** (`str`) – Which options to print.
- **`**key`** (`str`) – An input function key.

Note:

- If an input function *key* is provided, `print_options()` will do the following:
  1. Retrieve options and descriptions (in a tuple) from `key_options()` - a function that starts with *key* and ends in '_options'- if it is defined.
  2. Get options from `get_options()` using the input function *key*.
- Options are printed in the following order:
  1. Options from `key_options()`
  2. Keyword options from `get_options()`

3. Argument options from `get_options()`
4. Argument options passed into `print_options()`

- Besides the options from `key_options()`, option descriptions are taken from the `desc` of the `Option` instance registered under it. If an option is not `registered`, then `""` is used for the description.
- `print_options()` is decorated with:

```
@options.register("o", "Options", retry=True, lock=True, newline=True)
def print_options(self, *opts, **key):
    ...
```

For more information, refer to `options.register`.

`Demo.`**`print_help`**`(`***kwargs*`)`

Format and print `help_text`.

Parameters

- **`symbols`** (*`list`*) – A list of symbols for each level of indentation. Defaults to `[" ", "", "", "", ""]`.
- **`width`** (*`int`*) – The maximum width for a line printed. Defaults to `60`.
- **`indent`** (*`int`*) – The number of spaces per indent for the text printed. Defaults to `4`.
- **`border`** (*`str`*) – The character used for the border for `help_text`. Defaults to `"~"`.
- **`title`** (*`str`*) – The character used for the border for the `"Help"` title. Defaults to `"="`.
- **`subtitle`** (*`str`*) – The character used for the border for the name of each `Demo` subclass. Defaults to `"-"`.
- **`include`** (*`bool`*) – Whether to include the `help_text` of all superclasses that are subclasses of `Demo`. Defaults to `False`.

Note: `print_help()` is decorated with:

```
@options.register("h", "Help.", retry=True, newline=True)
def print_help(self, **kwargs):
    ...
```

For more information, refer to `options.register`.

Control flow tools of a Demo instance

`Demo.`**`restart`**`(`*text=None*`)`

Restart the main `run()` loop.

Parameters **`text`** (*`str, optional`*) – The text to print when restarting.

Raises `DemoRestart`

Note: `restart()` is decorated with:

```
@options.register("r")
def restart(self, text=None):
    ...
```

For more information, refer to `options.register`.

`Demo.`**`quit`**`(`*text=None*`)`

Break out of the main `run()` loop.

Parameters **`text`** (*`str, optional`*) – The text to print when quitting.

Raises `DemoQuit`

Note: `quit()` is decorated with:

```
@options.register("q")
def quit(self, text=None):
    ...
```

For more information, refer to `options.register`.

`Demo.`**`retry`**`(`*text=None*`)`

Go back to the last input function.

Parameters **`text`** (*`str, optional`*) – The text to print when retrying.

Raises *`DemoRetry`*

1.2.2 CodeDemo

`class` `cli_demo.code.`**`CodeDemo`**

Bases: *`cli_demo.demo.Demo`*

CodeDemo improves Demo by introducing a feature called *`commands`*, which allows the user to select from a set of code snippets and view the result of it being passed into *`execute()`*.

`setup_code`

str – The code to run in *`setup_callback()`*.

`command_prompt`

str – The input prompt for *`get_commands()`*.

`commands`

list[str] – The code snippets for the user to choose from in *`get_commands()`*.

`locals`

dict – The local namespace populated in *`setup_callback()`*.

`globals`

dict – The global namespace populated in *`setup_callback()`*.

Program logic of a CodeDemo instance

`CodeDemo.`**`run`**`(`**args, **kwargs*`)`

The main logic of a *`CodeDemo`* program.

First, call `print_intro()`, then print the options for `run_setup()` using `print_options()` before calling `run_setup()`, and then repeat the same process for `get_commands()`.

Note: `run()` is decorated with:

```
@catch_exc
def run(self):
    ...
```

`setup` process of a CodeDemo instance

`CodeDemo.`**`setup_callback`**`(`*response*`)`

Handle user input to `run_setup()`.

Set `locals` to the global namespace of `__main__` before updating with *response*. Then, copy the `__builtins__` of `__main__` into `globals`. Finally, `exec` `setup_code` in `locals` and `globals` before printing it using `print_setup()`.

Parameters **`response`** (*`str`*) – The user input to `run_setup()`.

Note:

- The `CodeDemo` instance is available in `locals` under the name *demo*, and the user response under *response*.
- `setup_callback()` is decorated with:

```
@options.register("setup")
def setup_callback(self, response):
    ...
```

For more information, refer to `options.register`.

`commands` process of a CodeDemo instance

`CodeDemo.`**`get_commands`**`(`**args*, ***kwargs*`)`

Prompt the user to select a command from `commands`.

Note: `get_commands()` is decorated with:

```
@options("c", "o", "r", "q", key="commands")
def get_commands(self):
    ...
```

For more information, refer to `options`.

`CodeDemo.`**`commands_callback`**`(`*response*`)`

Handle user input to `get_commands()`.

`execute()` the respective code snippet or all `commands` if *response* is a valid index or `"a"`. Otherwise, `retry()` with the error message: `"Invalid index. Please try again."`.

Parameters **response** (*str*) – The user input to *get_commands()*.

Note: *commands_callback()* is decorated with:

```
@options.register("commands", retry=True)
def commands_callback(self, response):
    ...
```

For more information, refer to *options.register*.

CodeDemo.**commands_options**()
Provide options for *get_commands()*.

Note:

- The descriptions and options are the code snippets and their enumerations.
- An additional option is "a", which is "Execute all of the above.".

CodeDemo.**execute**(*commands*, *print_in=True*)
exec each command in *locals* and *globals*.

print_in() the command if *print_in* is True. Remove any comments, then compile the command if there are multiple lines or assignments. exec the code snippet, and *print_out()* the result or catch and print any errors. If there are any assignments in the code snippet, *execute()* their assigned names.

Parameters

- **commands** (*list*) – The code snippets to exec.
- **print_in** (*bool*) – Whether to *print_in()* a command.

Print functions of a CodeDemo instance

CodeDemo.**print_setup**()
Print *setup_code*.

Note: *print_setup()* is decorated with:

```
@options.register("c", "Setup code.", retry=True, newline=True)
def print_setup(self):
    ...
```

For more information, refer to *options.register*.

CodeDemo.**print_in**(*text*)
Print each line in *text* starting with ">>>" or "...".

CodeDemo.**print_out**(**args*)
Pretty-print *args* using pprint().

1.2.3 SandboxDemo

class cli_demo.sandbox.**SandboxDemo**

Bases: *cli_demo.code.CodeDemo*

SandboxDemo extends CodeDemo by providing *sandbox()*, a Python shell in which users can experiment with the context that has been set up.

commands process of a SandboxDemo instance

SandboxDemo.**get_commands**(**args*, ***kwargs*)

Prompt the user to select a command from *commands*.

Note:

- *get_commands()* is decorated with:

```
@options("c", "o", "s", "r", "q", key="commands")
def get_commands(self):
    ...
```

  For more information, refer to *options*.

- "s", for *"Sandbox mode."*, has been added to the available options.

SandboxDemo.**sandbox**(*key*)

Set up an interactive shell to experiment with.

Prompt the user for input, *execute()* the entered command or code block, and then repeat. If the input is "quit()", print the previous options using *print_options()* and return.

Parameters key (*str*) – The key of the input function which triggered sandbox mode.

Note: *sandbox()* is decorated with:

```
@options.register("s", "Sandbox mode.", retry=True, lock=True)
def sandbox(self, key):
    ...
```

For more information, refer to *options.register*.

1.2.4 DemoOptions

class cli_demo.options.**DemoOptions**

Designates options for input functions and forwards their registered callbacks dynamically.

demo

The parent *Demo* instance.

registry

dict – The options and their *Option* instances that have been registered.

cache

dict – A cache of input function key ids and their options and keyword options that have been captured.

Designating options for an input function

`DemoOptions.`**`__call__`**`(`**opts*, ***kw_opts*`)`

Designate a set of options to an input function.

If a user input falls within the designated options, invoke the `callback` of the corresponding `Option` instance through its `call()` method.

Parameters

- **`retry`** (*`str, optional`*) – The text to print before the input function is called again when the user response is invalid. Defaults to `"Please try again"`.
- **`key`** (*`str, optional`*) – The key of the input function.
- **`args`** (*`tuple, optional`*) – The arguments that should be passed into the `callback` of the `Option` instance registered under *key*. Defaults to `()`.
- **`kwargs`** (*`dict, optional`*) – The keyword arguments that should be passed into `callback` of the `Option` instance registered under *key*. Defaults to `{}`.
- **`*opts`** – The user responses that should be accepted.
- **`**kw_opts`** – The user responses that should be redirected.

Note: If *key* is provided:

- *key* will be used to store a record of *opts* and *kw_opts* in `cache`.
- To reference the options stored in `cache` when calling `print_options()`, you can pass in *key* as the *key* argument.
- If a user input does not fall within the designated options, the response will be forwarded to the `callback` of the `Option` instance registered under *key* through its `call()` method.

If *key* is not provided:

- The **input function** itself will be used to store a record of *opts* and *kw_opts* in `cache`.
- To reference the options stored in `cache` when calling `print_options()`, you need to pass in the input function itself as the *key* argument.
- If a user input does not fall within the designated options, `retry()` will be called and *retry* will be printed.

Returns A decorator which takes a function (expected to be an input function) and returns a wrapped function.

Return type `options_decorator()`

The following exceptions will only be raised when the wrapped function is called.

Raises

- `OptionNotFoundError` – If an option does not exist in `registry`, or if its value is not an instance of `Option`.
- `CallbackNotFoundError` – If the `callback` of an `Option` instance has not been set.
- `CallbackLockError` – If the `lock` attribute of an `Option` instance is `True` but its `callback` does not accept a *key* argument.

- `CallbackResponseError` – If *key* is provided but the `callback` of the `Option` instance registered under *key* does not accept a *response* argument.

Getting the options of an input function

`DemoOptions.`**`get_options`**`(`*key*`)`

Get the options that were set with *key*.

Parameters **`key`** – A key for a set of options and keyword options.

Returns The options and keyword options set under *key*.

Return type list[list, dict]

Raises *`KeyNotFoundError`* – If the id of *key* does not exist in *`cache`*.

`DemoOptions.`**`has_options`**`(`*key*`)`

Check if there are any options set with *key*.

Parameters **`key`** – A key for a set of options and keyword options.

Returns `True` if the id of *key* exists in *`cache`*, `False` otherwise.

`static` `DemoOptions.`**`get_id`**`(`*key*`)`

Create a unique id for *key*.

Parameters **`key`** – A key for a set of options and keyword options.

Returns The id of *key*.

Return type int

Setting the options of an input function

`DemoOptions.`**`set_options`**`(`*key*, **opts*, ***kw_opts*`)`

Change the options that were set with *key*.

If *opts* or *kw_opts* are provided, override the options or keyword options that were recorded previously.

Parameters

- **`key`** – A key for a set of options and keyword options.
- **`*opts`** – Argument options for *key*.
- **`**kw_opts`** – Keyword options for *key*.

`DemoOptions.`**`insert`**`(`*key*, *kw*, *opt*, ***kw_opts*`)`

Insert an option into the options that were set with *key*.

Insert *opt* into the argument options at index *kw* if it is an int or a digit. Otherwise, update the keyword options with *kw* and *opt*.

Parameters

- **`key`** – A key for a set of options and keyword options.
- **`kw`** – An index for argument options or a keyword option.
- **`opt`** (*`str`*) – The option to insert.
- **`**kw_opts`** – More *kw* and *opt* arguments.

Raises *`KeyNotFoundError`* – If the id of *key* does not exist in *`cache`*.

Registering an Option instance

`DemoOptions.`**`register`**`(`*option*, *desc=”*, ***kwargs*`)`

Register an option.

Create an `Option` instance based on the arguments and keyword arguments provided and then store in `registry`.

Returns A decorator which takes a function, sets the `callback` of the `Option` instance using `set_callback()`, and returns the original function.

Return type `register_decorator()`

Parameters

- **`option`** (*`str`*) – The name of the option.
- **`desc`** (*`str, optional`*) – The description of the option that should be printed in `print_options()`. If not provided, it will be set to the name of the function passed into `set_callback()`.
- **`newline`** (*`bool, optional`*) – Whether an empty line should be printed before `callback` is called. Defaults to `False`.
- **`retry`** (*`bool, optional`*) – Whether an input function should be called again once `callback` has returned. Defaults to `False`.
- **`lock`** (*`bool, optional`*) – Whether the *key* of a triggering input function should be received by `callback`. Defaults to `False`.
- **`args`** (*`tuple, optional`*) – The default arguments that should be used to call `callback`. Defaults to `()`.
- **`kwargs`** (*`dict, optional`*) – The default keyword arguments that should be used to call `callback`. Defaults to `{}`.

Note:

- *option* can be an expected user response or an input function key.
- If *option* is an input function key:
  - The function passed into `register_decorator()` must accept a *response* argument- the user’s response to that input function.
  - Any response to that input function which does not fall within its designated options will be forwarded to the function through the `call()` method of the `Option` instance for further processing.
- If *lock* is `True`, the function passed into `register_decorator()` must accept a *key* argument- the key of the input function that triggered it.

`class` `cli_demo.options.`**`Option`**`(`***kwargs*`)`

Holds information about a registered option.

`name`

str – The name of the option.

`desc`

str – The description of the option that should be printed in `print_options()`.

`callback`

function – The function that `call()` should wrap.

`newline`
bool – Whether an empty line should be printed before `callback` is called in `call()`.

`retry`
bool – Whether an input function should be called again once `callback` has returned.

`lock`
bool – Whether the *key* of a triggering input function should be received by `callback`.

`args`
tuple – The default arguments that should be used to call `callback` in `call()`.

`kwargs`
dict – The default keyword arguments that should be used to call `callback` in `call()`.

Invoking the callback of an Option instance

`DemoOptions.`**`call`**`(`*option*, **args*, ***kwargs*`)`
Invoke the `callback` of the `Option` instance through its `call()` method.

Parameters

- **`option`** (`str`) – The `name` used to register the `Option` instance.
- **`*args`** – The arguments to use when calling `callback`.
- **`**kwargs`** – The keyword arguments to use when calling `callback`.

Returns The return value of `callback`.

Raises

- `DemoException` – If `demo` is not set.
- `OptionNotFoundError` – If *option* does not exist in `registry`, or if its value is not an instance of `Option`.
- `CallbackNotFoundError` – If the `callback` of the `Option` instance has not been set.

`Option.`**`call`**`(`*demo*, **args*, ***kwargs*`)`
Call the registered `callback`.

Parameters

- **`demo`** – The `Demo` instance that should be passed into `callback`.
- **`*args`** – The arguments that should be passed into `callback`.
- **`**kwargs`** – The keyword arguments that should be passed into `callback`.

Note:

- `args` is used if *args* is empty.
- `kwargs` is used if *kwargs* is empty.
- An empty line is printed before `callback` is called if `newline` is `True`.
- `retry()` will be called if `retry` is `True` and `callback` successfully returned.

Getting attributes of an Option instance

`DemoOptions.`**`__contains__`**`(`*option*`)`

Check if an `Option` instance is registered.

Parameters **`option`** (*`str`*) – The `name` used to register the `Option` instance.

Returns `True` if *option* exists in `registry` and its value is an instance of `Option`, `False` otherwise.

`DemoOptions.`**`__getitem__`**`(`*option*`)`

Get the registered `Option` instance.

Parameters **`option`** (*`str`*) – The `name` used to register the `Option` instance.

Returns The `Option` instance registered under *option*.

Raises `OptionNotFoundError` – If *option* does not exist in `registry`, or if its value is not an instance of `Option`.

`DemoOptions.`**`get_callback`**`(`*option*`)`

Get the `call()` method of the `Option` instance.

Parameters **`option`** (*`str`*) – The `name` used to register the `Option` instance.

Returns The `call()` method of the `Option` instance.

Raises

- `OptionNotFoundError` – If *option* does not exist in `registry`, or if its value is not an instance of `Option`.
- `CallbackNotFoundError` – If the `callback` of the `Option` instance has not been set.

`DemoOptions.`**`is_lock`**`(`*option*`)`

Check if the *key* of a triggering input function will be received by the `callback` of the `Option` instance.

Parameters **`option`** (*`str`*) – The `name` used to register the `Option` instance.

Returns `True` if the `lock` attribute of the `Option` instance is `True`, `False` otherwise.

Raises `OptionNotFoundError` – If *option* does not exist in `registry`, or if its value is not an instance of `Option`.

`DemoOptions.`**`will_retry`**`(`*option*`)`

Check if an input function will be called again once the `callback` of the `Option` instance has returned.

Parameters **`option`** (*`str`*) – The `name` used to register the `Option` instance.

Returns `True` if the `retry` attribute of the `Option` instance is `True`, `False` otherwise.

Raises `OptionNotFoundError` – If *option* does not exist in `registry`, or if its value is not an instance of `Option`.

`DemoOptions.`**`has_newline`**`(`*option*`)`

Check if an empty line will be printed before the `callback` of the `Option` instance is called.

Parameters **`option`** (*`str`*) – The `name` used to register the `Option` instance.

Returns `True` if the `newline` attribute of the `Option` instance is `True`, `False` otherwise.

Raises `OptionNotFoundError` – If *option* does not exist in `registry`, or if its value is not an instance of `Option`.

`DemoOptions.`**`get_desc`**`(`*option*`)`

Get the description of the `Option` instance.

Parameters **`option`** (`str`) – The `name` used to register the `Option` instance.

Returns The `desc` attribute of the `Option` instance.

Return type str

Raises `OptionNotFoundError` – If *option* does not exist in `registry`, or if its value is not an instance of `Option`.

`DemoOptions.`**`get_args`**`(`*option*`)`

Get the default arguments that will be used to call the `callback` of the `Option` instance.

Parameters **`option`** (`str`) – The `name` used to register the `Option` instance.

Returns The `args` attribute of the `Option` instance.

Return type tuple

Raises `OptionNotFoundError` – If *option* does not exist in `registry`, or if its value is not an instance of `Option`.

`DemoOptions.`**`get_kwargs`**`(`*option*`)`

Get the default keyword arguments that will be used to call the `callback` of the `Option` instance.

Parameters **`option`** (`str`) – The `name` used to register the `Option` instance.

Returns The `kwargs` attribute of the `Option` instance.

Return type dict

Raises `OptionNotFoundError` – If *option* does not exist in `registry`, or if its value is not an instance of `Option`.

Setting attributes of an Option instance

`DemoOptions.`**`set_callback`**`(`*option*, *callback*`)`

Set the `callback` of the `Option` instance.

If the `desc` of the `Option` instance is blank, use the name of *callback* to set it.

Parameters

- **`option`** (`str`) – The `name` used to register the `Option` instance.
- **`callback`** – The function that the `call()` method of the `Option` instance should wrap.

Raises `OptionNotFoundError` – If *option* does not exist in `registry`, or if its value is not an instance of `Option`.

`DemoOptions.`**`set_lock`**`(`*option*, *lock*`)`

Set whether the *key* of a triggering input function should be received by the `callback` of the `Option` instance.

Parameters

- **`option`** (`str`) – The `name` used to register the `Option` instance.
- **`lock`** (`bool`) – Whether the *key* of a triggering input function should be received by `callback`.

Raises `OptionNotFoundError` – If *option* does not exist in `registry`, or if its value is not an instance of `Option`.

`DemoOptions.`**`set_retry`**`(`*option*, *retry*`)`

Set whether an input function should be called again once the `callback` of the `Option` instance has returned.

Parameters

- **`option`** (`str`) – The `name` used to register the `Option` instance.
- **`retry`** (`bool`) – Whether an input function should be called again once `callback` has returned.

Raises `OptionNotFoundError` – If *option* does not exist in `registry`, or if its value is not an instance of `Option`.

`DemoOptions.`**`set_newline`**`(`*option*, *newline*`)`

Set whether an empty line should be printed before the `callback` of the `Option` instance is called.

Parameters

- **`option`** (`str`) – The `name` used to register the `Option` instance.
- **`newline`** (`bool`) – Whether an empty line should be printed before `callback` is called.

Raises `OptionNotFoundError` – If *option* does not exist in `registry`, or if its value is not an instance of `Option`.

`DemoOptions.`**`set_desc`**`(`*option*, *desc*`)`

Set the description of the `Option` instance.

Parameters

- **`option`** (`str`) – The `name` used to register the `Option` instance.
- **`desc`** (`str`) – The description that should be printed in `print_options()`.

Raises `OptionNotFoundError` – If *option* does not exist in `registry`, or if its value is not an instance of `Option`.

`DemoOptions.`**`set_args`**`(`*option*, **args*`)`

Set the default arguments that should be used to call the `callback` of the `Option` instance.

Parameters

- **`option`** (`str`) – The `name` used to register the `Option` instance.
- **`*args`** – The default arguments that should be used to call `callback` in `call()`.

Raises `OptionNotFoundError` – If *option* does not exist in `registry`, or if its value is not an instance of `Option`.

`DemoOptions.`**`set_kwargs`**`(`*option*, **kwargs`)`

Set the default keyword arguments that should be used to call the `callback` of the `Option` instance.

Parameters

- **`option`** (`str`) – The `name` used to register the `Option` instance.
- **`**kwargs`** – The default keyword arguments that should be used to call `callback` in `call()`.

Raises `OptionNotFoundError` – If *option* does not exist in `registry`, or if its value is not an instance of `Option`.

Inheriting an instance of DemoOptions / Option

DemoOptions.**copy**()

Initialize a new copy of *DemoOptions*.

Returns An instance of *DemoOptions* with a deep copy of the *cache* and *registry* belonging to `self`.

Option.**copy**()

Initialize a new copy of *Option*.

Returns An instance of *Option* with a deep copy of all attributes belonging to `self`.

1.2.5 exceptions

This module contains exceptions for Demo.

cli_demo.exceptions.**catch_exc**(**demo_exc*)

Catch instances of *demo_exc* raised while running a function.

Parameters ***demo_exc** – One or a few subclasses of *DemoException*, and possibly a function to wrap.

Returns A decorator that takes a function and returns a wrapped function. As a shortcut, if a function was passed into *demo_exc*, the wrapped function is returned instead.

Return type `catch_exc_decorator()`

Note:

- Non-subclasses of *DemoException* are ignored, aside from a function or method.
- *DemoException* is the default if no subclasses are provided.
- Non-instances of *demo_exc* will not be caught. They should typically be handled by a higher level and more general kind of *catch_exc()*.
- If a `KeyboardInterrupt` is raised while running the function, it will be caught and *DemoExit* will be re-raised.

exception cli_demo.exceptions.**DemoException**(*text=None*)

Bases: `exceptions.Exception`

Base exception for any error raised in a *Demo*.

text

str – The text to print when an instance of *DemoException* is caught in *catch_exc()*.

__init__(*text=None*)

Format (if `"{}"` is present) or override *text* if *text* is provided.

Parameters **text** (*str, optional*) – A custom error text.

exception cli_demo.exceptions.**DemoRestart**(*text=None*)

Bases: *cli_demo.exceptions.DemoException*

Raised when user wants to restarts a *Demo*.

exception cli_demo.exceptions.**DemoExit**(*text=None*)

Bases: *cli_demo.exceptions.DemoException*

Raised when user wants to quit a *Demo*.

exception `cli_demo.exceptions.`**`DemoRetry`**`(`*text=None*`)`

Bases: *`cli_demo.exceptions.DemoException`*

Raised when an input function in a *`Demo`* should be called again.

exception `cli_demo.exceptions.`**`KeyNotFoundError`**`(`*text=None*`)`

Bases: *`cli_demo.exceptions.DemoException`*

Raised when a key id could not be found in a *`cache`*.

exception `cli_demo.exceptions.`**`OptionNotFoundError`**`(`*text=None*`)`

Bases: *`cli_demo.exceptions.DemoException`*

Raised when an *`Option`* instance could not be found in a *`registry`*.

exception `cli_demo.exceptions.`**`CallbackNotFoundError`**`(`*text=None*`)`

Bases: *`cli_demo.exceptions.DemoException`*

Raised when the *`callback`* of an *`Option`* instance has not been set.

exception `cli_demo.exceptions.`**`CallbackLockError`**`(`*text=None*`)`

Bases: *`cli_demo.exceptions.DemoException`*

Raised when the *`lock`* attribute of an *`Option`* instance is `True` but its *`callback`* does not accept a *key* argument.

exception `cli_demo.exceptions.`**`CallbackResponseError`**`(`*text=None*`)`

Bases: *`cli_demo.exceptions.DemoException`*

Raised when an *`Option`* instance is registered under an input function key but its *`callback`* does not accept a *response* argument.

Python Module Index

Index